Advances in Efficiency and Privacy in Payment Channel Network Analysis

by

Michelle Yeo

October, 2023

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Maria Ibáñez, Chair
Krzysztof Pietrzak
Krishnendu Chatterjee
Pavel Hubáček

Institute of Science and Technology Austria

The thesis of Michelle Yeo, titled *Advances in Efficiency and Privacy in Payment Channel Network Analysis*, is approved by:

Supervisor: Krzysztof Pietrzak, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Krishnendu Chatterjee, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Pavel Hubáček, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Signature: _____

Defense Chair: Maria Ibáñez, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Michelle Yeo
October, 2023

Signed page is on file

Abstract

Payment channel networks are a promising approach to improve the scalability bottleneck of cryptocurrencies. Two design principles behind payment channel networks are efficiency and privacy. Payment channel networks improve efficiency by allowing users to transact in a peer-to-peer fashion along multi-hop routes in the network, avoiding the lengthy process of consensus on the blockchain. Transacting over payment channel networks also improves privacy as these transactions are not broadcast to the blockchain.

Despite the influx of recent protocols built on top of payment channel networks and their analysis, a common shortcoming of many of these protocols is that they typically focus only on either improving efficiency or privacy, but not both. Another limitation on the efficiency front is that the models used to model actions, costs and utilities of users are limited or come with unrealistic assumptions.

This thesis aims to address some of the shortcomings of recent protocols and algorithms on payment channel networks, particularly in their privacy and efficiency aspects. We first present a payment route discovery protocol based on hub labelling and private information retrieval that hides the route query and is also efficient. We then present a rebalancing protocol that formulates the rebalancing problem as a linear program and solves the linear program using multiparty computation so as to hide the channel balances. The rebalancing solution as output by our protocol is also globally optimal. We go on to develop more realistic models of the action space, costs, and utilities of both existing and new users that want to join the network. In each of these settings, we also develop algorithms to optimise the utility of these users with good guarantees on the approximation and competitive ratios.

Acknowledgements

I would like to thank my supervisor Krzysztof Pietrzak for proposing the topic of blockchains and consequently introducing me to payment channel networks. I am also grateful to Krzysztof for allowing me immense freedom and encouragement to pursue my own research interests. I also thank my committee Krishnendu Chatterjee and Pavel Hubáček, with whom I had many great collaborations and learned a lot from them.

My PhD would not have been successful without my collaborators. I would like to especially thank two of my coauthors Stefan Schmid and Jakub Svoboda, with whom I have shared many exciting brainstorming and rush paper writing sessions (over lots of good food and bonfires). I hope that we will continue solving problems together for a long time. I am also grateful for my collaborators who hosted me for research visits – Pavel Hubáček, Christian Cachin, Ignacio Amores-Sesar, and Orestis Alpos – I enjoyed visiting the beautiful cities of Prague and Bern as much as I enjoyed our long and fruitful research discussions.

My PhD was also extremely smooth-sailing due to the support of two individuals at ISTA. Heartfelt thanks goes to Rita Six and Hitomi Tsukuda for helping me navigate various administration issues both at ISTA and in Austria.

My friends from ISTA – Alex, Linda, Sarath, Nataliia, Elizabeth, Guille and Bryan – and colleagues in my group made the PhD journey more enjoyable by sharing joint happiness and suffering. Special thanks to Ahad, Ben and Charlotte for lending me your names to replace "Alice", "Bob", and "Charlie". My PhD journey would not even have started without my fellow demonic occultist twin Elden. Thank you for pushing me to consider a PhD, and for teaching me the importance of always being slightly impudent. Many thanks also to my angelic twin Feryal for teaching me to constantly look near and far, and to find beauty in small things. I also thank Kim for single-handedly ensuring that I finish the PhD in a fashionable state, and for being a great recommendation system for tv shows.

Part of my thesis is dedicated to Djordje (who also replaced "Dave"). Thank you for inspiring me with your intelligence, courage, and strength. Because of you, I will always look for both mystery and cuteness in life.

I thank my family for believing in my dreams. I am especially grateful to my sister Mellissa for spoiling me by buying me endless amounts of snacks, gadgets and clothes whenever I am back in Singapore. You are simply the best! Finally, an important person who has been (mostly) silently supporting me the entire time is my mother Meliana. The remainder part of my thesis is dedicated to her. She is generous and kind, but also most unimpressed and hard to please. When I was playing Elden Ring and encountered the hardest boss Malenia I observed that Malenia is an anagram of Meliana. I did not manage to defeat Malenia, thus I hope this thesis serves as a gift to Meliana.

Funding. This work was funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385.

About the Author

Michelle Yeo obtained a bachelor degree in Statistics at the University of Chicago and a masters in Computational Statistcs and Machine learning at University College London. She worked for 2 years each at Morgan Stanley and Google DeepMind in London before joining ISTA in September 2018. Her research focuses on advancing payment channel network analysis. Her main research interests include blockchain scalability and security, algorithmic game theory and mechanism design, and decentralised finance. Her work has been published at premier venues in blockchain research (FC, AFT), distributed computing (SIROCCO, ICDCS), and cryptography (TCC) communities. Her work also won the best paper award at SIROCCO 2023.

List of Collaborators and Publications

1. Chapter 3 is based on "Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Michelle Yeo.[†] LightPIR*: Privacy-Preserving Route Discovery for Payment Channel Networks*. In **IFIP Networking 2021**"

2. Chapter 4 is based on "Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, Michelle Yeo.[†] *Hide & Seek: Privacy-Preserving Rebalancing on Payment Channel Networks*. In Financial Cryptography and Data Security - 26th International Conference, **FC 2022**"

3. Chapter 5 is based on "Stefan Schmid, Jakub Svoboda, Michelle Yeo[†]. *Weighted Packet Selection for Rechargeable Links in Cryptocurrency Networks: Complexity and Approximation*. In Structural Information and Communication Complexity - 30th International Colloquium, **SIROCCO 2023** (best paper award)"

4. Chapter 5 is based on "Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali-Maddah Ali, Stefan Schmid, Jakub Svoboda, Michelle Yeo.[†] *Online Admission Control and Rebalancing in Payment Channel Networks.*. In Corr Volume abs/2209.11936 (2022)"

5. Chapter 6 is based on " Zeta Avarikioti, Tomasz Lizurej, Tomasz Michalak, Michelle Yeo.[†] *Lightning Creation Games*. In 43rd IEEE International Conference on Distributed Computing Systems, **ICDCS 2023**"

The following list contains other works written during the PhD period but are not included in the thesis.

6. Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali-Maddah Ali, Stefan Schmid, Jakub Svoboda, Michelle Yeo[†]. *Route Discovery in Private Payment Channel Networks*. IACR Cryptol. ePrint Arch. 2021: 1539 (2021)

[†]Authors ordered alphabetically.

7. Samarth Tiwari, Michelle Yeo, Zeta Avarikioti, Iosif Salem, Krzysztof Pietrzak, Stefan Schmid. *Wiser: Increasing Throughput in Payment Channel Networks with Transaction Aggregation*. In 4th ACM Advances in Financial Technologies, **AFT 2022**

8. Pavel Hubáček, Jan Vaclavek, Michelle Yeo†. *Foundations of Fiat-Denominated Loans Collateralized by Cryptocurrencies*. In The 3rd Workshop on Decentralized Finance, **DeFi 2023**

9. Orestis Alpos, Ignacio Amores-Sesar, Christian Cachin, Michelle Yeo†.*Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks*. CoRR Volume abs/2307.02954. (2023)

10. Suvradip Chakraborty, Stefan Dziembowski, Malgorzata Galazka, Tomasz Lizurej, Krzysztof Pietrzak, Michelle Yeo†. *Trojan-Resilience Without Cryptography*. In Theory of Cryptography - 19th International Conference, **TCC 2021**

11. Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, Krzysztof Pietrzak. *Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement*. In 42nd IEEE Symposium on Security and Privacy, **SP 2022**

12. Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, Michelle Yeo†. *Inverse-Sybil Attacks in Automated Contact Tracing*. In Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track, **CT-RSA 2021**

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Quiet pond
a frog leaps into
the sound of water.

Matsuo Bashō

1.1 Prologue

In 2008, a mysterious, unknown entity named Satoshi Nakamoto created the Bitcoin protocol, which presents a completely decentralised solution that achieves distributed consensus. Since then, blockchains and cryptocurrencies have been rapidly gaining in popularity – just in the summer of 2022 the author of this thesis could pay for her coffee at a random cafe in Islington, London using Bitcoin.

Apart from becoming more mainstream and "cool", there are also some unique aspects of blockchain solutions that make them particularly useful for the betterment of society. Here, we highlight a few salient examples:

- the decentralised and private aspect of blockchains and cryptocurrencies make these solutions a necessary part of the lives of individuals living in societies where there is a lack of trust in the government or economic systems
- the transparency and security of blockchains makes it easy to achieve democracy and reduce corruption

- blockchains that support advanced smart contracts help streamline financial systems by automating the verification of complex financial protocols

Although blockchain protocols are certainly important and useful for society as a whole, these protocols, however, are notoriously inefficient. This is mainly due to the need to achieve consensus among all the participants in the blockchain, which can be a lengthy process, especially when compared to competing protocols in centralised finance. For instance, Bitcoin can only process an average of 7 transactions per second (compared to Visa's processing rate of 24000 transactions on average per second) [PD15]. As such, this presents a clear impediment to the adoption of blockchain solutions in everyday life situations.

In the light of these inefficiencies of blockchains, recent work have proposed solutions to address the scalability issue of blockchains, and can be broadly classified into two categories. Layer 1 solutions, as the name suggests, work directly on the blockchain layer and aim to improve the efficiency of the blockchain either by developing more efficient consensus protocols [KRDO17], or by sharding [LNZ+16, CDE+16]. Layer 2 solutions typically act as an overlay on top of the blockchain network, and aim to transfer the bulk of the payment load from the blockchain to the overlay layer, using the blockchain only as a last resort to settle disputes. Examples of layer 2 solutions are payment channel networks [PD15, DSW16, Rai17], which will be the focus of this thesis.

1.2 Payment Channel Network Analysis

Payment channel networks improve the scalability of blockchains by shifting the transaction load off-chain to payment channels. A payment channel acts as a ledger between two parties. However, instead of tracking the transaction history between the two parties, the payment channel simply records changes in the balances of both parties, debiting the balance of the payer and crediting the balance of the payee by the payment amount every time a payment is made from one party to the other. In this way, instead of achieving consensus among all participants of the blockchain for every transaction, only the two parties that are involved in each payment channel would need to agree on any new state of the channel, thus making it a lot faster to agree on transactions as compared to the blockchain. Additionally, transacting frequently over payment channels can be a lot cheaper compared to transacting on the blockchain. This is due to the fact that apart from the fees involved in opening and closing a payment channel, there are no fees involved when directly transacting with another party across a payment channel. This makes it potentially a lot cheaper compared to transacting on the blockchain, where every transaction incurs a transaction fee.

Although the analysis of payment channel networks spans several dimensions – privacy [TM20, PSSY21b, ABM+21], efficiency [RBS16, PZS+16b, RMSKG17, SVA+18, ABWW19, APS+22c, TYA+22], conceptual [DEFM19, AEE+20], game theoretic [AKWZ21a, ERE20b, ALW20, AHWW20, GHS21] etc. – two of the most important dimensions of payment channel network analysis are privacy and efficiency, which will also be the focus of this thesis. In the following, we summarise the main research directions, prior work, as well as important challenges on both the privacy and efficiency frontiers.

1.2.1 Privacy

Network monitoring and probing attacks. By design, payment channels are intended to be *transaction private*. This definition is best illustrated by the following example: consider two users u and v that open a payment channel between themselves and an eavesdropping adversary that can only monitor the blockchain. Apart from knowing that u and v are transacting with each other, as well as when the payment channel between u and v is initialised and closed (and accordingly, the respective balances of the users when these events happen), which are published on the blockchain, the adversary should not know any other transactions that occur between u and v off-chain over their channel as they are not published on the blockchain.

However, the same conclusion cannot be made if the adversary is allowed to additionally monitor or probe the network, especially if such actions are essentially cheap [KYP+20]. For instance, although channel balances are typically private, an adversary with nodes in the network can send out transactions of various sizes between adversarial nodes (in the guise of "transaction routing") to probe and estimate the balances of channels in the network [HNR+19, KYP+20, BNT22]. Moreover, the adversary does not even need to send out transactions in order to gather information about routing patterns and channel balances. Indeed, an adversary monitoring and routing payments in the network can also learn information about the source and receiver of transactions simply by tracing the hash used to lock payments when payments are routed using HTLCs [MMSK+17, KYP+20, TZS20a, TZS20b, KER21]. Even in the case where knowing the source or receiver of a transaction is arguably benign, for instance in the case of rebalancing where the source and the receiver have to be the same node, knowing some information about the transaction path reveals information about depleted channels, the transaction rate along certain channels, and consequently, the overall transaction demand in the network.

Private routing. To counter probing attacks, [BNT22] suggest creating multiple parallel channels between any two nodes and splitting transactions among these channels. In this way, the prober will only know the sum of all the channel balances but not how these balances are split among these parallel channels. Another approach is to

add some noise to channel balances and publish these noised channel balances in response to a routing query [TWFO20, DTZG22]. The amount of noise has to be cleverly selected so as to balance between privacy as well as utility, as too much noise could give false information to senders and ultimately hinder transaction routing. Thus, the protocol presented in [DTZG22] adds noise in a way such as to guarantee differential privacy against all probing queries in any given time window. Information leakage of the payment path typically occurs during the transaction execution phase, especially if routing is conducted using HTLCs. To counter this, several works have proposed some form of hashlock decorelation by rerandomisation of secrets along the payment path during the transaction execution phase [MMSK+17, MMS+19, TM20]. Finally, to address route discovery in an unknown or private network, where channels might be completely hidden, [ABM+21] proposed exploratory probing with message rerandomisation to improve unlinkability.

Private rebalancing. There are currently two main approaches to rebelancing payment channels in payment channel networks. The first method, and currently the main approach in the Lightning Network, finds cycles starting and ending with the user that wants to initiate the rebalancing process [Gitb]. This cycle-finding method is private in the sense that exact channel balances are not revealed to users outside of the channel. However, cycle-finding might ignore globally optimal rebalancing solutions. The second method introduced in [KG17a] formulates the rebalancing problem as a linear program and sends the linear program together with channel balance constraints to a randomly selected user or an external third party to solve for the globally optimal rebalancing. Although doing so achieves the globally optimal rebalancing solution, this method exposes private channel balances, resulting in a serious privacy violation.

1.2.2 Efficiency

Efficient routing. As mentioned previously, payment channel networks were designed with efficient payment making in mind, and indeed efficient transaction routing is one of the most well-studied areas in payment channel network research. Efficient transaction routing algorithms aim to address the problem of finding shortest (i.e., cheapest) paths in the network while ensuring a high transaction success probability. The difficulty of this problem lies in the fact that although the topology of the payment network, the total capacity of each channel, and the fee functions on each channel are publicly available, the way the channel capacity is split on both ends of the channel is private. As such, some transaction routes might fail during the execution simply because some channels do not have sufficient balance on one end to forward the transaction. Hence, it is common for senders in the network to try several transaction paths sequentially until one of them succeeds.

One approach to ensuring transactions occur with high success probability is to split payments into smaller amounts and send each unit individually, as smaller transaction units have a higher probability of being accepted by more channels. This is the approach adopted by [SVA+18] which also propose a multi-path routing algorithm to route these smaller payment units. Another approach is to shift the focus to the route discovery process and use well-connected nodes to perform the search, the idea being that well-connected nodes typically lie on many shortest paths. This method is used by [RBS16, MMSKM17, RMSKG17] which all employ landmark routing using these well-connected nodes as landmarks, as well as multiparty computation to secret share the balances of the channels to the landmarks. A similar technique is also employed by [PZS+16b] using beacon nodes. Finally, [DEFM19] introduces virtual channels as a way to bypass intermediaries in the routing process altogether.

Optimisation of decisions. Another important facet of efficiency in payment channel networks is optimisation, in particular, the optimisation of the decision making process faced by users in the network. Examples of some decisions a user in the network would have to face would be deciding how much funds to inject into the channel during channel creation, which transactions to accept to forward and which to reject, when to rebalance channels, and what fee function to allocate to channels so as to maximise profits. These decisions typically come with associated costs, and impact the future decision making process of not only the user that makes these decisions, but also the whole network. For instance, a myopic user that decides to impose larger fees on their channels compared to the network average in order to maximise short-term revenue might end up deterring potential senders from routing transactions through their channels and hence lose out on larger long-term revenue. In this thesis, we study this optimisation problem from two perspectives: (1) from the perspective of a user that is already inside the network, and (2) from the perspective of a new user that wants to join the network.

The problem of optimising some set of decisions of users that are already inside payment channel networks has been studied in the online setting against various classes of adversaries by [ABWW19]. Another approach to the problem is to look at the offline setting, where the goal is to design an optimal scheduler that schedules certain decisions in advance. This is the approach taken by [FNS21] where the scheduler decides when to close and reopen channels with more funds, and by [PN20a] where rebalancing is scheduled periodically based on a measure of the imbalance of channels. The study of the optimisation problem for a new user that wants to join the network has roots in network creation games by [FLM+03a], where the goal is to find a network connection strategy that optimises the utility of the new user. The model and analysis in [FLM+03a] was adapted in both [AHWW20] and [ERE20b] with different cost models and objectives that captures various aspects of the utility of users in payment channel

networks. Finally, an orthogonal but complementary direction to both these optimisation perspectives is the development and study of richer cost models. This is the case for channel costs in the work of [GHS21] which proposes a model that takes into account opportunity costs and interest rates, and also provides an analysis on when to create channels given the cost model and the transaction demand along a channel.

1.2.3 Challenges

Privacy vs. efficiency. Most existing routing and rebalancing algorithms in payment channel networks focus on either privacy or efficiency but not both. This problem is especially prominent when observing the proposed solutions for route discovery. The method proposed by [PZS+16b] using beacon nodes to discover paths leaks the source and destination to these beacon nodes, as well as the available channel balances for nodes on the payment path. The methods that employ landmark routing as proposed by [RBS16, MMSKM17, RMSKG17] find routes between senders and receivers efficiently. Nevertheless, the way these methods achieve privacy of channel balances and the endpoints of the path either rely extensively on multiparty computation, which is expensive, or strong assumptions about the amount and location of adversarially controlled nodes in the network. Finally, although the exploratory probing approach as proposed by [ABM+21] works well to discover shortest paths in a completely unknown or private network, it is inefficient as it potentially involves a significant portion of the network with every query.

The same challenge is also observed in rebalancing. Although the cycle-finding rebalancing procedure [Gitb] achieves privacy, it is extremely inefficient as users have to query each cycle sequentially to check if users in the cycle are willing and have sufficient funds to participate in rebalancing. The solution found by this method also lacks global optimality. On the other hand, the method proposed by [KG17a] achieves a globally optimal rebalancing solution, however it exposes a serious privacy loophole by leaking the balances of channels to a third party.

Modelling actions and costs. On the optimisation frontier, a crucial challenge is developing a model of actions and costs that strikes the balance between being realistic and efficiently solvable – an extremely rich model that accounts for a large action space and detailed costs would certainly conform better to real-life decision making in payment channel networks, but the corresponding optimisation problem would be completely intractable. For optimisation problems that focus on the setting of an existing user in the network, a common limitation among the proposed solutions is the restricted action space. For instance, [ABWW19] only focus on accepting or rejecting transactions and do not consider rebalancing or reopening channels with more funds. Additionally, the cost model for rejecting transactions in [ABWW19] only assigns a constant unit cost

of rejection for transactions of any size. This is unrealistic as the rejection cost of a transaction should take into account the opportunity cost of rejecting the transaction and consequently missing out on the associated transaction fee, which is a function of the transaction size. Similarly, [FNS21] and [PN20a] only focus on optimising a single action: when to optimally schedule reopening channels with more funds and rebalancing respectively.

Similar drawbacks can be observed in the case of optimisation problems that focus on the setting of a new user that wants to join the network. A common limitation of the models proposed in [AHWW20] and [ERE20a] is the assumption that users in the network transact uniformly at random with other users in the network. This assumption is often justified by tractability reasons as well as the fact that the distribution of transactions is hidden in payment channel networks. Nevertheless, this does not make this assumption any less unrealistic. Additionally, the model of channel creation costs in both [AHWW20] and [ERE20a] is also limited in the sense that they do not account for opportunity cost of the locked funds in the channels. In contrast, [GHS21] proposed a detailed model of channel creation costs which accounts for factors like opportunity costs and interest rates. However, their model of the utility of new users is incomplete as they do not take into account potential profits the new user can gain from routing transactions as well as expected fees from routing to other users that are not directly connected to the new user.

1.3 Thesis Goal

Given the aforementioned challenges, the goal of this thesis is twofold: the first goal seeks to address the shortcomings in existing routing and rebalancing algorithms in payment channel networks that either focus on efficiency or privacy but not both. The second goal of the thesis is to extend prior models for the optimisation of decisions for existing and new users in meaningful ways, as well as develop efficient optimisation algorithms for these models that achieve good approximation and competitive ratios.

To achieve the first goal, we eschew focusing on privacy during transaction execution as there already exist many efficient solutions that address this problem [MMSK$^+$17, MMS$^+$19, TM20]. Rather, we focus on route discovery, in particualar, the question of how to efficiently and privately *discover* shortest paths in payment channel networks. We first make the crucial observation that hub-labelling algorithms combine well with private information retrieval and can be used in tandem to achieve a solution that is both private, and has good asymptotic guarantees on efficiency. Additionally, we also propose a rebalancing protocol that is private (users only know the predecessor and successor in their rebalancing cycle, and no information about any channel's balances is known to users outside of the channel), and achieves a more optimal solution compared

to previous rebalancing proposals like [Gitb, KG17a].

For the second goal, we first propose an enlarged model of the action space that includes rebalancing channels as well as reopening channels with more funds injected in the setting of optimising the utility of an existing user in the network. We also account for the opportunity cost of rejecting transactions in our cost model. In the setting of optimising the utility of new users that want to join the payment channel network, we also extend prior models of transactions and utilities in a more realistic fashion by dropping the uniform assumption on the transaction distribution as well as adding the opportunity cost of locked funds. In both of these settings, we show that even when working with these realistic and more complex models, we can still develop efficient algorithms with good guarantees on the approximation or competitive ratios.

1.4 Outline and Contributions

Roadmap. Chapter 2 provides the necessary preliminaries crucial to the analysis of payment channel networks in this thesis. The main body of the thesis can be broken down into two parts. The first part, comprising of Chapters 3 and 4, study the problem of efficient and private routing and rebalancing on payment channel networks. The second part, comprising of Chapters 5 and 6, focus on the developing realistic models of the utility, action space, and costs faced by users that are already inside a payment channel network and new users that are not yet inside the network respectively. These chapters also develop competitive algorithms for optimising the utility of these users. Each of these chapters is pretty much self-contained and can be read independently. Finally, we conclude the thesis by contextualising our contributions as well as outlining future research directions in Chapter 7.

Contributions. Contributions in each of the main technical chapters of this thesis can be summarized as follows:

- Chapter 3 considers the problem of both efficient and private payment route discovery in payment channel networks. This goal of this chapter is to present a solution for efficiently finding the cheapest path for routing transactions, but also ensuring that no information about the shortest path query is leaked. To the best of our knowledge, prior work on this topic focuses on only either the efficiency or privacy aspect but not both. Thus, we present the first efficient and privacy-preserving route discovery mechanism for payment channel networks. Our protocol uses techniques from hub-labelling to boost efficiency, and private information retrieval to guarantee privacy, and we show that the precise combination of both of these techniques is critical to boosting the asymptotic efficiency of our protocol.

We also present an empirical evaluation of the efficiency of our approach on real-world Lightning Network snapshots.

- Chapter 4 considers the problem of efficient and private rebalancing of channels in a payment channel network. Unlike the majority of rebalancing approaches that use cycle-finding, which could potentially ignore globally optimal solutions, we present a protocol that is both privacy-preserving and also finds globally optimal rebalancing solutions. The main idea behind our protocol is to formulate the rebalancing optimisation problem as a linear program. The resulting flow computed from the linear program is further decomposed into cycles which can be efficiently executed using HTLCs. Finally, we use multiparty computation (MPC) to solve the linear program and decompose the flow into cycles so as to preserve the privacy of channel balances.

- Chapter 5 considers the optimisation problem an existing user in a payment channel network would face: how should a user handle incoming transactions and manage the funds in their channels to avoid channel depletion? We study the problem in both the offline and online setting. In the offline setting, we focus on the decision problem of how much capacity to inject into a channel and, given the capacity injected as well as a transaction stream along a single link as input, whether to accept or reject transactions over the link. We show that the this problem is NP-hard. Our main result in the offline setting is an efficient algorithm that decides on the capacity injected into the channel as well as which transactions to accept or reject that approximates the optimal solution with a constant approximation ratio. In the online setting, we still focus on the problem over a single link, but we expand the action space to include reopening the channel with more funds injected, as well as rebalancing the channel. Our main result is a $7 + 2\lceil \log C \rceil$-competitive online algorithm, where C denotes the length of the rebalancing cycle (≤ 4 in most cases in the Lightning Network). The offline and online variants of the problem are also intrinsically connected – a fundamental component of our main online algorithm is that it uses an optimal offline algorithm as an oracle, and our offline approximation algorithm shows precisely how to approximate this.

- Chapter 6 considers a complementary optimisation problem to the problem posed in Chapter 5: how does a new user optimise their utility when connecting to a payment channel network? Unlike in Chapter 5, the focus of the optimisation problem is not transaction handling and channel management, but rather channel creation (i.e., who to connect to, how many channels to create, how much funds to lock into each channel etc.). The main contribution in this chapter is a novel and realistic model of utilities, which takes into account a realistic transactions distribution in the network. We also prove some theoretical limitations

in Theorems 6.4.1, 6.4.2, and 6.4.3 regarding the objective function, and show that these limitations render the objective function impossible to optimise with good guarantees on the approximation ratio with the available optimisation algorithms. However, we also show that with some meaningful constraints, we can develop optimisation algorithms with good guarantees on the approximation ratio in these constrained optimisation settings. Our secondary contribution in this chapter is a topological study of stable networks under our realistic model of utilities and transactions distribution, and we show in Theorem 6.5.2 that the star topology is stable under some reasonable conditions.

CHAPTER 2

Preliminaries

This chapter presents the requisite information regarding blockchains and payment channels, which will be assumed as background knowledge throughout the rest of the thesis. We start by fixing mathematical notation, and then briefly describe some underlying cryptographic primitives. Then we present a brief overview of blockchains, payment channel networks, and related protocols.

2.1 Mathematical Preliminaries

We begin by defining set and vector notation that will be used throughout the thesis. We will then define two useful properties (submodularity and monotonicity) of functions that aid in constructing efficient optimisation algorithms with strong approximation guarantees. These notions will be used heavily in the technical sections of Chapter 6.

Sets and sequences. We use $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}$, $\mathbb{R}^{\geq 0}$ to denote the natural numbers, natural numbers extended with 0, reals, and nonnegative reals respectively. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. For two sequences S_1, S_2, we write $S_1 \prec S_2$ if S_1 is a prefix of S_2.

Vectors. We use boldface symbols to denote vectors. For a set S, $n \in \mathbb{N}$, and $\mathbf{x} \in S^n$, we use $\mathbf{x}[i]$ to denote the ith component of $\mathbf{x}$, for $1 \leq i \leq n$.

Monotone functions. Let S be some set of size n and let f be some real-valued set function $f : 2^S \mapsto \mathbb{R}$. We say that the function f is *monotone* if for all sets $A \subseteq B \subseteq S$, $f(A) \leq f(B)$.

Submodular functions. Let S be some set of size n and let f be some real-valued set function $f : 2^S \mapsto \mathbb{R}$. For a $A \subseteq S$ and an element $x \notin A$, we denote the *marginal gain* of adding x to A by $MG_A(x)$ and define it as $f(A \cup x) - f(A)$. We say f is *submodular* if for all sets $A \subseteq B \subseteq S$ and for an element $x \notin B$, $MG_A(x) \geq MG_B(x)$. Note that if f is a utility function, submodularity captures the natural notion of diminishing returns where the marginal gain of adding a new element to a given set is smaller when the set is larger.

2.2 Cryptographic Preliminaries

In the protocols introduced in the next section, as well as for the rest of the thesis, we assume all processes (honest and adversarial) are computationally-bounded. That is, they are restricted to run in probabilistic polynomial time (PPT).

Negligible functions. Let $f : \mathbb{N} \mapsto \mathbb{R}$. We say f is negligible if for every $c \in \mathbb{N}$, there is an n_c such that $f(n) \leq \frac{1}{n^c}$ for all $n \geq n_c$. That is, f approaches 0 faster than any inverse polynomial.

Hash functions. Hash functions form the foundation of the protocols we will describe in the next section, thus we will first define a hash function (taken from [KL07])

Defintion 2.2.1. *(Hash function.) A* hash function *is a pair of probabilistic time polynomial algorithms (Gen, H) fulfilling the following:*

- *Gen is a probabilistic algorithm that takes in a security parameter input 1^n and outputs a key s.*
- *there exists a polynomial ℓ such that H is a deterministic polynomial time algorithm that takes in s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{\ell(n)}$*

The key parameter s specifies a particular hash function H^s from a family of valid hash functions. As the protocols presented in the rest of thesis are oblivious to the exact hash function H^s (we simply assume that H^s is a valid hash function), we simply drop the key s when we denote hash functions for the rest of the thesis in order to reduce excessive notation. An important property that the hash functions used in the protocols in the next section need to satisfy is *collision resistance*. That is, a computationally-bounded adversary can find two distinct strings x, x' such that $H(x) = H(x')$ only with negligible probability. Formally, we define the following collision-finding game (taken from [KL07]):

Collision-finding game $\mathsf{HashColl}_{\mathcal{A},\Pi}(n)$**:**

- A key is chosen: $s \longleftarrow \mathsf{Gen}(1^n)$
- The adversary $\mathcal{A}$ is given s and needs to output a pair x, x': $(x, x') \longleftarrow \mathcal{A}$
- The output of the experiment is 1 if $x \neq x'$ and $H(x) = H(x')$.

We say a hash function $\Pi = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that $\mathbb{P}[\mathsf{HashColl}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$.

2.3 Blockchain and PCN Preliminaries

Blockchain. A blockchain is a distributed ledger whereby data is aggregated into blocks and then appended onto the ledger. Consensus over the state of the ledger is achieved via consensus mechanisms such as proof of work [Nak08], proof of stake [DGKR18], proof of space and time [CP19] etc. Note that the process of achieving consensus can be time-consuming, as it potentially involves all parties in the blockchain.

Blockchain protocols should satisfy the following properties (taken from the Bitcoin backbone protocol [GKL15]). Note that the following definitions assume synchronous communication, hence, an underlying blockchain protocol that proceeds in rounds. Also, all participants in the protocol are either honest or adversarial, and that the fraction of adversarial participants is bounded.

Blockchain properties:

- Common prefix (parameterised by $k \in \mathbb{N}$): for any two honest parties P_1, P_2 that adopt chains C_1, C_2 at rounds $t_1 < t_2$ respectively, it holds that $C_1^{-k} \prec C_2$, where C_1^{-k} denotes chain C_1 with the most recent k blocks removed. This property is also known as "persistence" – any block that is k deep in an honest party's chain should also "persist" in all other honest parties' chains.
- Chain quality (parameterised by $\mu \in [0, 1]$ and $\ell \in \mathbb{N}$): for any honest party with chain C, it holds that for any sequence of consecutive blocks of length ℓ, the fraction of honest blocks in the sequence is at least μ.
- Chain growth (parameterised by $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$): for any honest party with chain C, after any s consecutive rounds it should adopt a chain that is at least $\tau \cdot s$ longer than C.

Intuitively, the common prefix property states that there is some "common data history" in the blockchain that all honest parties agree on. The chain quality property states

that as long as an honest party looks at some subsequence of its chain that is long enough, the number of adversarial data is bounded, thus the "quality" of the chain is sufficient. Finally, chain growth states that the blockchain should grow at a sufficiently fast rate.

Payment channels. Adding transactions to the blockchain can be slow as consensus over the state of chain needs to be achieved among all honest parties in the blockchain. Payment channels offer an alternative method for two parties to transact with each other in an efficient, private, and secure manner. To open a payment channel, two users u, v first need to make an *initialisation transaction* on the blockchain. This records the amount of funds that u and v (say u locks a units and v locks b units) each contribute as capital to fund the payment channel. Note that these funds are "locked" into the channel in the sense that they can only be used between u and v to transact with each other over this channel and they cannot be used somewhere else as long as the channel remains open. At any point in time, the *balance* of u (resp. v) in the channel is simply the amount of funds u (resp. v) has on their end in the channel, and we denote it by balance(u, v). (resp. balance(v, u)). For example, the initial balance of u in the channel is a and the balance of v in the channel is b. The *state* of the channel at any point in time is simply the balance of both parties in the channel, and the capacity of the channel refers to the sum $a + b$ of these funds.

Once the payment channel is open, u and v can transact with each other by simply updating the channel balances in favour of the other party by the transaction amount, so long as balance$(u, v) \geq 0$ and balance$(v, u) \geq 0$. Concretely, any party that wants to make a transaction would propose a new state of the channel which involves a different split of the capacity on both ends. Both parties will then have to sign the new state of the channel (2 of 2 multi-signature). If one of the parties does not sign a new state, the state of the channel remains at its current one. Refer to Figure 2.1 for an example of payments along a payment channel and the updated balances of each user in the channel after each payment amount.

A payment channel can be closed at any point in time, releasing the locked funds in the channel for use in some other application. This involves a *closing transaction* on the blockchain to record the final balances of both parties in the payment channel. Thus, with just two blockchain transactions, any two users can make an unlimited amount of transactions with each other as long as the transactions respect the balances of each user in the channel. Since the only transactions recorded on the blockchain are the initialisation and closing transactions, payment channels offer users a large degree of privacy as all their intermediate transactions in the channel are only known to both users in the channel. Finally, payment channel security stems from the fact that both users in the channel have to sign (and hence agree on) every new channel state.

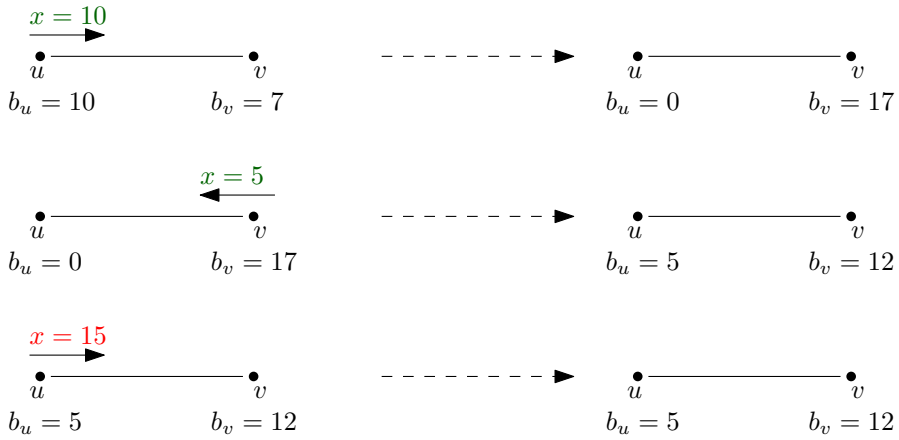


Figure 2.1: Example of payments (denoted by x) between users u, v along their payment channel and their updated balances (denoted by b_u and b_v) in the channel after the payments. The final payment amount of $x = 15$ cannot happen as it would result in a negative balance of u in the channel.

Payment channel networks. A payment channel network (PCN) is simply a network of payment channels. Typically, we represent a PCN as a directed graph $G = (V, E)$ where the vertices of G are the users in the PCN and the directed edges are payment channels between two users. We represent bidirectional payment channels (where funds can move in both directions along a payment channel) between two users $u, v \in V$ as two directed edges (u, v) and (v, u). Two examples of commonly used PCNs are Bitcoin's Lightning Network [JP], and Ethereum's Raiden Network [Rai17].

Routing in PCNs. Users who are not directly connected by a channel in a PCN can still transact with each other if they are connected by a path of payment channels. This is done in a multi-hop fashion with the intermediary nodes along the payment path forwarding the payment amount from sender to receiver. These intermediary users typically charge a fee for forwarding the transaction that depends on the transaction amount. For a transaction to be successful, the sender has to first send enough money to cover both the desired payment amount and all the fees charged by each user on the payment path. That is, suppose user s wants to send x coins to user r along a payment path $p = \{(s, u_1), ..., (u_k, r)\}$. Then s must send $x + \sum_{i=1}^{k} \text{fee}(u_i)$. Secondly, the balance of each user along the path must be large enough to forward the payment amount together with fees. Then for each user u_i on p, $\text{balance}(u_i, u_{i+1}) \geq x + \sum_{j=i+1}^{k} \text{fee}(u_j)$. Although the channel capacities are typically public information, the individual balances on each end are private; so senders typically have to try different payment paths until one of them succeeds.

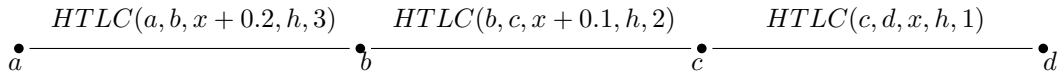


Figure 2.2: Using HTLCs to enforce an atomic payment between users a and d.

Hash timelock contracts (HTLCs). A desired guarantee for payment routing through a path in a PCN is atomicity, i.e., for all users along the path, either all of them update their balances or none of the balances in the path get updated. This is enforced in the Lightning Network using HTLCs [JP]. An HTLC ($HTLC(u, v, x, h, t)$) is a smart contract between any two users u and v that locks some amount of coins x using a hash output h from a global hash function $H(\cdot)$ and a timelock t. To get the locked funds, v has to produce the preimage r to the hash $h = H(r)$ within time t, upon which the locked funds will be released to v. If v cannot do so within the time limit, u can claim the locked funds. Payment path atomicity is enforced using HTLCs for each channel on the path with the same hash value (determined using a secret chosen by the receiver on the path), but with decreasing timelock values from sender to receiver to guarantee the security of funds. Figure 2.2 depicts an example of how two users a and d who are not directly connected by a payment channel in the PCN can use HTLCs to send an amount of x coins along the payment path $\{(a, b), (b, c), (c, d)\}$. We assume the intermediate users on the payment path c and d each charge 0.1 coins for forwarding the payment of size x. To do so, the receiver of the payment d first samples a random secret r and sends $h = H(r)$ to a. a creates a HTLC between a and b, locking in $x + 0.2$ coins to account for the intermediate fees along the payment path. The hashlock for this HTLC is h and the timelock is 3 units of time. Then, b creates a HTLC between b and c with the same hashlock but with payment amount $x + 0.1$ and timelock 2. Finally, c creates a HTLC with d with payment amount x and timelock 1. To execute the payment, d simply reveals the secret r to c within 1 unit of time. c then reveals the secret to b, who then reveals the secret to a. Note that intermediate users are incentivised to reveal the secrets to the previous user on the payment path within the time frame allocated by the timelock value as this allows them to enforce the payments from the previous user, which consequently ensures that they do not lose money.

Rebalancing depleted channels. We say a payment channel is depleted if the balance of a user in the channel is close to or at 0. If the user with the depleted balance in the channel does not wish to transact further with their channel partner, they can simply close the channel and withdraw their funds. However, if they wish to continue transacting with their channel partner, they would need some means of increasing their balance in the channel. The most obvious way of doing so is simply to close the channel and then reopen a new channel with more funds injected (note that we call this "recharging" the channel in Chapter 5). However, this would involve two

additional blockchain transactions (the closing and initialisation transactions), as well as added latency from consensus in the blockchain. A more cost-efficient solution to this problem is to rebalance the channel. Rebalancing involves shifting funds from channel to channel in a cyclic fashion while maintaining the invariant that the *total balance* of the involved users, that is, the sum of all their balances on their incident channels, remain the same. Rebalancing is often coupled with HTLCs to ensure atomicity of the procedure – either the rebalancing happens, or no rebalancing happens. Either way, no user loses money in the process. As rebalancing is conducted entirely off-chain, it provides an efficient alternative to reopening new channels. Refer to Figure 2.3 for an example of how rebalancing is done among 3 users in a PCN. In this example, node u can rebalance their close-to-depleted channel (u, v) by 10 coins by shifting the funds in a cyclic fashion beginning with the (u, w) channel. That is, u first sends 10 coins to w along (u, w). Then w sends the 10 coins to v along (w, v). Finally, v sends the coins back to u along (v, u). Note that no involved user gains or loses any coins in the process – they simply reallocate their funds to different channels.

Figure 2.3: The graph on the left depicts three nodes u, v, w connected in a cycle in a PCN. The red numbers by each channel represent the balance of a node in a certain channel. The graph on the right depicts the updated balances of each node on each channel after rebalancing.

CHAPTER 3

Efficient and private routing on PCNs

> The journey, not the destination matters.
>
> T.S. Eliot

This section is based on the following publication:

- Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Michelle Yeo.† LightPIR*: Privacy-Preserving Route Discovery for Payment Channel Networks*. In **IFIP Networking 2021**.

3.1 Introduction

Scalable route discovery mechanisms. PCNs present a promising solution to the scalability problem of blockchains. However, PCNs also introduce new challenges to the issue of efficiently transacting with other users. In particular, since users determine transaction forwarding fees in their channels, different transaction routes in the PCN would result in different fees. Thus, nodes require scalable mechanisms to discover "short" (i.e., low-cost) routes. To ensure scalability and support lightweight nodes (e.g., wallets) in the network, these route discovery mechanisms must be efficient, both in terms of disk space and control traffic. Especially since payment channel networks can be more dynamic compared to classic communication networks (e.g., channels and

†Authors ordered alphabetically.

liquidities can change over time) [GMSR+19], it is crucial that resource-constrained nodes do not have to store and maintain the entire network topology to be able to compute a route toward the transaction's target. Thus, over the last years, several innovative approaches for a more scalable routing in payment channel networks have been proposed [PZS+16b, RMSKG17, Wee19, MMSKM17, SVA+18, TZS20b]. For example, Lightning introduced the notion of trampoline nodes [Wee19] to provide route discovery services to light-weight clients. The trampoline nodes are more powerful nodes which know the entire topology (e.g., using gossiping); a wallet then just needs to know how to reach the route server nodes in its neighborhood and can then request the desired route.

Privacy. Besides scalability, payment channel networks also need to provide a high degree of privacy: it is critical that other nodes in the payment channel networks cannot learn who transacts with whom. In order to provide anonymity in the transaction routing, payment channel networks typically implement onion routing [DMS04, RMSKG17, MMSKM17]. However, while onion routing provides security for the routing process, it does not provide security for the route *discovery* process. In particular, by requesting shortest paths to certain destinations from the route discovery servers (e.g., the trampoline nodes in Lightning), a node may reveal critical or sensitive information about its planned financial transactions [TS20, NFSD20].

Prior approaches and drawbacks. Prior work on route discovery on paymwent channels is typically only focus on either the efficiency [RBS16, RMSKG17, MMSKM17, SVA+18, PZS+16b] or privacy [WZPM16, TM20] aspect of the problem. To the best of our knowledge, the approach presented in this chapter is the first approach to address both of these concerns simultaneously.

Contribution. Motivated by this shortcoming of state-of-the-art payment channel networks, we initiate the study of the design of scalable privacy-preserving route discovery mechanisms for payment channel networks in this chapter. The main contributions of this chapter is the first scalable and privacy-preserving route discovery mechanism for payment channel networks, *LightPIR**. *LightPIR* provides information theoretic security guarantees, and is evaluated both analytically as well as empirically, using the Lightning network. We show that our approach is efficient and practical, and may hence be useful also in other networks where the security of route discovery is critical. Concretely, we tested our approach with different snapshots of the Lightning network taken over a period of two years and our heuristic produces a small hub database (about 2MB), which at the moment is an improvement by an order of magnitude compared to the

*The name stands for private information retrieval for Lightning-like networks.

size of the complete network. With the growth of the network, this discrepancy will change further in favor of approaches similar to ours.

3.2 Related Work

While the security of blockchains, e.g., related to the underlying consensus protocols [GKW+16, WG16] but also related to the interconnecting network [GRKC15], has been extensively investigated, much less is known about the security of the recent concept of payment channel networks. The security aspects of payment channel networks considered so far mainly revolve around connectivity [RMT19] and the multi-hop routing of payments [TZS20b, MMSK+17]. We refer the reader to the survey [GMSR+20] for an overview of specific threats related to hot wallets, mass exits, synchronizing protocols, among others.

Prior work on route discovery primarily focused on efficiency. For example, SpeedyMurmurs [RMSKG17] (which extends VOUTE [RBS16]) and SilentWhispers [MMSKM17] rely on clever protocols to speed up route discovery by two orders of magnitude, while maintaining the same success ratio. Another interesting approach is pursued in the SpiderNetwork [SVA+18]: the payments are split into units and the route discovery algorithm routes each of them individually (similarly to a packet-switched network). This method however also does not account for privacy and cost effectiveness. To trade off high throughput and probing overhead Flash [WXJW19] distinguishes between mice and elephant payments, splitting elephant flows across multiple flows. Flare [PZS+16b] proposes to improve scalability of route discovery by determining beacon nodes, an approach which also motivates our model. To overcome the cost overheads of multi-hop routing, thee off-chain channel network Perun [DEFM19] introduces a notion of virtual payment channels; while this allows to avoid intermediaries for each payment, it does not solve the discovery problem.

Private route discovery on street networks has been explored in [WZPM16]. Their work is similar to ours in that they also use a PIR protocol to privately retrieve shortest path entries in a database. The main difference is the database compression technique: in [WZPM16] the authors develop a compression algorithm specific to the grid-like structure of street networks in North America (i.e. each node has degree 4 corresponding to the 4 cardinal directions), whereas we use hub labelling which has good compression guarantees on generic networks and we develop an additional heuristic based on the topology of the Lightning network to get small hubs set sizes.

Our work is motivated by recent empirical work demonstrating issues with confidentiality during the route discovery process in the Lightning network [TS20, NFSD20]. To the best of our knowledge, the only work suggesting a more secure route discovery

mechanism is MAPPCN [TM20], which however does not account of the information leaking problem addressed in this work.

Our work also builds upon classic works on efficient shortest path computations on extremely large networks [GH05, Lau06, Gut04]. Of particular interest in our context are algorithms based on transit node routing (TNR) [BFM+07], which focus on networks in which a small set of vertices covers most shortest paths, i.e., networks with small so-called highway dimension. Recently, Abraham et al. [AFGW10, ADF+11] proved theoretical guarantees on the efficiency of common shortest path search algorithms for graphs which satisfy this property. The hub-labelling algorithm [ADGW11, ADGW12] is a variant of TNR and it was shown to be significantly faster compared to other state of the art shortest path algorithms when tested on real world road networks [ADGW11]. In this chapter, we build upon this approach to scalable routing, tailoring it to the specific topological properties of payment channel networks.

Information theoretic PIR protocols were first introduced by Chor et al. [CKGS98]. With 2 servers and a simple linear bit summation scheme, they achieve perfect privacy with a communication complexity of $\mathcal{O}(n^{1/3})$. Since then, follow up work in this area has mainly focused on lowering the communication complexity [BI01, BIKR02b, DG16, Efr12, DHS14]. Recently, Beimel et al. [BIKR02a] achieved a communication complexity of $n^{\mathcal{O}(\frac{\log\log k}{k\log k})}$ with k servers using locally decodable codes. Under the conjecture that there are infinitely many Mersenne primes, Yekhanin [Yek08] removed the dependency on large k in the communication complexity bound and achieved a communication complexity of $n^{\mathcal{O}(\frac{1}{\log\log n})}$ for a 3 server PIR protocol. Although these newer protocols achieve a much lower communication complexity compared to the original protocol, the techniques used in these protocols are also much more complicated. As these benefits from a lower communication complexity are not that pertinent to us, we focus on simpler PIR protocols which are similar to [CKGS98] in this chapter.

PIR protocols have already been successfully deployed in many contexts, such as DNS [ZHS07], private presence service [BDG15], and private retrieval of security updates [Cap13]. However, we are not aware of any applications of PIRs in the context of payment channel networks.

3.3 Problem Statement

We model a payment network by a public directed graph $G = (V, E)$, where each edge $e \in E$ has some cost $c(e) \geq 0$. The nodes are the clients and an edge $e = (u, v)$ means there's a channel available from u to v with transaction cost $c(e)$. We reserve the letters $n = |V|$ to denote the number of vertices, $m = |E|$ to denote the number of edges, $\delta = |E|/|V|$ to denote the average degree and d for the edge length of the

longest shortest path in G (if all edge costs are identical this is just the diameter). We assume the nodes are labelled by λ-bit strings, while $\lambda = \lceil \log n \rceil$ is sufficient, we leave this as a parameter as one might want to use longer labels in practice.

Below we give our definition of a *private route discovery mechanism*. The definition is very similar to the classic definition of private information retrieval (PIR): on the one hand, is more restricted as we only consider the particular problem of retrieving shortest paths; on the other hand it is more general as the client is not just supposed to recover some database entry, but can perform a more general computation on the retrieved answers to compute the path.

3.3.1 Private Route Discovery Mechanism (PRDM) Definition

The involved parties are a content provider $\mathcal{CP}$, a set of servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$ and a client $\mathcal{C}$.

The content provider $\mathcal{CP}$ gets as input a graph $G = (V, E)$ and processes it into t databases $\mathsf{DB}_1, \ldots, \mathsf{DB}_k$. DB_i is then sent to $\mathcal{S}_i$.

The client $\mathcal{C}$ on input two nodes $s, t \in V$ can now interact with the servers and the *correctness property* we require is that $\mathcal{C}$'s final output is a shortest path from s to t in G (assuming $\mathcal{CP}$ and the $\mathcal{S}_i$'s follow the protocol).

The *security guarantee* we require is (ℓ, k)-privacy: the view of any of the $\ell > 0$ out of the k servers should not reveal any information about the query source and target s, t.

We are interested in finding *efficient* schemes, that is, the scheme has to scale well with the potential growth of the Lightning network.

Let us stress that once we learn a shortest path in a payment network like Lightning (where shortest will typically mean the path with lowest fees), it's not guaranteed that we can actually route a payment through that path as one endpoint on a channel on the path might not hold enough balance to perform the transfer. As the exact balances on channels are typically private information (if they were not, one could learn about transfers by just observing how balances at the endpoints change), we cannot solve this problem by just using public information.

3.4 Possible Approaches and Drawbacks

We start by giving a description of several schemes that fulfill both the correctness property and the security guarantee as described above but have varying degrees of efficiency. Note that for this problem efficiency can be defined in terms of several costs, namely, the amount of storage incurred by the server or the client, the amount of

	Server storage	Communication	Server computation	Client computation	Client storage
Trivial solution	$\mathcal{O}(n\lambda\Delta)$	$\mathcal{O}(n\lambda\Delta)$	$\mathcal{O}(1)$	$\mathcal{O}(m + n\log n)$	$\mathcal{O}(n\lambda\Delta)$
APSP solution	$\mathcal{O}(n^2d\lambda)$	$\mathcal{O}(n^2d\lambda)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2d\lambda)$
APSP + PIR	$\mathcal{O}(n^2d\lambda)$	$\mathcal{O}(n\sqrt{d\lambda})$	$\mathcal{O}(n^3 + n^2d\lambda)$	$\mathcal{O}(n\sqrt{d\lambda})$	$\mathcal{O}(1)$
APSP + Hub Labelling (HL)	$\mathcal{O}(nhd\lambda)$	$\mathcal{O}(nhd\lambda)$	$\mathcal{O}(n^3 + n^2\max(d,\log n)\log d)$	$\mathcal{O}(h)$	$\mathcal{O}(nhd\lambda)$
Our solution (APSP + PIR + HL)	$\boldsymbol{\mathcal{O}(nhd\lambda)}$	$\boldsymbol{\mathcal{O}(\sqrt{nhd\lambda})}$	$\boldsymbol{\mathcal{O}(n^3 + nhd\lambda + n^2\max(d,\log n)\log d)}$	$\boldsymbol{\mathcal{O}(h + \sqrt{nhd\lambda})}$	$\boldsymbol{\mathcal{O}(hd\lambda)}$

Table 3.1: Asymptotic server and client side storage, communication, and computation cost for the trivial solution, the APSP solution, the APSP solution with a PIR, the APSP solution with Hub Labelling, and our solution which is the APSP solution with a PIR and Hub Labelling. n is the number of vertices in the graph, m is the number of edges, Δ is the maximum degree of each vertex, λ is the length of the binary representation of each vertex, d is the length of the longest shortest path in the graph, and h is the maximal hub set size.

computation involved on the server or the client sides in order to get the final output, and the amount of data communicated between the client and the server. The amounts of storage incurred by the server and the client are most crucial parameters and our main focus in the following. Another important cost is the amount of computation the client incurs in order to retrieve the desired shortest path, as client-side machines are typically not as powerful as server-side machines. We start with discussing the most naive solutions, gradually augmenting them with simple strategies to improve their efficiency. To provide a summary of the efficiency of each method, we present all the relevant asymptotic costs described above for each method in Table 3.1.

3.4.1 Trivial Solution: Downloading Entire Graph

A trivial $(1, 1)$-private solution is to simply have the client download the entire graph and compute the shortest path locally. Although the server does not need to perform any computation for this solution, the communication from server to client is significant, i.e., the entire graph of size $\approx \lambda \cdot n \cdot \delta$, and the client has to perform a non-trivial computation (e.g., Dijkstra which is $\mathcal{O}(m + n\log n)$) to retrieve the desired shortest path. In addition, the client also has to store the entire graph. Taking into account the fact that in most cases clients only sporadically send small amounts of queries, and also that the underlying network needs to be frequently updated to account for new users and channels, it is useful to push the heavy computation and storage burden to the server; the latter typically runs on machines with superior hardware and storage capacity.

3.4.2 Server Computes All Pairs Shortest Paths

A natural $(1,1)$-private solution which pushes the burden of computation onto the server is for the server to precompute and store all the all pair shortest paths (APSPs). This solution gives a database with $N = n(n-1) \approx n^2$ entries (every pair of nodes) with entries of length $L = \lambda \cdot d$, i.e., the λ bit labels of the $\leq d$ nodes for each path.

Naturally the amount of computation the server would have to do to compute these APSPs ($\mathcal{O}(n^3)$) increases compared to the previous trivial $(1,1)$-private solution where the server does not have to perform any computation. Nevertheless this factor is not as crucial as the stored database size ($\mathcal{O}(n^2 d\lambda)$), which grows quadratically with n. Even for the moderate sized Lightning network with $n \approx 7000$, this gives an almost $2TB$ database (with $d = 9$ and $\lambda = 20$ bit labels).

In addition, to maintain privacy, the client cannot simply query the server for the desired index, as this blatantly reveals the path information. The client would therefore have to download the entire database from the server to hide their query. Thus, the price of a constant client-side computation cost is an additional client-side storage and communication cost of $\mathcal{O}(n^2 d\lambda)$ (see Table 3.1).

3.4.3 Trivial PRDM from Private Information Retrieval (PIR) Using APSPs

In a (ℓ, k)-private information retrieval (PIR) scheme we have k servers, each holding a copy of a database DB which contains N entries, each L bits long. A client $\mathcal{C}$ who want to learn DB$[i]$ for some i, computes queries q_i to every $\mathcal{S}_i$, and then can (efficiently) compute DB$[i]$ from the answers $a_1, \ldots, a_k$. The security property (ℓ-privacy) requires that any union of ℓ servers cannot learn anything about the query i.

The paper introducing PIR [CKGS98, Corollary 4.2.1.] has a particularly simple $(1,2)$-private PIR where the communication complexity is just $4\max\{L, \sqrt{N \cdot L}\}$, in particular, that's $4L$ if $L \geq N$.[†]

We can get a (ℓ, k)-private mechanism using a (ℓ, k)-private information retrieval on top of the APSP solution. Using a PIR protocol similar to [CKGS98] and assuming that the storage size of each shortest path is smaller that the total number of entries in the database, i.e. $d \cdot \lambda < n^2$, this solution reduces the communication cost between the client and servers from $\mathcal{O}(n^2 d\lambda)$ to $\mathcal{O}(n\sqrt{d\lambda})$ as shown in Table 3.1. In addition, the client now does not need to download and store the database but just has to perform

[†]In this construction we think of DB as an $L \times N$ matrix when $L \geq N$, $q_1 \in \{0,1\}^N$ is random and $q_2 = q_1 \oplus e_i$ where e_i is the zero vector with the ith position set to 1. From the answers $a_i = \mathsf{DB} \cdot q_i$ the client computes $a_1 \oplus a_2 = \mathsf{DB} \cdot e_i = \mathsf{DB}[i] \ni \{0,1\}^L$. If $L < N$ one does almost the same but thinks of DB as a $\sqrt{N \cdot L} \times \sqrt{N \cdot L}$ matrix.

$\mathcal{O}(n\sqrt{d\lambda})$ XOR operations to get the desired shortest path so the client-side storage cost is a constant.

From Table 3.1, we see that this solution incurs a larger asymptotic computation cost on both the server and client side compared to the vanilla APSP solution. Specifically, the server has to perform $\mathcal{O}(n^2 d\lambda)$ XOR operations upon each query from the client in addition to the initial $\mathcal{O}(n^3)$ computations to get the APSPs. These cost increases are not too alarming as, firstly, server-side computation costs are not as important as user-side computation and storage costs. More importantly, the length of the longest shortest path in the Lightning network d is very small and unlikely to increase much as the network grows due to the centralised nature of the network (see Section 3.5.2), and we can reasonably assume $\lambda \ll n$.

The server-side storage requirements of this solution is unfortunately the same as the vanilla APSP solution and is the main drawback of this approach. In addition, the PIR protocol requires the servers to perform XOR computations linear in the size of the entire database in response to every query. Thus it would be highly beneficial if the database could be kept in memory. As outlined above, for Lightning that is already not possible with this approach.

3.4.4 Hub Labelling

In a hub labelling scheme [ADGW11] one preprocesses a graph such that later, one can efficiently find shortest paths in-between any two nodes. For some graphs, in particular street networks, the preprocessed data is *much* smaller than storing all pairwise shortest paths.

Concretely, given $G = (V, E)$ the idea is to precompute APSPs. Then compute for every vertex $v \in V$ a set of hubs $hub(v) \subset V$ such that for all $u, v \in V$, the shortest path from u to v (if G is directed also from v to u) contains a vertex that lies in the intersection of their hub sets $hub(u) \cap hub(v)$. This is the *covering property* of the hub sets.

The preprocessed DB now contains, for every $v \in V$ its hub set $hub(v)$ and for every $u \in hub(v)$ the shortest path from v to u (and u to v).

If $h = \max_{v \in V}\{|hub(v)|\}$ is the size of the largest hub, we can think of the preprocessed data DB as a database with $n = |V|$ entries of length $L = h \cdot \lambda \cdot d$ bits, i.e., for every $v \in V$ we have $hub(v) \leq h$ paths, each of length $\leq d$, and thus the asymptotic server-side storage drops from $\mathcal{O}(n^2 d\lambda)$ to $\mathcal{O}(nhd\lambda)$. For $h \ll n$, which is the case for the Lightning network (see Section 3.6), this allows us to achieve at least an order of magnitude reduction in server-side storage costs.

Compared to the vanilla APSP solution, adding hub-labelling increases the asymptotic server-side computation by an additive factor of $n^2 \max(d, \log n) \log d$. The increase in computation cost is due to the hitting set computation for each radius. Nevertheless we note that the increase is not big as $d \ll n$ in the Lightning network and so the main cost is still the $\mathcal{O}(n^3)$ from the APSP computation. In addition, server-side computation is not as important as server-side storage, and these computations are also not costs incurred with every query but need only to be be done relatively infrequently, for instance every two weeks or so to take into account changes in the underlying network topology.

The main drawback of this solution is that the client needs to download the entire database to maintain the security guarantee. This would incur an asymptotic storage cost of $\mathcal{O}(nhd\lambda)$ which is problematic as client-side machines have less storage capacity compared to server-side machines.

3.5 The LightPIR Approach

Given the above considerations, we are now ready to present our proposed solution, *LightPIR*.

3.5.1 Basic Approach

In a nutshell, our construction of a private route discovery mechanism augments the basic APSP solution with both a PIR protocol (specifically a 2-server RAID-PIR protocol [DHS14] which is similar in flavour to [CKGS98]) and hub labelling. A client who wants to learn the shortest path from u to v will query the PIR servers for the u, v entries, which will be the hub sets of u and v respectively. Then, the client should perform a set intersection of the two hub sets, which will be non-empty by the covering property of these hubs, and determine the $w \in hub(u) \cap hub(v)$ for which the length of the path $u \to w$ plus the length of $w \to v$ is minimized. The path $u \to w \to v$ is the sought shortest path from u to v.

LightPIR reaps the benefit of a lower asymptotic storage cost of $\mathcal{O}(nhd\lambda)$ from hub labelling and a lower asymptotic communication cost of $\mathcal{O}(\sqrt{nhd\lambda})$ from the 2-server PIR protocol. The composition of these two techniques also lowers the per query asymptotic server-side computation due to the PIR protocol from $\mathcal{O}(n^2 d\lambda)$ XOR operations to $\mathcal{O}(nhd\lambda)$. Compared to the vanilla APSP solution, the user now has to perform some computation to determine the desired shortest path. This computation is extremely simple, basically $\mathcal{O}(\sqrt{nhd\lambda})$ XOR operations to privately recover the query result from the servers and a set intersection on two moderately sized sets of at most $h \ll n$ to get the shortest path, and thus can be performed by even the weakest

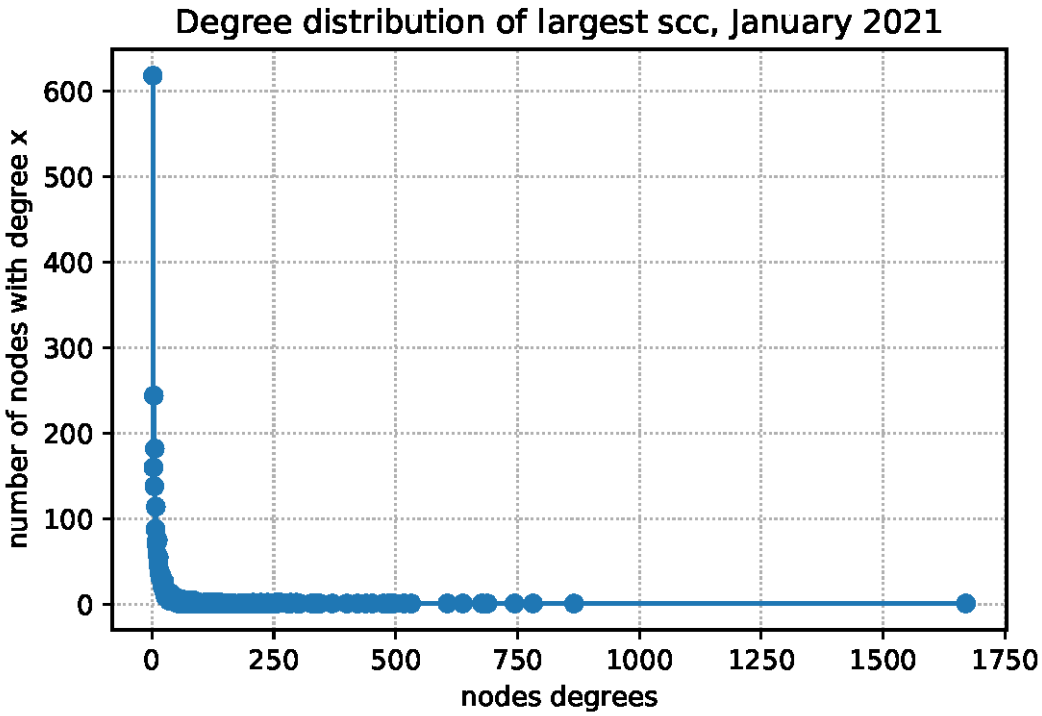


Figure 3.1: A plot showing the number of nodes for each degree in the largest strongly connected component (scc) of the Lightning network on January 2021. The distribution of degrees is quite similar in Lightning network snapshots taken in dates within the past two years.

clients. Finally we note that although the asymptotic server and user side computation costs is higher in our solution compared to the APSP solution with hub labelling, the asymptotic storage incurred by the user drops from $\mathcal{O}(nhd\lambda)$ to $\mathcal{O}(hd\lambda)$ since the user just has to store the hub sets of the desired source and target nodes instead of all the hub sets of all nodes. Again, since $h \ll n$ in the case of the Lightning network, this is at least an order of magnitude drop in client-side storage requirements.

3.5.2 Optimization of Hub Sets

In order to lower the database storage further, *LightPIR* relies on a heuristic which leverages the specific topology of the Lightning network to optimize the size of hub sets. We first begin with a brief description of the Lightning network topology and then describe the heuristic.

Network Topology Characteristics

To understand and exploit the specific topological characteristics of payment channel networks, we downloaded a number of historical snapshots of the Lightning network from the Lightning github repositories [Gita, Dec]. For instance, a representative network snapshot on January 2021 comprises of 6,376 nodes, 39,993 channels, and 3,650 strongly connected components, although except for the largest one, almost all of the rest components have only a very small number of nodes. We thus focus our analysis on the largest connected component of each snapshot. The one of January 2021 has 2,707 nodes and comprises a majority (31,350) of the total number of channels.

As can be seen from Figure 3.1, the majority of the nodes in the largest component of the Lightning network have very low degrees and about 100 nodes have degrees of > 100. The largest component also has a relatively small diameter of 9. We can therefore infer a large degree of centralisation in the Lightning network, with a few central nodes of high degree connected to each other as well as connected to shorter chains of vertices (see Figure 3.2).

A channel (i.e. an edge) between two nodes in the network is weighted by its channel fee. This usually comprises of a base fee which is independent of the amount one wishes to move through the channel, and an amount-specific rate fee which depends on the amount sent across the channel. Table 3.2 includes a number of network characteristics for a number of snapshots of the Lightning network over the past two years.

Hub Set Computation: Greedy Heuristic

One way of computing the hubs for each node in the network is to make use of the highway dimension (HD) algorithm [AFGW10, ADF$^+$11]. Intuitively, HD is a graph metric that captures the size of a subset of nodes that intersects all shortest paths of non-trivial length. In order to compute the HD, we compute sets of nodes that intersect all paths of size $(r, 2r]$ for a small set of radii and measure how sparse these sets are in the network. In the following, we explain how we use these sets to compute the hubs for each network node (Algorithm 3.1).

In [AFGW10], the authors give a definition of HD based on shortest-path covers and also give an approximation algorithm to compute the HD of a network using this definition. In [ADF$^+$11], the authors present a modification that improves their earlier work, which also applies to the shortest-path covers algorithm. We further modify this approximation algorithm with a heuristic from observing the topology of the Lightning network, as shown in Algorithm 3.1, to both return a hub set for each node in the network, and also maintain an upper bound on the HD of the network. That is, we always overapproximate the HD but never underapproximate it.

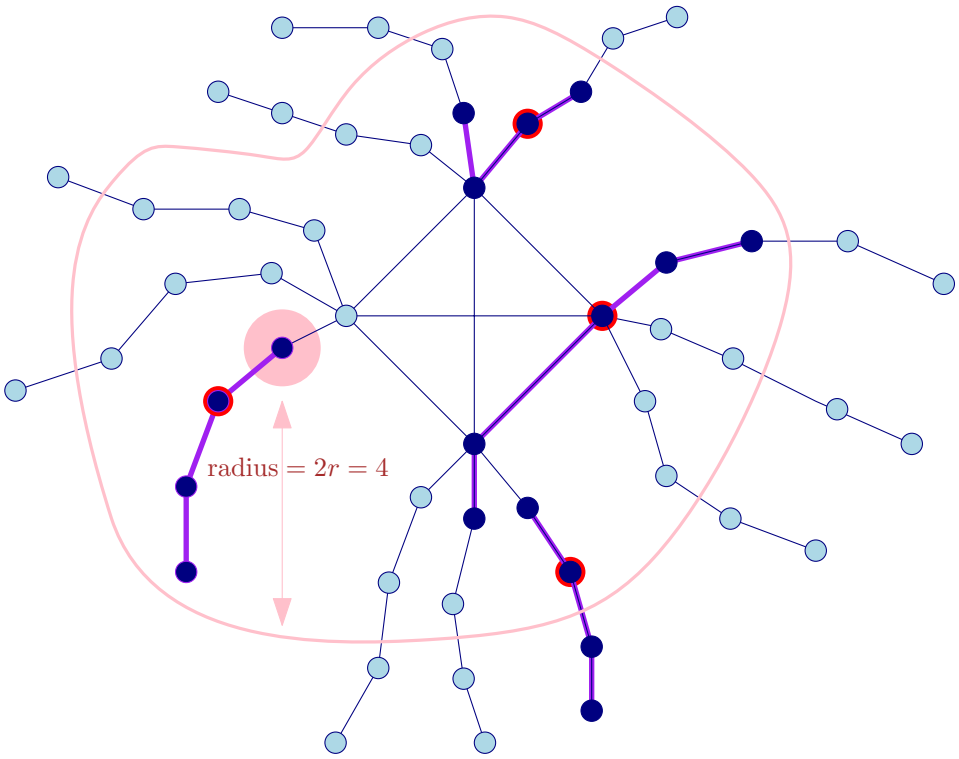


Figure 3.2: A graph created from star graphs where the centers of the graphs form a clique. A few paths of length in $[r+1, 2r]$ for $r = 2$ with a (red) cover point on each path are shown. A ball (neighborhood) of radius $2r = 4$ is shown, and it contains a substantial fraction of the graph, and thus the cover points.

We present a heuristic for computing an upper bound of the highway dimension (HD) in Algorithm 3.1. Our heuristic is a modification of the approximation algorithm derived from the shortest-path cover definition of HD in [AFGW10].

We present a few necessary ingredients from [ADF+11, Section 4], before elaborating on our heuristic. Let $G = (V, E)$ be the network topology. The neighborhood $N_x(u)$ of a node $u \in V$ for a radius x is the set of all nodes that appear in a path P, such that $v \in P$ and $|P| \leq x$, where $|P|$ denotes the length of the path P ($N_x(v)$ contains the same number of nodes as the ball with center u and radius x). Moreover, a set C of nodes is a (k, r)-shortest path cover ((k, r)-SPC) of G if and only if (I) for each shortest path P, such that $r < |P| \leq 2r$, $P \cap C \neq \emptyset$ and (II) $\forall u \in V$, $|C \cap N_{2r}(u)| \leq k$. Then, if G has HD h, then for any $r > 0$ there exists an (h, r)-SPC of G. Thus, the authors claim that by using the greedy $\mathcal{O}(\log n)$ approximation for computing hitting sets, we can combine the above into an algorithm that computes a bound of HD in $\mathcal{O}(h \log n)$, where $h = \mathcal{HD}(G)$ is the highway dimension of G. Specifically, given a hitting set $cover[r]$ of all pairs shortest paths of length in $(r, 2r]$, they make sure that

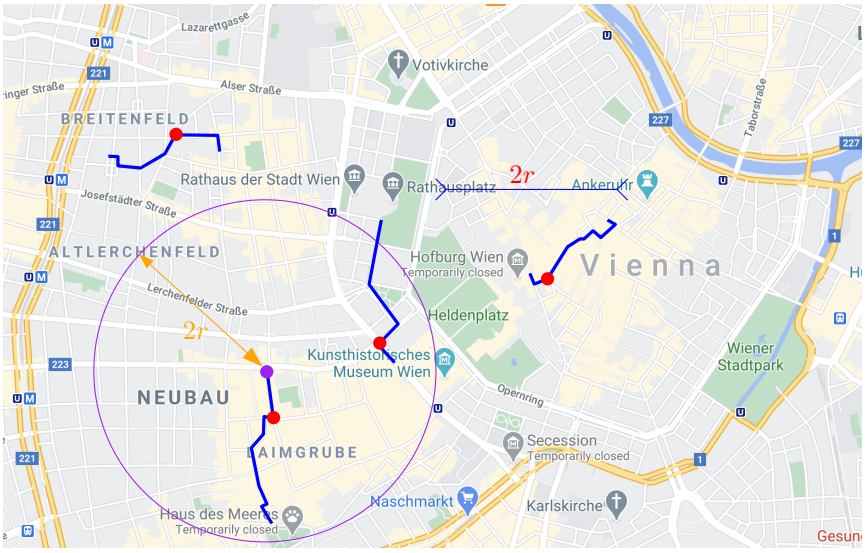


Figure 3.3: A street network with 4 shortest paths of length in $(r, 2r]$ (for $r \approx 500m$) and a cover point of those paths (in red) highlighted. A ball (neighborhood) with radius 2r (one with center at the starting point of a path shown in purple) will only contain cover points resulting from relatively few paths "in the neighbourhood".

condition I holds and then they can fulfill Condition II by taking all $cover[r] \cap N_{2r}(v)$ intersections for all $v \in V$ and setting k to be the maximum size of those intersections. As mentioned in [ADGW11], it suffices to compute the above with the set of radii $r \in \{2^i \mid i \in \mathbb{N} \wedge 2^i \leq diameter(G)\}$ Of course, an exact hitting set algorithm would give a smaller $cover[r]$, but computing it is NP-hard.

Our heuristic is based on the key observation that, due to the centralised nature of the network, high degree nodes in the Lightning network appear in a vast majority of shortest paths, as opposed to the less dense and centralised road networks for which HD was originally designed (Figure 3.3). Thus many neighborhoods for center u and radius $2r$ include these high degree nodes, which work as gateways to many remote nodes. We denote by $base(\ell)$ the set of the ℓ nodes with the highest degree (in short, $base$). We noticed that for network topologies similar to the one of Lightning removing the base nodes after computing the covers can significantly reduce the HD (Figure 3.2). Specifically, we adapt the HD computation by first computing the cover sets as in [AFGW10, ADF$^+$11], but then removing the base nodes and all the edges connected to the base nodes to obtain the topology $\overline{G} = G \setminus base$, in which we compute the per-node hubs as follows. That is the hub set of node u for radius $2r$ is defined by $hubs\_in\_neighb(u, r) = base \cup (N^{\overline{G}}_{2r}(u) \cap cover[r])$, where $N^{\overline{G}}_{2r}(u)$ is the neighborhood $N_{2r}(u)$ computed in $\overline{G}$. Note that [ADF$^+$11] (in addition to [AFGW10]) requires that the path weights are unique integers, but this is easily achieved with

insignificant perturbations of edge weights (in our case we initially set the weights to be the channel base fees and then perturbed them). We illustrate this heuristic in Algorithm 3.1 and show that it correctly computes an upper bound $\overline{\mathcal{HD}}(G)$ of $\mathcal{HD}(G)$ in Lemma 3.5.1.

Lemma 3.5.1. *Algorithm 3.1 computes an upper bound of $\mathcal{HD}(G)$.*

Proof. We prove that for any two communicating nodes s and d, there exists a hub node in the set of hubs that lies on the shortest path between s and d, $SP(s,d)$. Let $|SP(s,d)| = k$ and let r be the radius for which $r < k \leq 2r$ (there has to be at least one such radius as our set of radii covers all shortest paths). If $SP(s,d) \subset N_{2r}^{\overline{G}}(s)$, i.e. if $SP(s,d)$ entirely appears in the neighborhood with center s and radius $2r$ in the resulting topology *after* removing $base$ from G, then the hub for s and d is in $SP(s,d) \cap cover[r]$, as $cover[r]$ hits all shortest paths of length in $(r, 2r]$. Otherwise, if $SP(s,d)$ does not entirely appear in $N_{2r}^{\overline{G}}(s)$, then there exist at least one node in $base$ that intersects $SP(s,d)$. Since $base$ is included in the set of hubs (Algorithm 3.1, line 14), then the proof is complete. □

Note that from the proof of Lemma 3.5.1, we guarantee the covering property and hence correctness of the hub sets: the hub set of any node covers all shortest paths from that node to any other node in the network. This means that any two hub sets in a connected network must certainly have a non-empty intersection.

The time complexity of Algorithm 3.1 is computed as follows. The APSP step takes $\mathcal{O}(n^3)$ and the partitioning of APSP according to their length takes $\mathcal{O}(n^2 \log d)$. The computation of covers, which includes the computation of hitting sets (greedy heuristic costs $\mathcal{O}(n^2 \max(d, \log n))$), takes $\mathcal{O}((n^2 \max(d, \log n)) \log d)$ and the hub computation takes $\mathcal{O}(n \log d)$. Thus, in total $\mathcal{O}(n^3 + n^2 \log d + (n^2 \max(d, \log n)) \log d + n \log d) = \mathcal{O}(n^3 + n^2 \max(d, \log n) \log d)$ (cf. Table 3.1). In practice, the diameters of the Lightning network snapshots that we used to test Algorithm 3.1 were quite small, thus the running time was dominated by the APSP computation (which was a matter of minutes).

3.6 Empirical Evaluation

To provide a first empirical evaluation of our approach, we implemented *LightPIR* and ran experiments on historical snapshots of the Lightning network from the Lightning network gossip repository [Dec]. In general, we find that our approach indeed achieves an improvement by an order of magnitude in terms of storage requirements and a roughly 40% reduction in the highway dimension of each snapshot, which directly translates to a reduced hubs database.

Algorithm 3.1: *LightPIR*'s hub set optimization

Input: $G = (V, E)$, the network topology and ℓ, the number of high degree nodes to use as base

Output: $\overline{\mathcal{HD}}(G)$ (bound on $\mathcal{HD}(G)$) and $hubs(u)$, $\forall u \in V$

$APSP \leftarrow allPairsShortestPaths(G)$;
$radii \leftarrow \{2^i \mid i \in \mathbb{N} \wedge 2^i \leq diameter(G)\}$;
$base \leftarrow highDegreeNodes[0 : \ell - 1]$;
/* compute shortest path sets */
for $r \in radii$ **do** $shortestPaths(r) \leftarrow \emptyset$; /* init */
for $path \in APSP$ **do**
  for $r \in radii$ **do**
    if $r < |path| \leq 2r \wedge base \cap path = \emptyset$ **then**
      $shortestPaths(r).add(path)$;
/* compute hitting sets and covers */
for $r \in radii$ **do**
  $cover[r] \leftarrow base \cup hittingSet(shortestPaths(r))$;
/* compute $N_{2r}^{\overline{G}}(u) \cap cover[r]$ sizes & hubs */
$\overline{G} \leftarrow G \setminus base$;
for $u \in V$ **do** $hubs(u) \leftarrow \emptyset$; /* init $hubs$ */
for $(u, r) : u \in V \wedge r \in radii$ **do**
  $hubs\_in\_neighb(u, r) \leftarrow base \cup (N_{2r}^{\overline{G}}(u) \cap cover[r])$;
  $hubs(u) \leftarrow hubs(u) \cup (hubs\_in\_neighb(u, r))$;
$\overline{\mathcal{HD}}(G) = \max_{u,r} |hubs\_in\_neighb(u, r)|$;

Setup In our experiments, we used a virtual machine with 24 cores (Intel Xeon CPUs E5-2650 v4 at 2.20GHz) and 16 GB RAM, running Ubuntu 18.04.1 LTS. Our experiments were implemented in Python (version 3.7.6) using networkx [HSS08] among other libraries.

Data The extracted snapshots range from March 2019 to the current state (January 2021) with the smallest snapshot having 3,480 nodes and the largest having 6,376 nodes. We ignore earlier snapshots as they have too few nodes. We performed our empirical evaluation on the largest strongly connected component (SCC) in the snapshot and we found that although the network size almost doubles from March 2019 to January 2021, the number of nodes in the largest SCC remains pretty consistent across all snapshots (≈ 2500 nodes). The first seven columns of Table 3.2 give a complete view of the datasets used.

Table 3.2: Lightning Network Snapshot Characteristics and Evaluation Results (scc stands for strongly connected component(s))

date	nodes	channels	scc	largest scc								
				nodes	channels	diam.	baseline HD bound	heuristic HD bound	base size	max hub set size	hub DB size (MB)	exec. time (h)
13-03-2019	3480	45071	994	2480	42025	7	386	209	100	271	1.5	2.03
13-07-2019	4469	53084	1804	2661	46733	7	312	191	100	248	1.4	1.41
13-09-2019	4733	51508	2027	2706	45774	7	389	228	100	304	2.1	2.64
13-11-2019	4598	44784	2158	2422	38011	6	306	186	110	235	1.2	1.31
13-03-2020	5070	48340	2356	2712	41571	6	368	212	100	270	1.7	2.67
13-05-2020	5557	49569	2570	2985	42093	8	347	196	100	257	1.6	1.98
13-07-2020	5888	51932	2748	3136	44005	7	361	211	100	286	1.8	2.27
13-09-2020	5972	51783	2800	3171	43844	7	373	220	112	285	1.8	2.47
13-11-2020	6074	48361	2978	3089	40695	8	419	253	103	315	2.3	2.96
13-01-2021	6376	39993	3650	2707	31350	9	422	259	106	314	1.8	1.83

Results We computed the optimal base size for the largest SCC in each snapshot by running a binary search to find the size of the base set that minimises the approximate highway dimension of the SCC as computed by our algorithm (ℓ in Algorithm 3.1's input). The baseline algorithm of [AFGW10, ADF+11] is the one where $\ell = 0$, i.e., there the base set is empty. As shown in Figure 3.4, for most Lightning network snapshots a base size of around 100 nodes minimises the approximate highway dimension of the largest SCC. In fact, the reduction in highway dimension is around 40% across all snapshots when compared to the baseline (see Table 3.2 baseline HD bound vs heuristic HD bound).

Across all snapshots the median hub set size as computed by our algorithm is between 150 to 200 nodes, with the maximum hub set size being 315 nodes. The distribution of hub sizes for each snapshot is shown in Figure 3.5. Given that the average size for storing each snapshot is about 13MB, our method provides an overall reduction of an order of magnitude in terms of the hub set sizes (cf. second last column of Table 3.2). Due to the inclusion of the base, the minimum hub set size is at least 100. Specifically, as shown in Table 3.2, the average database size across all snapshots is 1.65MB.

The running time for each snapshot is about two hours (last column of Table 3.2). This includes the search for the base size that minimizes the HD (and thus the hubs database). Thus, it is possible to run our algorithm periodically to update the hub sets as the lightning network topology changes.

3.7 Future Directions

We would like to highlight three interesting directions extend our work on private and efficient routing on PCNs.

Firstly, when shortest paths are not unique, most available algorithms for selecting a shortest path out of a set of paths of equal length just select a random one from the set.

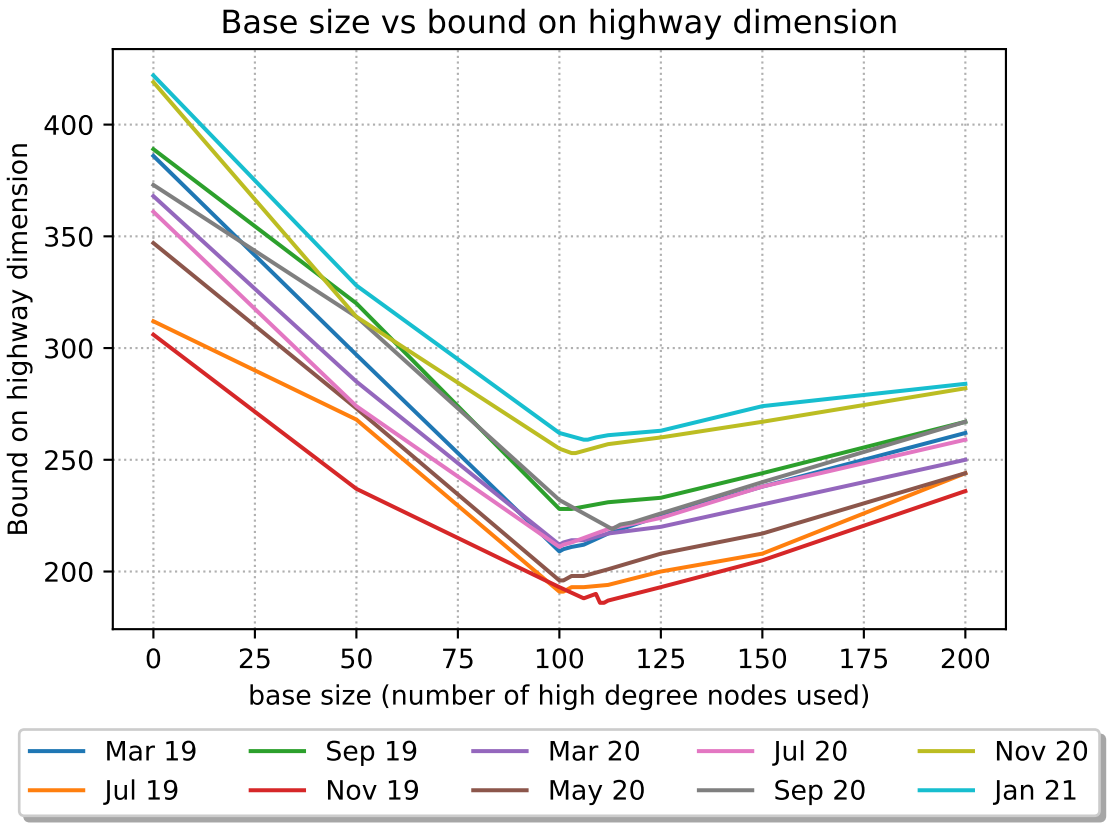


Figure 3.4: The x-axis shows the different base set sizes tested (a base with size ℓ includes the ℓ nodes with the highest degree) and the y-axis shows the corresponding highway dimension bound. The baseline computation corresponds to the values for base size 0, which are the highest. We run our heuristic with the base size that minimizes the highway dimension bound (thus also the hub database) and reported the results in Table 3.2.

In terms of reducing the size of the hub set, one can design smarter algorithms that do not select the path randomly, but optimise for paths with a larger vertex overlap with the set of other paths already selected. This ensures that when we greedily select vertices to be in the hub set, we would select vertices that cover a larger set of paths and so could potentially decrease the size of the hub set.

Secondly, our current heuristics to calculate the highway dimension of the network and to reduce the size of the hub sets are heavily inspired by the topology of the Lightning network. It would be interesting to track the topology of the Lightning network as the network grows in size, to see if there are significant changes and to think about how to incorporate these changes in the network topology to further optimise the hub set sizes.

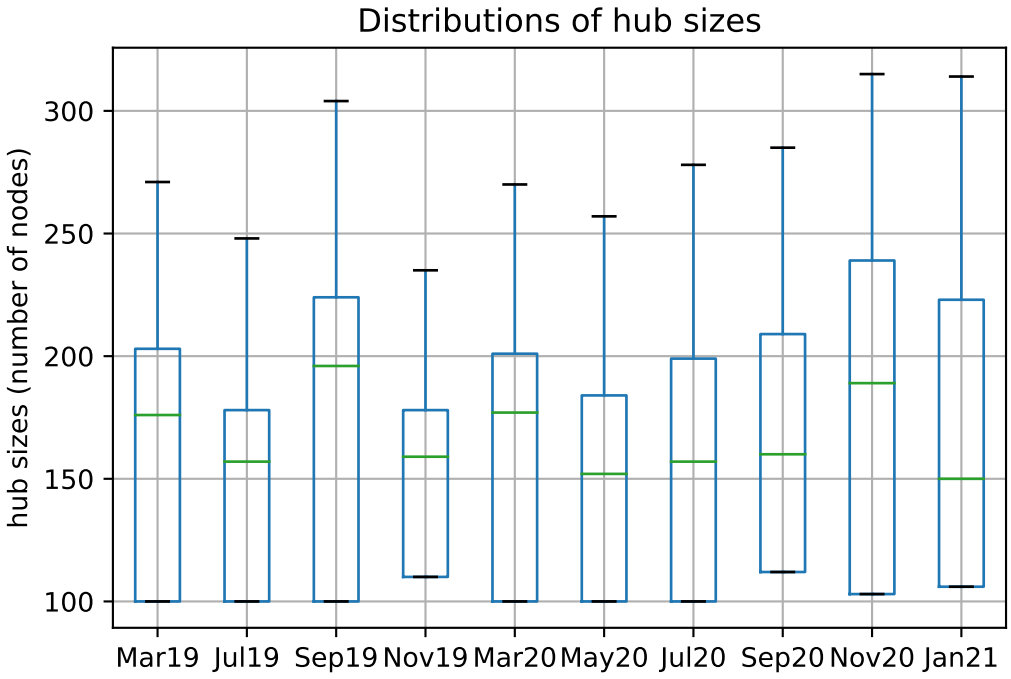


Figure 3.5: Box plots of hub sizes (in number of nodes) for a number of snapshots of the Lightning network. The green line shows the second quartile (median), the bottom and top of each box show the first and third quartiles, respectively, and the extended lines out of each box (whiskers) end at the min (bottom) and max (top) values of the datasets.

Lastly, we note that the network topology and thus the hub sets change with the amount of Bitcoins a user wants to send, e.g., due to the capacity limitation of channels in the network (i.e. the amount of Bitcoins each channel can forward). For instance the shortest path from Ahad to Ben might go through Charlotte when Ahad wants to send 1 Btc but the path might have to go through Djordje when Ahad wants to send 10 Btc because Charlotte cannot forward more than 1 Btc. Additionally, the topology could also change due to the way the fees are calculated. The fees on channels comprise of a base fee and a rate fee which depends on the amount sent over the channel and thus the shortest path information could change drastically between sending small or large amounts of Bitcoin. Since we do not yet take into account fees, our current work can be seen as optimised for sending a fixed amount of Bitcoin and thus an interesting future direction would be to take into account the amount of Bitcoins a user wants to send and design a similarly efficient and private method of extracting shortest paths.

CHAPTER 4

Efficient and private rebalancing on PCNs

A kettle had beans inside,
And stalks of the beans made a fire;
When the beans to the brother-stalks cried,
"We sprang from one root, why such fire?"

Cao Zhi

This section is based on the following publication:

- Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, Michelle Yeo. [†] *Hide & Seek: Privacy-Preserving Rebalancing on Payment Channel Networks*. In Financial Cryptography and Data Security - 26th International Conference, **FC 2022**

4.1 Introduction

Depleted payment channels. As users transact more frequently in a PCN, the likelihood that their balance in their channels gets depleted consequently increases.

[†]Authors ordered alphabetically.

Unlike a classic bank account where users can always top up their account with more funds if necessary, a major drawback of payment channels is that users cannot simply top up their balance in the channel off-chain once it is depleted. Instead, they have to go on-chain to close and then re-open a new payment channel with more funds injected. As transacting on the blockchain is expensive (in the form of added transaction fees and delays), it is imperative for nodes to avoid on-chain solutions as much as possible.

Rebalancing as a solution to channel depletion. A solution to extend the lifetime of payment channels that circumvents the blockchain altogether is *rebalancing*. Rebalancing updates payment channels off-chain with the crucial condition that the overall balance of each node is unchanged (see Figure 2.3 in Chapter 2 for an example of rebalancing among 3 nodes in a PCN). Although it is not possible to shift funds from one payment channel to another off-chain, the effect of rebalancing is precisely that: funds from well-funded payment channels transfer to depleted ones.

Current rebalancing approaches and drawbacks. There are two predominant approaches to rebalancing. The first involves a local search of rebalancing cycles (i.e., transactions of a fixed amount that begin and end with the same user) initiated by a single user. This is the current rebalancing approach in the Lightning Network [Gitb]. The second approach (introduced in [KG17a] and termed "Revive") is global instead of local: nodes looking to rebalance specify a maximum amount of rebalancing flow along each of their channels, where the rebalancing transactions are determined by a global evaluation of the state of the network.

A drawback of single-user based cycle finding is it overlooks other rebalancing requests across the network, leading to *local solutions*. Figure 4.1 illustrates one such consequence which we call the "cancelling out" effect. Suppose a user Charlotte wants to move 10 coins from her channel with Ben to his channel with Ahad. If Charlotte utilises the cycle finding approach, she will only manage to rebalance 6 coins as depicted in the graph on the right. The channel between Ben and Ahad would be ignored because of the lack of sufficient balance on Ben's end. In a globally optimal solution, however, the entire rebalancing in the graph on the left can be executed, as it takes into account that transactions in both directions can be above the capacity of a channel, as long as they "cancel out" and the resulting transaction is within the capacity.

Furthermore, users must check if the other users on the cycle are willing to forward the rebalancing transaction amount, even after finding rebalancing cycles. This could lead to a prolonged and laborious search for cycles with willing participants. Lastly, this approach requires users to have global knowledge of the network topology which can be unrealistic in terms of storage as the size of the network increases.

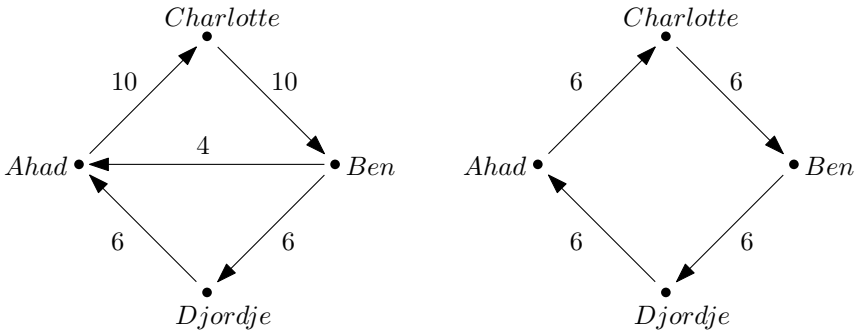


Figure 4.1: The cancelling out effect. The weighted, directed edges on the graph on the left specifies the maximum coins a user can forward along the direction of the edge. The graph on the right shows the maximum rebalancing cycle Charlotte can achieve using the cycle finding approach.

The second approach does not suffer from local limitations such as the canceling out effect, and theoretically achieves the global optimal rebalancing. Revive [KG17a] implemented this method by assigning a random delegate, either a trusted external third party or someone from the set of participants, to receive channel constraints and solve a linear program that models rebalancing. This is a serious *privacy loophole*, since the delegate now has information on the concerned payment channels. Moreover, the delegate has control over the rebalancing output; for instance, the delegate may compute the rebalancing transactions in a malicious or suboptimal way, favouring some transactions over others. Although the authors proposed a method for any participant to challenge the rebalancing transactions, the process is lengthy and requires giving the challenger access to the balances of all participants.

Contribution. In the light of these drawbacks of current rebalancing approaches on PCNs, we present Hide & Seek, the first opt-in rebalancing protocol that is both *private* and achieves a *globally optimal rebalancing*. Each party that is interested in rebalancing specifies the maximum amount to be forwarded in each of the party's channel. We employ selected delegates that receive the maximum amounts per channel, calculate and share with each party the exact amount to be moved in each channel. We formulate our problem as a linear program and set our objective function to maximize the total amount of funds to be rebalanced in the network. Our protocol does not involve transaction fees.

On the other hand, we leverage multi-party computation (MPC) to obtain a *fully private solution*. Specifically, the participants in Hide & Seek only learn the information they would have learned if a trusted third party computed the optimal rebalancing and returned to each participant the amount to be moved along each of their channel. No sensitive information such as the channel balances is leaked.

	Private	Globally optimal	Opt-in	Network locality
Cycle finding solution	✓	✗	✗	✗
Revive	✗	✓	✓	✓
Our solution	✓	✓	✓	✓

Table 4.1: Summary of main approaches for rebalancing. Private solutions are solutions that do not leak balance information. Globally optimal refers to the optimality of the rebalancing solution. Opt-in refers to solutions where willing users choose to participate in the protocol and non-willing users are not involved at all. Network locality refers to solutions that only require local knowledge of the PCN.

Finally, we guarantee the rebalancing can be securely and efficiently executed. We propose a simple way to decompose the optimal rebalancing circulation into a set of transaction cycles. As a result, the transactions of each cycle are easy to execute atomically using HTLCs. We note that atomicity is limited to each rebalancing cycle, therefore increasing the protocol's robustness; any cycle can be executed successfully regardless of the success of other cycles.

We highlight the advantages of our approach in Table 4.1.

4.2 Related Work

Rebalancing PCNs. There are several payment channel primitives proposed in literature [Spi, PD15, DW15, AKWZ21a, ALW20, MBB+19b]. Regardless of the primitive, a challenge all PCNs share is how to route transactions in the PCN while maintaining balanced channels for as long as possible. Classic routing studies in PCNs like SilentWhispers [MMSKM17], SpeedyMurmurs [RMSKG17], and others [PZS+16b] ignore that channels may be slowly depleting. A promising approach to avoid channel depletion and prolong the network availability for transaction routing is to maintain balanced channels or occasionally perform rebalancing. But transaction routing is a challenging task on its own because the channel balances remain secret for privacy purposes [KG17a, SVR+20a, YXK+18], let alone avoiding channel depletion on-top.

Khalil and Gervais introduce the first channel rebalancing protocol, called Revive [KG17a]. They formulate the problem as an LP, similarly to our work. Then, a delegate is elected to solve the LP and return the solution to the rebalancing participants. Although our work lies close to Revive, it also differs in several aspects. First, Revive considers rebalancing as an LP as well, but Hide & Seek employs faster and more specific min-cost flow algorithms. Second, Revive relies on a single delegate to compute the optimal rebalancing which leaks private information about balances to the delegate. In contrast, Hide & Seek uses MPC to achieve full privacy guarantees. Finally, atomic execution of the rebalancing transactions in Revive requires the transaction language of

the underlying blockchain to be Turing-complete, and thus it is not suitable for Bitcoin. Hide & Seek avoids this issue by first decomposing the optimal rebalancing into cycles, and then executing these cycles atomically using HTLCs. The cycle decomposition in Hide & Seek also ensures that, as long as the channel is not part of all cycles, some rebalancing can still occur if individual HTLCs fail on a channel.

From a practical perspective, rebalancing in the Lightning Network currently utilises a brute force search for rebalancing cycles with sufficient capacity. An automated approach for doing so using the imbalance measure was proposed by [PN20a]. Unlike Hide & Seek, these methods do not leverage other rebalancing requests to find the globally optimal rebalancing. These methods also require nodes to have global knowledge of the network whereas nodes in Hide & Seek only need to have local knowledge of the PCN.

Recently some works introduce routing protocols that attempt to maintain balanced channels. In particular, Spider [SVR+20a] is a payment routing algorithm that maximizes the throughput while maintaining the original channel balances, without providing rebalancing however. Li et al. [LMZ20a] propose to extend the lifetime of payments channels by estimating payment demand, and using this estimate to decide on the initial balance of channels. Engelshoven and Roos [vER20], on the other hand, leverage routing fees to incentivize the balanced use of payment channels. All these works are orthogonal and complementary to ours, as we introduce an opt-in rebalancing protocol.

Network flows and MPC. The general problem of solving network flow problems via MPC is considered in [Aly15]. Various privacy preserving implementations of combinatorial optimization problems are presented. The author acknowledges that the cost for privacy is very high even for the simplest of problems. Roughly speaking, their MPC implementations must iterate for the theoretical worst-case number of iterations to maintain privacy. For the practical problem of rebalancing though, we do not choose to implement extra iterations. On the other hand, we believe that suboptimal rebalancing is better than no rebalancing, and recommend terminating the min-cost flow solution prematurely if needed. Both scaling algorithms and network simplex algorithms monotonically generate better solutions in each iteration, leaving the participants with a feasible solution if they stop early.

4.3 Preliminaries

In this section, we summarise some relevant background information and standard algorithms regarding network flows which will be useful for the rest of this chapter.

Consider a directed graph $G = (V, E)$ and the associated $|E|$-dimensional Euclidean space of non-negative flow along each edge. A circulation is a flow $\mathbf{f} = (f(u, v))_{(u,v)\in E}$

such that the net flow through each vertex is zero: $\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u), \forall u \in V$. Two circulations $\mathbf{f}_1, \mathbf{f}_2$ can be added to get yet another circulation: $\mathbf{f}_1 + \mathbf{f}_2 = (f_1(u,v) + f_2(u,v))_{(u,v) \in E}$. A cycle is a sequence of vertices $v_1, v_2 \dots v_k$ such that $(v_i, v_{i+1}) \in E, \forall 1 \leq i \leq k-1$ and $(v_k, v_1) \in E$ as well. We may equivalently refer to this cycle as $(e_1, e_2 \dots e_k)$ where $e_i = (v_i, v_{i+1}), \forall 1 \leq i \leq k-1$ and $e_k = (v_k, v_1)$. We call k the length of this cycle. A cycle flow $\mathbf{f}$ of weight w on cycle C is a circulation where $f(e) = w, \forall e \in C$ and $f(e) = 0$ otherwise.

A standard result of network flow theory is that any circulation may be expressed as a sum of at most $|E|$ cycles. We refer the reader to the textbook of Ahuja, Magnanti and Orlin([AMO93]) for a detailed treatment.

4.4 Protocol Overview and Model

4.4.1 System model

Payment network topology. We model the PCN as a graph $\tilde{G} = (\tilde{V}, \tilde{E})$, with a vertex for each node and an edge between u and v if there is a payment channel between them. Let $V \subset \tilde{V}$ be the users in the PCN that are interested in rebalancing and let $G = (V, E)$ be the subgraph of $\tilde{G}$ induced by V. We denote $|V| = n, |E| = m$. We assume each user u has only local knowledge of the PCN topology, i.e., only knows the capacities and balances on the edges incident to u.

Cryptographic assumptions. We assume the existence of secure communications channels, hash functions and signatures. We follow [DN03] and assume the concept of an arithmetic black box for MPC $\mathcal{F}_{ABB}$, in particular with functionalities like secret sharing, storage, retrieval, addition, multiplication, and comparisons.

Blockchain & network model. We assume a synchronous network, i.e., there is known bounded message delay. We further assume the underlying blockchain satisfies persistence and liveness as defined in [GKL15].

4.4.2 Protocol overview

In a nutshell, our proposed protocol Hide & Seek consists of two phases: an exploration phase and an execution phase. Firstly, the goal of the exploration phase is to discover rebalancing cycles privately and efficiently. Then, the goal of the execution phase is to guarantee that the rebalancing transactions are executed in a secure manner. At the same time we want to maximise the efficacy of our protocol, that is, we want as many rebalancing cycles to go through as possible.

Exploration phase. The exploration phase first formulates the rebalancing problem as a linear program. Then we randomly select k delegates out of the participants to perform an MPC protocol to jointly solve the linear program. Next, any set of participants that wish to participate in the rebalancing protocol prepares the shared inputs to the delegates. The output of the exploration phase is a rebalancing circulation.

Execution phase. We first efficiently decompose the rebalancing circulation output of the exploration phase into a set of cycle flows. These cycle flows have the property that they are *sign-consistent*, i.e. they are consistent with the direction of the flows in the rebalancing circulation. This makes executing these cycles incentive-compatible, as no user would have to execute transactions which violate their specified rebalancing capacity and direction along channels. Once this is done, we enforce atomicity of these cycles by creating an HTLC for each cycle which ensures either transactions along the entire cycle goes through or none at all.

4.4.3 Desired properties & threat model

In general, we assume a computationally bounded adversary, i.e., runs in probabilistic polynomial time. The properties HIDE & SEEK should guarantee are the following:

1. **Balance conservation (security):** The total balance of each node, which is the sum of the node's balances on each incident channel, must remain the same before and after HIDE & SEEK, even when *all other participants are corrupted* by the adversary.

2. **Privacy:** The information revealed during HIDE & SEEK should be not exceed the minimum required to execute rebalancing: (a) the participants must only learn the transaction amounts for each of their payment channels; (b) the delegates of MPC should not be able to determine private financial information of the participants. Both (a) and (b) should hold as long as *one of the delegates is not corrupted*.

3. **Optimality (completeness):** Assuming *every participant is honest*, the result should be optimal in that no other rebalancing yields greater total change over all payment channels.

4.5 The Hide & Seek Protocol

4.5.1 Exploration phase

Linear programming for rebalancing

The practical problem of rebalancing has many facets, including keeping participants' financial information private and facilitating coordination. We overlook these considerations momentarily to present the underlying optimization problem of rebalancing.

For a payment channel between (u, v), the users would like to move the state balance(u, v) towards a desired state balance$^*(u, v)$. If balance$^*(u, v) >$ balance(u, v) then rebalancing would involve u transferring funds to v, and we model this as a directed edge from u to v with capacity $m(u, v) :=$ balance$^*(u, v) -$ balance(u, v). If balance$^*(u, v) <$ balance(u, v) then there is a directed edge from v to u with capacity $m(v, u) :=$ balance$(u, v) -$ balance$^*(u, v)$.

The capacities $m(u, v)$ represent the most flow that can occur through each channel during rebalancing. If $m(u, v) = 0$ the edge from u to v is either non-existent or equivalently, a zero-capacity edge. We also enforce that if $m(u, v) > 0$, then necessarily $m(v, u) = 0$.

Let us denote a potential rebalancing by $\mathbf{f} \in \mathbb{R}^{|E|}$ on this directed graph, where $f(u, v)$ denote the flow from u to v. Since rebalancing should not result in a net financial gain or loss for any participant, we require $\mathbf{f}$ to be a circulation. Recall that it means the net flow through each vertex is zero:

$$\sum_{v:(u,v)\in E} f(u, v) = \sum_{v:(v,u)\in E} f(v, u).$$

Not only must the flows be non-negative, but they must also satisfy the capacity constraints as specified by participants:

$$0 \leq f(u, v) \leq m(u, v).$$

Thus, the set of valid rebalancings is a polytope in m-dimensional Euclidean space defined by $n + 2m$ linear constraints: n zero flow constraints for each vertex and m pairs of flow capacity constraints for each edge.

We wish to compute a rebalancing that maximizes the linear objective $\sum_{(u,v)\in E} f(u, v)$. We call the linear program so specified the rebalancing problem. This choice of objective function amounts to maximizing the total change in each payment channel's balance towards its desired state.

Solving the rebalancing problem

One can apply any linear programming algorithm of preference to solve the rebalancing, such as any from the family of simplex methods. In fact, the rebalancing problem can be reduced to the min-cost flow problem, a specialization of linear programming which can be solved more easily. For instance, the min-cost flow problem admits a strongly polynomial algorithm, meanwhile the corresponding question for linear programming is a major open problem in the field. [APS+22a] illustrates how the rebalancing problem is equivalent to a min-cost flow problem with the same number of vertices and edges. Henceforth, we refer to the rebalancing problem as a min-cost flow problem.

Delegate selection and multi-party computation

Delegate selection can be done using a simple version of cryptographic sortion as in [GHM+17]. Each of the k delegates involved in the MPC gets $n + 2m$ inputs which are shares of each of the n participant's zero flow constraints and the $2m$ rebalancing capacity constraint along the m directed edges (for each edge we have two constraints: one which specifies the maximum rebalancing flow in one direction, and another which specifies the flow has to be 0 in the other direction). The objective function is also shared and given as an input to the delegates. The delegates jointly compute the optimal solution to the rebalancing LP problem and each delegate outputs a share of the final flow on each edge at the end of the protocol.

4.5.2 Execution phase

Cycle decomposition

The exploration phase concludes with a solution to the rebalancing linear program obtained through multi-party computation. This solution $\mathbf{f}^*$ is in fact encoded as shared secrets given to individual participants. Instead of directly sending each $f^*(u, v)$ to u, we decompose the circulation into a sum of cycle flows. This makes the execution of rebalancing via HTLCs easier; instead of the entire network committing their funds to a large atomic rebalancing transaction, each cycle only requires coordination between nodes constituting the cycle.

As mentioned earlier, each circulation can be expressed as a sum of cycle flows. We briefly describe a standard algorithm to compute this decomposition efficiently. Algorithm 4.1 uses depth-first search as a subroutine to detect cycles and then induce cycle flows on them. Figure 4.2 depicts a circulation and its decomposition into cycle flows.

Algorithm 4.1: Depth-first Search Cycle Decomposition

input : Circulation $\mathbf{f}$ on directed graph $G = (V, E)$
output : A set of cycle flows $\mathcal{S}$ that sum to $\mathbf{f}$

initialize $i = 1$
initialize $R \longleftarrow \{e \in E : f(e) \neq 0\}$ set of active edges
while $R \neq \emptyset$ **do**
  pick an edge $e_1 \in R$
  run depth first search to find a cycle $C_i = (e_1, e_2, \dots e_k)$ in R
  $w_i \longleftarrow \min f(e), e \in C_i$
  initialize $\mathbf{f}_i \longleftarrow 0$
  for $e \in C_i$ **do**
    $f_i(e) = w_i$
    $f(e) \longleftarrow f(e) - f_i(e)$
    if $f(e) = 0$ **then**
      delete e from R
  $i \longleftarrow i + 1$
return $\mathcal{S} = \{\mathbf{f}_1, \mathbf{f}_2 \dots \mathbf{f}_i\}$

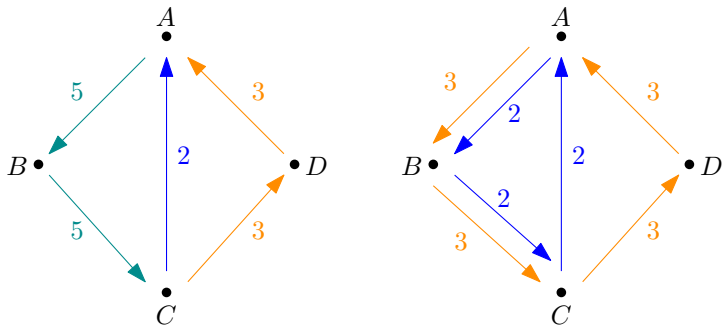


Figure 4.2: The graph on the left depicts a circulation. The weight of each edge is the transaction amount to send along the edge. The cycles in the graph on the right is a sign consistent decomposition of the the circulation.

HTLC commitments per cycle

Given such a decomposition, we need to enforce atomicity of each cycle flow by creating an HTLC for each cycle c in the set. This can be done by first selecting a user in each cycle at random to initiate the cycle. This user has to choose a random secret r_c from some domain $\mathcal{X}$ and create a hash of the secret $h_c = H(r_c)$. The timelock for the initiator of the cycle and the next user is set equal to the length of the cycle. The transaction amount to send along each cycle is the weight of the cycle w_c. Every

Algorithm 4.2: HTLC creation for cycles

input : $\mathcal{S}$ set of directed cycles

1 **for** $c \in \mathcal{S}$ **do**
2 select starting user u_c at random from users in c
3 timelock $t_c \longleftarrow len(c)$
4 u_c chooses random secret r_c and creates hash $h_c = H(r_c)$
5 **for** $e_c = (u, v) \in c$ *starting from* u_c **do**
6 u creates $HTLC(u, v, w_c, h_c, t_c)$
7 decrement t_c by 1

subsequent user in the cycle decrements the timelock value by 1 and looks up the next user in the cycle they should create an HTLC with (determined by the vertex order in the cycle). They then create an HTLC with that user with the decremented timelock value (lines 5-7 in Algorithm 4.2).

Finally we note that Algorithm 4.1 and Algorithm 4.2 can be computed privately using MPC. To prevent any two users on a cycle c from sharing their hash h_c with each other and thus finding out they are in the same cycle, one can use MAPPCN [TM20] to preserve user anonymity.

4.6 Analysis

4.6.1 Desired Properties

An execution of Hide & Seek satisfies the desired properties as stated in Section 4.4.3. Let us study each of the properties in order.

Balance Conservation. Suppose there is a node v that enjoys net financial gain through the execution of Hide & Seek under a malicious adversary. Hide & Seek specifies a set of cycle flows that the nodes may execute, and v must have participated in some subset of these. Note that by atomicity of cycle flows ensured through HTLCs, it is not possible for a cycle to be executed partially (even when parties act maliciously). If v's balance increased, that means there must be at least one cycle flow with net positive flow through v. But this contradicts the definition of cycle flows, since they must satisfy zero flow through each node:

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u) \quad \forall v \in V.$$

Privacy. The sensitive data used in the exploratory phase of Hide & Seek remains private as long as at least one delegate of the MPC is honest (inherited by the MPC). In the execution phase, users do not know the other users in their cycle except their predecessors and successors in their cycles as we use MAPPCN to preserve user anonymity.

Optimality. Assuming the delegates compute the solution correctly, the circulation returned by the min-cost flow algorithm maximizes the total flow through each edge. Under the same assumption, the cycle decomposition algorithm would result in an equivalent (and thus also optimal) set of cycle flows.

4.6.2 Efficiency

We break the analysis of the efficiency of Hide & Seek into three parts: (1) solving the rebalancing problem, (2) cycle decomposition, and (3) MPC.

Solving the rebalancing problem. Solving the underlying min-cost flow problem is the most computationally intensive aspect of Hide & Seek. Fortunately we can leverage the vast body of algorithms for this problem being asymptotically optimal in different parameter regimes. The complexity of these algorithms is analyzed in terms of n, m, the largest capacity U and the largest cost W of an edge. We may presently ignore the term W as each edge has identical cost 1. For the parameter regime of rebalancing, we recommend the double scaling algorithm of Ahuja, Goldberg, Orlin and Tarjan with computes the optimal solution in time $O(nm \log nW \log \log U)$ ([AGOT92]).

An alternative is to use a network simplex algorithm. This family of algorithms are excellent in practice, although theoretical analysis of their effectiveness is an active area of research in optimization. Simplex algorithms are also incredibly simple, and for this reason they have been recommended in Toft's framework for privately solving linear programs([Tof09] despite the somewhat poorer theoretical guarantees. We recommend the network simplex algorithm of Orlin ([Orl96]) for the rebalancing problem, which terminates in at most $O(nm \log n)$ pivots. Generally, the amortized cost per pivot is $O(n)$, but Orlin presents a modification with total runtime $O(nm \log n \log nW)$.

Cycle decomposition. If the rebalancing circulation obtained by solving the min-cost flow contains n' vertices and m' edges, then the cycle decomposition algorithm as detailed in Algorithm 4.1 terminates in $O(n'm')$ time, which is $O(nm)$ at worst. Every loop iteration removes at least one edge, and each iteration visits at most n' vertices before finding a cycle. The pre-processing of G to obtain the subgraph induced by the circulation takes $O(n + m)$ time.

Also note that the timelocks used in the execution of a cycle flow are bounded by the length of the cycle.

MPC. Although MPC implementations of optimization algorithms incur a penalty in speed, there are multiple methods to speed up the implementation of HIDE & SEEK:

Firstly, the rebalancing problem, much like many other min-cost flow problems, satisfies the Hoffman-Gale conditions: the optimal solution, along with the vertices of the polytope, is guaranteed to be integral. This means the MPC can be performed over faster integer arithmetic rather than slower floating point arithmetic.

HIDE & SEEK can be implemented even faster by reducing the number of bits per variable. This quantity is governed by the maximum capacity per edge as well as the granularity of rebalancing, so that the number of bits required depends on the specific cryptocurrency. For instance, an implementation of HIDE & SEEK for Bitcoin with just 20 bits per variable may restrict all quantities to multiples of $2^{10} = 1024$ satoshis up to 2^{30} satoshis which is approximately 10 bitcoins.

The number of delegates chosen to compute the MPC also contributes to the communication cost during rounds, and here we note that HIDE & SEEK does not place any limitations on this number. In fact, it can be as low as two delegates as long as one of them is honest.

Finally, the efficiency of our protocol inherently depends on the MPC primitives used. This is a wide and active area of research, with a lot of new developments in making efficient MPC primitives [CFIK03, Aly15, CdH10, BOSS20].

4.7 Limitations and Extensions

In this section, we identify the limitations of our protocol and discuss possible extensions.

4.7.1 1 HTLC per channel

The first limitation of the cycle decomposition method in the execution phase of our protocol is that the funds on any edge which is a part of multiple (say k) cycles is split into k smaller transaction amounts. This could incur a higher cost in terms of larger number of UTXOs on the underlying blockchain (larger bloat) if the channel is closed. In what follows, we propose an alternative method which creates only 1 HTLC for each channel and thus overcomes this limitation.

We can decompose G into a set of cycles $\mathcal{S}$ using any standard cycle decomposition algorithm (for instance using Algorithm 4.1 to decompose G into cycles).

For each cycle $c \in \mathcal{S}$, we sample a random secret preimage r_c and compute a hash of this secret $h_c = H(r_c)$. This hash value would be given to all the edges in cycle c as input to their HTLC. For an edge e which is part of multiple cycles, say $c_1, ..., c_k$, instead of creating k HTLCs as in Algorithm 4.2, we can create 1 HTLC locked with the k hash values $h_{c_1}, ..., h_{c_k}$ with a total transaction amount being the weight of $e \in G$ and with a single timelock t (we discuss how to choose this later). The sender and the receiver of the HTLC at an edge e is determined by the direction of $e \in G$. That is, if $e = (u, v) \in G$ then the sender in the HTLC is u and the receiver is v and vice versa. Figure 4.3 top row highlights the difference between these two methods.

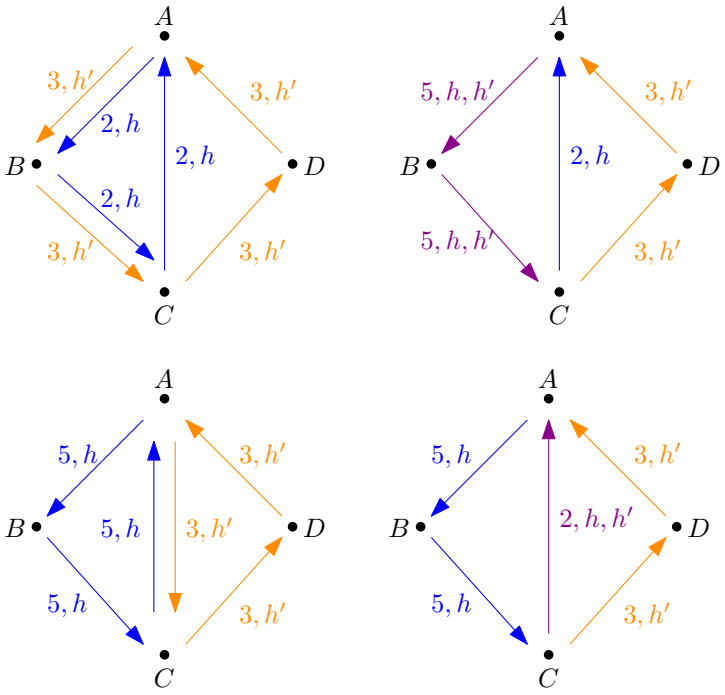


Figure 4.3: 1 vs multiple HTLCs per edge

Let c_h denote the cycle $c \in \mathcal{S}$ that uses the hash h for all edges in the cycle. We define the owner of h to be the first user in the directed edge that e corresponds to in the cycle c_h. In the graph on the top right of Figure 4.3 for instance, the owner of both h and h' on edge (A, B) is A. In the graph in the bottom right of Figure 4.3, the owner of h on edge (C, A) is C but the owner of h' on the same edge (C, A) is A.

Algorithm 4.3 describes how to create an HTLC for a transaction amount x on a directed edge $e = (u, v)$ between a sender u and a receiver v and a given timelock t but locked with multiple hash values. The main idea is to include with each hash value h an additional piece of information I which is an indicator function indicating which user (either u or v) "owns" the hash. The user who owns the hash would require the

Algorithm 4.3: HTLC with multiple hashes

```
input : users u, v, transaction amount x, set of hashes tuples H, timelock t
initialise R ⟵ ∅
while current time < t do
    for (h, I) ∈ H do
        if I(h) = v then
            check if u can produce preimage r | H(r) = h and if so add r to R
             check if v can produce preimage r | H(r) = h and if so add r to R
        else
            check if v can produce preimage r | H(r) = h and if so add r to R
    if |R| = |H| then
        allow v access to locked funds x
refund locked funds x to u
```

other user to produce the preimage r such that $H(r) = h$ within the time limit of t. This check is done for all hash values. If all the preimages are received within the time limit t, then x will be unlocked for the receiver v. Otherwise, the sender u can demand a refund of x.

Let G be the circulation output of the exploration phase. An important constraint for assigning timelocks on E is that, for any directed path $p \in G$, the timelocks of the edges in p have to be either strictly monotone decreasing, or there is only one point in the path where strict monotone decreasing timelocks does not hold (where the cycle closes). We say an assignment of timelocks on E is *valid* if it fulfils this constraint. For instance, in Figure 4.4, the graph on the left is the circulation output of the exploration phase and the graph on the right is a valid assignment of timelocks to this circulation. The timelocks on all the edges of all directed paths in the graph (for instance the cycle (C, F, A, B, C)) are strictly monotonic decreasing except at F. An interesting question is the maximum timelock in a valid assignment (as this specifies the maximum time some user would have to wait for the funds locked in their channel to be unlocked) and Lemma 4.7.1 shows that this is at most $|E|$.

Lemma 4.7.1. *There exists a valid assignment of timelocks to the edges $e \in E$ such that the maximum timelock is at most $|E|$.*

Proof. Let $G = (V, E)$ be a directed graph. First we show, if G is acyclic, that any topological ordering of G induces a valid timelock assignment to the edges of the graph. Second, for the case of a general directed graph G, we show that we can "cut" cycles in G by selecting at most one vertex on each cycle and obtain an acyclic graph G' from

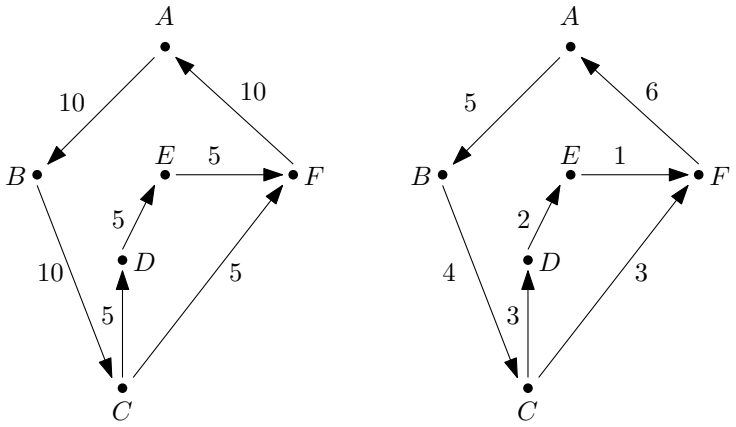


Figure 4.4: The graph on the left is a rebalancing circulation where each edge (u, v) is labelled with the total transaction amount u has to send to v. The graph on the right is a valid assignment of timelock values to the circulation on the left. The values on the edges depict one assignment of timelock values to execute the entire rebalancing atomically in our alternative proposal.

G. We then show that any topological ordering of G' gives rise to a valid timelock assignment on G.

Suppose G is acyclic. Consider the reverse graph G^r which has the same vertices as G but with the direction of every edge reversed. Since G is acyclic, G^r is also acyclic. Let $\mathcal{T} = (v_1, ..., v_{|V|})$ be a topological ordering of the vertices of G^r. Then define a timelock assignment on G by assigning to any edge $(v, u) \in G$ the index/position of u in the topological ordering $\mathcal{T}$ of G^r. Then a simple induction on the length of a path in G shows that, for any path in G, the vertex assignments along the path defined by this timelock assignment are monotonically decreasing.

Suppose now that G is not acyclic. We need to show that we can cut every cycle in G at exactly 1 location to make G acyclic. Let $\tilde{V}$ be a set of vertices such that every simple cycle in G has exactly 1 vertex in $\tilde{V}$. Note that such $\tilde{V}$ always exists. Indeed, let $\{c_1, ..., c_s\}$ be the set of all simple cycles in G. We can then define $\tilde{V}$ by choosing exactly 1 vertex v_i from each cycle starting from c_1 to c_s. The existence of this tuple $\tilde{V}$ can be proved through induction: for the case of one cycle, it is clear that one can select a vertex arbitrarily. Now suppose the claim is true for k cycles. Consider a set of $k + 1$ cycles. There must exist a vertex v_i which lies in exactly 1 cycle C_i, so we can assign v_i to C_i and remove C_i from the set, so we are left with a set of k cycles. Then consider the subgraph with v_i removed, and apply induction to obtain the other vertices.

We construct another graph G' as follows. For each vertex v in $\tilde{V}$, replace v with k vertices $v^1, ..., v^k$ where k is the number of edges $e^1, \ldots, e^k$ incident to v. Then, for

each $1 \leq i \leq k$, connect the vertex v^i to the edge e^i while keeping the direction of e^i as in G. By construction, $G' = (V', E')$ is an acyclic graph as every simple cycle in G is cut by a vertex $v \in \tilde{V}$. Observe that G' has potentially more vertices than G but has the *same* number of edges.

Since G' is acyclic, there is a topological ordering and hence a valid timelock assignment for E'. This topological ordering has maximum timelock value of $\min(|V'|, |E|) \leq |E|$. This assignment of timelocks can also be used for G as the edges are the same. Hence, the claim of the lemma follows. □

As a remark, we can use integer programming to construct the set $\tilde{V}$ by assigning each vertex a value of 0 or 1 and imposing the constraint that the sum of vertices on each cycle is exactly 1.

While this approach creates only 1 HTLC per edge, it also has two drawbacks compared to the cycle decomposition method. First, splitting transactions into different HTLCs could potentially increase the total amount of rebalancing at the end of the protocol. With 1 HTLC per edge, if any user fails to forward the secret preimage along their edge within the time limit, the entire rebalancing circulation would fail. Every user is thus a single point of failure. With multiple HTLCs per edge, as in the cycle decomposition method, the only way to prevent the entire rebalancing circulation from occurring is if one user on each cycle (or a user participating in all cycles) does not forward their preimage. Any other user (not on all cycles) can only fail the rebalancing cycles they are part of. There is thus a much higher probability that some rebalancing occurs in the cycle decomposition method compared to this method.

Second, this method is less efficient compared to the cycle decomposition method. In particular, in this method the funds in the circulation have to be locked for up to $|E|$ time-steps as opposed to l time-steps in the cycle decomposition method, where l is the length of the largest cycle in the decomposition $\mathcal{S}$. If $|E| >> l$, this presents a significant increase in opportunity cost for the participants in the protocol with regards to their locked funds.

Due to these drawbacks, we recommend using the cycle decomposition method in the execution phase. Nevertheless, we present this alternative method as we believe overcoming these limitations is an interesting direction for future work.

4.7.2 Optimality with corrupted participants

Participants' sensitive financial data, such as existence of a payment channel and its capacity for rebalancing, is not verified in the protocol, nor does our threat model consider falsification of this data with respect to optimality.

Unfortunately, this lack of verification can prevent any rebalancing to occur: an adversary with knowledge of the payment channel network can falsify edge data so that each cycle passes through one of their edges. The adversary can then refuse to participate in the execution phase and prevent others from rebalancing, even when cycle flows between honest parties exist.

To defend against such adversary, we propose that parties submit a zero knowledge proof of validity along with their edge constraint data. Although one cannot force participants to participate in rebalancing cycles, this modification certainly increases the success rate of rebalancing cycles in Hide & Seek even under an active adversary.

CHAPTER 5

Algorithms for optimising decisions for existing users in PCNs

> You once told me that the human eye is god's loneliest creation. How so much of the world passes through the pupil and still it holds nothing. The eye, alone in its socket, doesn't even know there's another one, just like it, an inch away, just as hungry, as empty.
>
> Ocean Vuong

This section is based on the following publications:

- Stefan Schmid, Jakub Svoboda, Michelle Yeo.[†] *Weighted Packet Selection for Rechargeable Links in Cryptocurrency Networks: Complexity and Approximation*. In Structural Information and Communication Complexity - 30th International Colloquium, **SIROCCO 2023** (best paper award)
- Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali-Maddah Ali, Stefan Schmid, Jakub Svoboda, Michelle Yeo.[†] *R2: Online Admission Control and Rebalancing in Payment Channel Networks.*. In Financial Cryptography and Data Security - 27th International Conference, **FC 2023**

[†]Authors ordered alphabetically.

5.1 Introduction

Decision making for an existing user in a PCN. An existing user in a PCN faces a myriad of decisions. Some decisions concern transaction forwarding, e.g., whether to forward a transaction or not so as to profit from the transaction fee. Some decisions involve maintaining the lifetime of their payment channels, whether by rebalancing or closing and reopening channels with more funds. These decisions are typically also highly correlated – a user that forwards transactions indiscriminately would find themselves in a position where their balance in their channels rapidly deplete and thus have to find a solution to their depleted channels. These decisions become even more consequential for users who join the network specifically to play the role of an intermediary node that routes transactions, creating channels and fees optimally and selecting the most profitable transactions to maximise their profit from transaction fees [AHWW20, ERE20b].

Offline setting. In this chapter, we analyse and present algorithms to optimise this decision making problem of an existing user in a PCN, over a *single* channel. We begin our analysis of this problem in the offline setting. The input to the problem in this setting is a sequence of transactions that stream along the given channel in both directions. We also define a set of available actions that can be played as a response to each element in the transaction sequence, as well as their associated costs. The goal is design an algorithm that outputs a sequence of actions as a response to the input transaction sequence, such that the total costs incurred by the algorithm is minimised.

Online setting. In addition to analysing the problem in the offline setting, we also look at the problem in the online setting, where the algorithm now has to process the transaction input sequentially and provide the actions to play after each sequential input. The study of the problem in the online setting is important if we want our analysis to be robust (i.e., does not depend on any knowledge or assumptions on the transaction demand in the PCN). Accordingly, we assume that transactions can arrive in an arbitrary order at a channel, and the aim is to design online algorithms which provide worst-case guarantees. We are in the realm of competitive analysis, and assume that an adversary with knowledge of our algorithms chooses the most pessimal online transaction sequence. Our objective is to optimise the *competitive ratio* [BEY05]: we compare the performance of our online algorithms (to which the transaction sequence is revealed over time) with the optimal offline algorithm that has access to the entire transaction sequence in advance.

Contribution. We first introduce and formalise the transaction selection for a channel problem in the offline setting, and show that the problem is NP-hard. Then, we provide

a constant approximation algorithm for the transaction selection for a channel problem which achieves an approximation ratio of $(1+\varepsilon)(1+\sqrt{3})$. In the online setting, we also start by formally defining our main problem and problem setting, as well as two more restrictive sub problems. The algorithms and analysis designed to address these sub problems are eventually used as building blocks for our main algorithm and main theorem for our main problem setting. Our main algorithm achieves a competitive ratio of $7+2\lceil\log C\rceil$ where $C+1$ is the length of the rebalancing cycle used to replenish the funds on the channel off-chain. We also show a connection between our analysis of the problem in the offline setting as well as the online setting by using the approximation algorithm in the offline setting as an oracle for our main online algorithm in the online setting. Finally, we complement our theoretical worst-case analysis by performing an empirical evaluation of the performance of our algorithm on randomly generated transaction sequences. We show empirically that our algorithm performs much better on average compared to our theoretical worst-case bound.

5.2 Related Work

Capacitated communication networks. Our analysis in the offline setting bears some resemblance to problems on optimising flows and throughput in typical capacitated communication networks [CKS04, RT85]. However, there is a crucial difference in our problem setting compared to these other problems. In traditional communication networks, the capacity is usually independent in the two directions of a link in the network [GK00]. In our case, however, the amount of transactions a user u sends to their neighbour v in a payment channel (u,v) directly affects v's capability to send transactions, as each transaction u send to v increases v's balance on (u,v).

Maintaining balanced payment channels. As channel balances are typically private, classic transaction routing protocols on payment channel networks like Flare [PZS+16a], SilentWhispers [MMKM17] and SpeedyMurmurs [RMKG18] focus mainly on throughput and ignore the issue of balance depletion. Recently, several works shift the focus on maintaining balanced payment channels for as long as possible while ensuring liveness of the network. Revive [KG17b] initiated the study of rebalancing strategies, Spider[SVR+20b] uses multi-path routing to ensure high transaction throughput while maintaining balanced payment channels, the Merchant[vER21a] utilises fee strategies to incentivise the balanced use of payment channels, and [LMZ20b] uses estimated payment demands along channels to plan the amount of funds to inject into a channel during channel creation, to just give a few examples. This chapter focuses on minimising costs incurred in the process of handling transactions across a channel and thus we also indirectly seek to maintain balanced payment channels. Moreover, in contrast to previous works which typically assume some form of offline knowledge of the transaction

flow in the network, we provide online algorithms which comes with provable worst-case guarantees.

Off-chain rebalancing. Off-chain rebalancing has been studied as a cheaper alternative to refunding a channel by closing and reopening it on the blockchain. In the Lightning Network, there are already several off-chain rebalancing plugins for c-lightning* and lnd†. An automated approach to performing off-chain rebalancing using the imbalance measure as a heuristic has been proposed in [PN20b]. Our work similarly studies when to rebalance payment channels, however we make the decision in tandem with other decisions like accepting or rejecting transactions. Recently, [APS+21] and [KG17b] propose a global approach to off-chain rebalancing where demand for rebalancing cycles is aggregated across the entire network and translated to an LP which is subsequently solved to obtain an optimal rebalancing solution. These approaches are orthogonal and complementary to ours as our focus concerns decision making in a single payment channel and not the entire network.

Online algorithms for payment channel networks. Online algorithms for payment channel networks have also been studied in [ABWW19] and [FNS21]. Avarikioti et al. [ABWW19] establish impossibility results against certain classes of adversaries, however they only consider a limited problem setting where their algorithms can only accept or reject transactions (with constant rejection cost). Fazli et al. [FNS21] considers the problem of optimally scheduling on-chain recharging given a sequence of transactions. In contrast to previous work, we consider a more general problem setting in our online analysis where our algorithms can not only accept or reject transactions, but also recharge and rebalance channels off chain. We also extend the cost of rejection to take into account the size of the transaction.

Relationship to classic communication networks. Admission control problems such as online call admission [AAF+97, LS15] are fundamental and have also received much attention in the context of communication networks. However, in classic communication networks the available capacity of a link in one direction is independent of the flows travelling in the other direction, and moreover, link capacities are only consumed by the currently allocated flows. In contrast, the capacities of links in payment-channel networks are permanently reduced by transactions flowing in one direction, but can be topped up by flows travelling in the other direction. The resulting rebalancing opportunity renders the underlying algorithmic problem significantly different.

*https://github.com/lightningd/plugins/tree/master/rebalance
†https://github.com/bitromortac/lndmanage

5.3 Offline setting

We begin by analysing our problem in the offline setting. For clarity, we will term the problem in this offline setting "transaction selection for a channel". We first show in Section 5.3.2 that the problem is generally NP-hard. Then, we proceed to describe a procedure in Section 5.3.3 to approximate the optimal capacity in a channel using binary search and use this approximation to derive a lower bound on the cost of the optimal algorithm. We then describe a linear program in Section 5.3.4 that fractionally accepts transactions (i.e. part of a transaction can be accepted) given a fixed channel capacity M and show that the solution of the linear program given M is a lower bound on the cost of the optimal algorithm given M. These results are used as building blocks for our main algorithm and theorem in Section 5.3.6. Nevertheless, we also present a simpler example of how to use the solution of the linear program that also comes with some guarantees in Section 5.3.5 which may be of independent interest.

5.3.1 Model

Transaction sequence. Let (u, v) be a channel in a PCN. We denote an ordered sequence of transactions by $X_t = (x_1, \ldots, x_t)$. Each transaction $x_i \in X_t$ has a size (i.e., the amount of the transaction) and a direction. We simply use $x_i \in \mathbb{R}^+$ to denote the size of the transaction x_i. We say a transaction x_i goes in the left to right direction (resp. right to left) if it goes from u to v (resp. from v to u). Let $X_{\rightarrow}$ denote the subsequence of X_t that consists of transactions going from left to right and $X_{\leftarrow}$ the subsequence of X_t that consists of transactions going from right to left.

User actions and costs. First, we note that creating a channel incurs an initial cost of the amount the user allocates in the channel. That is, if node u allocates b_u in channel (u, v), the cost of creating the channel (u, v) for u would be b_u. Consider a channel (u, v) in the PCN and a transaction going from u to v along the channel. User u can choose to do the following to the transaction:

- **Accept transaction.** User u can accept to forward the transaction if their balance in (u, v) is at least the size of the transaction. The result of doing so decreases their balance by the transaction size and increases the balance of v by the transaction size. Note that apart from gradually depleting a user's balance, accepting transactions does not incur any cost.
- **Reject transaction.** User u can also reject the transaction. This could happen if u's balance is insufficient, or if accepting the transaction would incur a larger cost in the future. For a transaction of size x, the cost of rejecting it is $f \cdot x + m$ where $f, m \in \mathbb{R}^+$.

We note that u does not need to take any action for transactions going in the opposite direction (i.e., from v to u) as these transactions do not affect u's cost.

Problem definition. We now formally define the transaction selection for a channel problem. The input to our problem is a channel (u, v) and a sequence of transactions X_t streaming along the channel. We adopt the optimisation perspective over the *entire channel*, instead of focusing on individual nodes. That is, we suppose nodes u and v collaborate and act as a coalition regardless of how they decide to initially split the capacity on both ends of the channel. The problem therefore is to compute the initial capacity and distribution (how it should be split on both ends) on the channel, as well as to decide on whether to accept or reject each transaction in X_t such that the overall solution minimises the sum of the rejection cost as well as the cost of the capacity locked in the channel.

Optimal algorithm and costs. Let $x_{\min}$ be the size of the transaction with the smallest size in X_t and $M_{\max}$ be the total capacity in the channel needed to accept all transactions. We first observe that $M_{\max}$ for X_t is easy to compute in time $\mathcal{O}(t)$ and is upper bounded by the sum of the size of all transactions in X_t, i.e., $M_{\max} \leq \sum_{i=1}^{t} x_i$. Similarly, given any sequence of *decisions*, we can compute the minimal cost of the capacity locked in the channel and optimal initial distribution of the capacity by greedy simulation. Let OPT denote the cost of the optimal algorithm and OPT_M the cost of the optimal algorithm using a capacity of M in the channel. Additionally, we use OPT^R to denote the cost of the optimal algorithm for rejecting transactions and OPT^C to denote the corresponding capacity cost (i.e., amount of capacity injected in the channel). Similarly, we use OPT^R_M to denote the cost for rejecting transactions of the optimal algorithm using a capacity of M in the channel (note that $OPT^C_M = M$, $OPT = OPT^C + OPT^R$).

5.3.2 Hardness

Having defined the transaction selection for a channel problem, we show in the next theorem that the problem is NP-hard.

Theorem 5.3.1. *Transaction selection for a channel is NP-hard.*

Proof. We show a reduction from the subset sum problem, which is known to be NP-hard [AB09]. In the subset sum problem, we are given a multiset of integers $\mathcal{I} := \{i_1, i_2, \ldots, i_n\}$ and a target integer S. The goal is to find a subset of $\mathcal{I}$ with a sum of S.

Consider the following question in the transaction selection for a channel problem: "is the cost below a given value?" We will show this question itself is NP-hard.

We set the constants to $m = 0$ and $f = \frac{3}{4}$. We create a transaction sequence $\{i_1, i_2, \ldots, i_n\}$ where the jth transaction in the sequence has size equal to i_j which is also the jth element in $\mathcal{I}$. These transactions all go from left to right. Then we add a transaction of size S going from right to left.

Suppose that there exists $\mathcal{I}' \subseteq \mathcal{I}$, such that $\sum_{j \in \mathcal{I}'} i_j = S$. Then we show that the cost is at most $\frac{1}{4}S + \frac{3}{4}\sum_{j \in \mathcal{I}} i_j$.

The solution reaching that cost is as follows: nodes start with capacity S on the right and accept all transactions from $\mathcal{I}'$ and then accept the last transaction of size S. The cost is then $S + \frac{3}{4}\sum_{j \in \mathcal{I} \setminus \mathcal{I}'} i_j$. Since $\sum_{j \in \mathcal{I}'} i_j = S$, the bound holds.

Now, suppose that there is no subset of $\mathcal{I}$ summing to S. Let $\mathcal{A} \subseteq \mathcal{I}$ be any set with sum A. We follow the same procedure as described above by starting with A capacity on the right and accept all transactions from $\mathcal{A}$. The cost for the transactions going from left to right is $A + \frac{3}{4}\sum_{j \in \mathcal{I} \setminus \mathcal{A}} i_j = \frac{1}{4}A + \frac{3}{4}\sum_{j \in \mathcal{I}} i_j$. Depending on whether $A < S$ or $A > S$, the last transaction of size S can be either rejected or accepted. Thus, we need to add $\min(\max(S - A, 0), \frac{3}{4}S)$ to the overall cost, which represents either the additional capacity cost of "topping up" the initial capacity of A on the right side by $S - A$ such that we can accept the last transaction, or the additional cost of rejecting the last transaction S, whichever is smaller. Since $A \neq S$, we know that

$$\frac{1}{4}A + \frac{3}{4}\sum_{j \in \mathcal{I}} i_j + \min(\max(S - A, 0), \frac{3}{4}S) > \frac{1}{4}S + \frac{3}{4}\sum_{j \in \mathcal{I}} i_j$$

Rearranging, we get $\min(\max(S-A, 0), \frac{3}{4}S) > \frac{1}{4}(S-A)$, which means that transaction selection for a channel is NP-hard. □

5.3.3 Approximating the optimal capacity

Here, we present a lemma that allows us to fix the capacity of the channel to some value $M \in \mathbb{R}^+$ for a small trade-off in the approximation ratio of the algorithm. Recall that $x_{\min}$ is the size of the smallest transaction in X_t and $M_{\max}$ is the capacity needed to accept all transactions in X_t. Observe that if the optimal capacity is not 0, it has to lie in the interval $[x_{\min}, M_{\max}]$. We thus fix some $\varepsilon > 0$ and perform a search for M over all $k \in \mathbb{N}$ such that $x_{\min}(1+\varepsilon)^k \leq M_{\max}$. Let us denote by LB_M any lower bound on OPT^R_M, the optimal rejection cost using at most capacity M in the channel.

Lemma 5.3.2. *For any $\varepsilon > 0$, let $\mathcal{M} = \{x_{\min}(1+\varepsilon)^k | k \in \mathbb{N}$ and $x_{\min}(1+\varepsilon)^k \leq M_{\max}\} \cup \{0\}$. Then, the following inequality holds:*

$$\min_{M \in \mathcal{M}} \left(LB_M + \frac{M}{1+\varepsilon} \right) \leq OPT$$

Proof. We first analyse the case where the optimal algorithm rejects all transactions. In this case, we know that since $\mathcal{M}$ contains 0, $LB_0 \leq OPT_0^R = OPT_0 = OPT$, so the inequality holds.

Now, suppose that the optimal algorithm accepts at least one transaction. This means $OPT^C \geq x_{\min}$. So there exists a $k \in \mathbb{N}$ such that $x_{\min}(1+\varepsilon)^{k-1} \leq OPT^C \leq x_{\min}(1+\varepsilon)^k$. Set $M = x_{\min}(1+\varepsilon)^k$. We need to show that that $LB_M + \frac{M}{1+\varepsilon} \leq OPT = OPT^R + OPT^C$. From the way we choose M, we know that $\frac{M}{1+\varepsilon} \leq OPT^C$.

Now we just need to show $LB_M \leq OPT^R$. Observe that the optimal rejection cost for any channel with larger capacity is always at most the rejection cost for any channel with smaller capacity, as in the worst case the algorithm in the former setting accepts the same set of transactions that the algorithm in the latter setting accepts. Thus, for any $M' \geq M$, $OPT_{M'}^R \leq OPT_M^R$. And since we chose M as an upper bound on OPT^C, it means $OPT^R = OPT_{OPT^C}^R \geq OPT_M^R \geq LB_M$. □

Looking ahead, we describe an algorithm that is a $(1+\sqrt{3})$-approximation of LB_M in Section 5.3.6. Thus, together with Lemma 5.3.2, we can use this algorithm to approximate transaction selection with a ratio of $(1+\varepsilon)(1+\sqrt{3})$ by running the algorithm at most $\frac{1}{\varepsilon}\log\frac{M_{\max}}{x_{\min}}$ times. We note that choosing a smaller value of ε yields a better approximation, but increases the running time of the algorithm.

5.3.4 Linear program formulation

Here, we describe a linear program that computes a lower bound for OPT_M^R. We first observe that due to the capacity constraints, the optimal algorithm with capacity M cannot accept transactions with size larger than M. Hence, for the rest of the analysis, we assume that all transactions in X_t have size less than M.

In the linear program, we allow accepting a fractional amount of a transaction. That is, we create a variable $0 \leq y_i \leq x_i$ for every transaction $x_i \in X_t$ that represents the extent to which the transaction is accepted. For instance, $y_i = \frac{x_i}{2}$ means that half of x_i is accepted. We introduce variables $S_{L,i}$ and $S_{R,i}$ denoting the capacity on the left and right ends of the channel after processing the first i transactions from X_t. We reiterate that due to the fact that the total capacity of a payment channel is invariant for the lifetime of the channel, $S_{L,i} + S_{R,i} = M$, and $0 \leq S_{L,i}, S_{R,i} \leq M$.

We can now formulate the linear program in eq. (5.1):

$$\begin{aligned} \text{minimise} \quad & \sum_i f(x_i - y_i) + m\frac{x_i - y_i}{x_i} \\ \text{subject to} \quad & \forall i : y_i, S_{L,i}, S_{R,i} \geq 0 \\ & \forall i : y_i \leq x_i \\ & \forall i : S_{L,i} + S_{R,i} = M \\ & \forall x_i \in X_{\rightarrow} : S_{L,i} = S_{L,i-1} - y_i \\ & \forall x_i \in X_{\rightarrow} : S_{R,i} = S_{R,i-1} + y_i \\ & \forall x_i \in X_{\leftarrow} : S_{L,i} = S_{L,i-1} + y_i \\ & \forall x_i \in X_{\leftarrow} : S_{R,i} = S_{R,i-1} - y_i \end{aligned} \tag{5.1}$$

Let LP_M be the solution of the linear program with capacity parameter M. The following lemma states that LP_M is a lower bound of the optimal cost of the transaction selection for a channel problem with capacity M.

Lemma 5.3.3. *$LP_M \leq OPT_M$ for all M.*

Proof. OPT_M is an admissible solution to the linear program. If some other (fractional) solution is found, we know that it is at most OPT_M. □

The linear program can be solved in time $\mathcal{O}(n^\omega)$ where n is the number of variables in the linear program and ω the matrix multiplication exponent [CLS21] (currently ω is around 2.37).

5.3.5 Example algorithm

We present an example of how to use the solution of the linear program in Section 5.3.4 for some fixed capacity M to create an algorithm that uses twice as much capacity as the linear program but guarantees that all transactions that are fully accepted by the linear program (i.e. $x_i = y_i$) will also be fully accepted by the algorithm.

Algorithm 5.1 describes the decision making process only for transactions coming from left to right on the channel, i.e. $X_{\rightarrow}$. As the decision process for $X_{\leftarrow}$ is symmetric, we omit it to avoid repetition. The algorithm takes as input the solution to the linear program and the transaction sequence X_t. Recall that $S_{L,i}$ and $S_{R,i}$ for $i \in [t]$ are the capacity distributions from the linear program solution on the left and right end of the channel respectively after processing the ith transaction. The algorithm uses the initial distribution $S_{L,0}$ and $S_{R,0}$, and additionally splits the extra M capacity into 2 "reserve buckets" R_L and R_R of size $\frac{M}{2}$ each on both ends. Thus, the initial capacity of the left

Algorithm 5.1: Algorithm accepting all fully accepted transactions

Input: transaction sequence X_t, capacity M, solution of LP: $S_{L,i}, S_{R,i}, y_i$
Output: decisions to accept or reject

```
1  Initialise R_L = M/2, R_R = M/2
2  for i ∈ [t] do
3      if x_i ∈ X_→ then
4          if R_L ≥ x_i − y_i then
5              Accept
6              R_L = R_L − (x_i − y_i)
7              R_R = R_R + (x_i − y_i)
8              S_{L,i} = S_{L,i} − y_i
9              S_{R,i} = S_{R,i} + y_i
10         else
11             Reject
12             R_L = R_L + y_i
13             R_R = R_R − y_i
14             S_{L,i} = S_{L,i} − y_i
15             S_{R,i} = S_{R,i} + y_i
```

node would be $S_{L,0} + R_L$ and the initial capacity of the right node would be $S_{R,0} + R_R$. Intuitively, one can think of the additional capacity in R_L and R_R as a reserve source of capacity that is used to help Algorithm 5.1 *fully* accept transactions that are *fractionally* accepted in the linear program solution. We stress that Algorithm 5.1 always maintains the invariant that $S_{L,i} + S_{R,i} = M$ and $R_L + R_R = M$ for all i.

When processing each transaction, say transaction i which is wlog in $X_{\rightarrow}$, the algorithm first checks if there is sufficient excess capacity in R_L to accept the remaining fraction of transaction i (Line 4 in Algorithm 5.1). If so, the transaction is accepted using $(x_i - y_i)$ capacity from R_L and y_i capacity from $S_{L,i}$. The capacity of $S_{L,i}$ decreases by y_i and the capacity of $S_{R,i}$ increases by y_i, and the capacity in R_L decreases by $(x_i - y_i)$ while the capacity in R_R increases by the same amount. If there is insufficient capacity in $R_{L,i}$, i.e., $R_{L,i} < x_i - y_i$, the algorithm takes y_i from $R_{R,i}$ and adds it to $S_{R,i+1}$, and takes y_i from $S_{L,i}$ and adds it to $R_{L,i+1}$ (see Lines 12, 13, 14, 15 in Algorithm 5.1). Note that the updates to $S_{L,i}$ and $S_{R,i}$ at each step are exactly as the solution to the linear program (Lines 8, 9 and 14, 15).

Lemma 5.3.4. *Given the solution of the linear program, a sequence of transactions X_t, and channel capacity M, Algorithm 5.1 incurs a channel capacity cost of $2M$ and accepts all transactions fully accepted in the linear program.*

Proof. Wlog, let the ith transaction in the sequence belong to $X_{\rightarrow}$. After processing the ith transaction x_i, we denote R_L (resp. R_R) at that step as $R_{L,i}$ (resp. $R_{R,i}$). We show that the channel, at time i, has capacity at least $S_{L,i}$ on the left and at least $S_{R,i}$ on the right.

When $R_{L,i}$ is large enough to accept transaction x_i, we use y_i capacity from $S_{L,i}$ and $x_i - y_i$ capacity from $R_{L,i}$. The capacity y_i from the accepted transaction goes to $S_{R,i+1}$ and the remainder $(x_i - y_i)$ of the capacity goes to $R_{R,i+1}$.

If the transaction is forced to be rejected, we know that $R_{L,i} < x_i - y_i$. Since $R_{R,i} = M - R_{L,i}$, we know that $R_{R,i} > M - x_i + y_i$, and because all transactions have size smaller than M, $R_{R,i} > y_i$ follows. This means we can take y_i from $R_{R,i}$ and add it to $S_{R,i+1}$ and remove y_i from $S_{L,i}$ (because the capacity disappeared from there) and add it to $R_{L,i+1}$.

If the transaction is fully accepted, then $x_i - y_i = 0$. This means that the condition $R_{L,i} \geq x_i - y_i$ is satisfied and the algorithm accepts it. □

We conclude this example with the following remark:

Remark 5.3.1. *The claim in Lemma 5.3.4 holds for any initial distribution of R_L and R_R so long as $R_L + R_R = M$.*

5.3.6 A constant approximation algorithm

Based on the insights in the previous section, we now present a $(1+\sqrt{3})$-approximation algorithm for the transaction selection for a channel problem with fixed capacity M. We present the formal description of the algorithm in Algorithm 5.2 for transactions $x_i \in X_{\rightarrow}$ and omit the procedure for $x_i \in X_{\leftarrow}$ as the decision process is symmetric.

In a nutshell, Algorithm 5.2 consists of three main ideas: first, it uses M capacity to follow the decisions made by the linear program solution as much as possible, using an additional $\sqrt{3}M$ as reserve capacity to fully accept some transactions that were fractionally accepted by the linear program. Second, Algorithm 5.2 tries to maintain a balance in the distribution of capacity in both ends of the channel. Intuitively, any transaction that is accepted by the linear program can be accepted when the algorithm is in this balanced state. Lastly, whenever the capacity is unbalanced: one side (wlog left side) has too little capacity, the algorithm prioritises accepting transactions that come from right to left as well as rejecting transactions that go from left to right. This brings the capacity at both sides to the balanced state, and our analysis shows that the approximation ratio is maintained below $1 + \sqrt{3}$.

Algorithm 5.2: $(1+\sqrt{3})$-approximation algorithm

Input: transaction sequence X_t, capacity M, solution of LP_M:$S_{L,i}, S_{R,i}, y_i$
Output: decisions to accept or reject

Initialise $R_L, R_R = \frac{\sqrt{3}}{2}M, \frac{\sqrt{3}}{2}M$
for $i \in [t]$ **do**
  if $x_i \in X_{\rightarrow}$ **then**
    if $R_L - (x_i - y_i) \geq \frac{\sqrt{3}-1}{2}M$ **then**
      Accept
      $R_L = R_L - (x_i - y_i)$
      $R_R = R_R + (x_i - y_i)$
    else if x_i *is little-accepted* **then**
      Reject
      $R_L = R_L + y_i$
      $R_R = R_R - y_i$
    else
      $\phi_A, \phi_R, U, R'_L, j \leftarrow \textsc{Divide}(R_L, LP_M, X_t, i)$
      $U_R \leftarrow \{\}$
      if $R'_L < 0$ **then**
        $U_R, R'_L \leftarrow \textsc{RejectBig}(X_t, U, R'_L)$
      Accept all $x_i \in \phi_A \cup (U \setminus U_R)$
      Reject all $x_i \in \phi_R \cup U_R$
      $R_L = R'_L$
      $R_R = \sqrt{3}M - R_L$
      $i = j$

Input and initial capacity distribution. Algorithm 5.2 takes as input X_t and the solution of linear program given a fixed capacity M. Recall that $S_{L,i}$ and $S_{R,i}$ for $i \in [t]$ are the capacity distributions from the linear program solution on the left and right end of the channel respectively after processing the ith transaction. The algorithm uses the initial distribution $S_{L,0}$ and $S_{R,0}$, and additionally creates 2 "reserve capacity buckets" R_L and R_R of size $\frac{\sqrt{3}}{2}M$ each on both ends. Thus, the initial capacity of the left node would be $S_{L,0} + R_L$ and the initial capacity of the right node would be $S_{R,0} + R_R$. Intuitively, one can think of the additional capacity in R_L and R_R as a reserve source of capacity that is used to help Algorithm 5.2 *fully* accept transactions that are *fractionally* accepted in the linear program solution.

Algorithm 5.2 accepts transactions in the following way: for a transaction of size x_i wlog

in $X_{\rightarrow}$, assuming there is sufficient capacity in R_L, the transaction is accepted using $(x_i - y_i)$ capacity from R_L and y_i capacity from $S_{L,i}$. The capacity of $S_{L,i}$ decreases by y_i and the capacity of $S_{R,i}$ increases by y_i, and the capacity in R_L decreases by $(x_i - y_i)$ while the capacity in R_R increases by the same amount. If the algorithm rejects x_i, the algorithm takes y_i from $R_{R,i}$ and adds it to $S_{R,i+1}$, and takes y_i from $S_{L,i}$ and adds it to $R_{L,i+1}$. We stress that in doing so, the algorithm always ensures that the updates to $S_{L,i}$ and $S_{R,i}$ at each step are exactly the same as the solution to the linear program. We also note that Algorithm 5.2 always maintains the invariant that $S_{L,i} + S_{R,i} = M$ and $R_L + R_R = \sqrt{3}M$ for all i.

We distinguish between three phases of Algorithm 5.2. We say the algorithm is in the *balanced phase* if both $R_L \geq \frac{\sqrt{3}-1}{2}M$ and $R_R \geq \frac{\sqrt{3}-1}{2}M$. Note that since Algorithm 5.2 starts with $\frac{\sqrt{3}}{2}M$ in both buckets R_L, R_R, the algorithm starts in the balanced phase. If $R_L < \frac{\sqrt{3}-1}{2}M$, we say the algorithm is in the *left phase*, and if $R_R < \frac{\sqrt{3}-1}{2}M$, we say the algorithm is in the *right phase*. We also distinguish between 2 types of transactions: little-accepted and almost-accepted transactions. We say a transaction is *little-accepted* if $\frac{y_i}{x_i} < \frac{\sqrt{3}}{1+\sqrt{3}}$, and *almost-accepted* if $\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{1+\sqrt{3}}$.

Balanced phase. In the balanced phase, Algorithm 5.2 accepts all transactions that are almost-accepted in the linear program solution. It also accepts little-accepted transactions that allow it to remain in the balanced phase. That is, for a little-accepted transaction x_i wlog in $X_{\rightarrow}$, it first checks if the left reserve R_L is sufficient to forward the transaction, and that doing so keeps the algorithm in the balanced phase (Line 4). If R_L does not have sufficient capacity, the algorithm rejects x_i (Line 8).

We first show in the following lemma that rejecting any little-accepted transaction is safe in the sense that doing so will not push the approximation ratio of the algorithm above $1 + \sqrt{3}$.

Lemma 5.3.5. *All little-accepted transactions can be rejected while keeping the approximation ratio below* $1 + \sqrt{3}$.

Proof. Recall that rejecting a transaction x_i incurs a cost of $fx_i + m$, and that a little-accepted transaction is a transaction such that $\frac{y_i}{x_i} < \frac{\sqrt{3}}{1+\sqrt{3}}$. From Equation 5.1, the cost of a little-accepted transaction x_i for the linear program is $f \cdot (x_i - y_i) + m\frac{x_i - y_i}{x_i} \geq \frac{fx_i}{1+\sqrt{3}} + \frac{m}{1+\sqrt{3}} = \frac{1}{1+\sqrt{3}}(fx_i + m)$. From Lemma 5.3.3, we know that the solution of the linear program for a fixed capacity M is a lower bound on the optimal solution with capacity M, hence rejecting little-accepted transactions will not increase the approximation ratio above $1 + \sqrt{3}$. □

In the next lemma, we show that processing little-accepted transactions does not affect whether the algorithm stays in the balanced phase or not. This simplifies the

decision making process of Algorithm 5.2 as it can just focus on the decision problem for almost-accepted transactions.

Lemma 5.3.6. *Algorithm 5.2 never leaves the balanced phase after processing a little-accepted transaction.*

Proof. Depending on whether it is accepted or rejected, each little-accepted transaction moves at most $\frac{1}{1+\sqrt{3}}M$ from the left side to the right side of the channel, and at most $\frac{\sqrt{3}}{1+\sqrt{3}}M$ from the right side to the left side of the channel.

Because $R_L + R_R = \sqrt{3}M$ and any transaction has size at most M, if $R_L - \frac{1}{1+\sqrt{3}}M < \frac{\sqrt{3}-1}{2}M$, then $R_R - \frac{\sqrt{3}}{1+\sqrt{3}}M \geq \frac{\sqrt{3}-1}{2}M$.

That means that rejecting a little-accepted transaction from $X_{\rightarrow}$ does not create a situation where $R_R < \frac{\sqrt{3}-1}{2}M$. □

Left phase. Since Algorithm 5.2 accepts all almost-accepted transactions in the balanced phase, it would sometimes have to enter the left or right phase. Here we describe the procedure for what happens in the left phase (the right phase is analogous).

Suppose Algorithm 5.2 enters the left phase after processing transaction x_{i-1}. The objective of Algorithm 5.2 in this phase is to accept all almost-accepted transactions among the unprocessed transactions (i.e., transactions $x_i, \ldots, x_t$). If this is not possible, the algorithm rejects some of them such that both of the following conditions hold: first, the approximation ratio remains $1 + \sqrt{3}$, and second, the algorithm returns to a balanced phase.

To do so, Algorithm 5.2 calls a subroutine Divide (Line 13 in Algorithm 5.2) to sort all unprocessed transactions into three sets: ϕ_A, ϕ_R, U. Set ϕ_A contains all transactions from $X_{\leftarrow}$. These will be accepted as they will increase the left capacity reserve R_L and help to bring Algorithm 5.2 back into the balanced phase. Set ϕ_R contains little-accepted transactions from $X_{\rightarrow}$. These will be rejected and from Lemma 5.3.5 we know that doing so does not increase the approximation ratio. Set U contains almost-accepted transactions from $X_{\rightarrow}$. Some of these transactions will be accepted and some rejected in a way that maintains the approximation ratio.

Divide (described in Algorithm 5.3) takes as input the transaction sequence X_t, the solution of the linear program as well as the current capacity in the left reserve R_L. Divide creates the sets ϕ_A, ϕ_R, U incrementally by processing each unprocessed transaction and accepting transactions from $\phi_A \cup U$ and rejecting transactions from ϕ_R until one of the following stopping conditions occurs:

1. $R_L < 0$ which would mean the left capacity reserves are depleted

2. $R_L > \frac{\sqrt{3}-1}{2}M$
3. all transactions are processed

If the first stopping condition is reached (Line 15 in Algorithm 5.2), the procedure RejectBig is called. RejectBig (described in Algorithm 5.4) takes as input the set U and outputs another set $U_R \subset U$. This set U_R is created by greedily selecting the biggest sized transactions in U (Line 3 in Algorithm 5.4) and adding them to U_R. These transactions will be rejected and the left capacity reserves will be accordingly updated after each rejected transaction (Line 6 in Algorithm 5.4). The procedure RejectBig terminates when the left capacity reserves $R_L \geq \frac{\sqrt{3}-1}{2}M$.

Now we show that if Divide terminates on either the second and third stopping condition, Algorithm 5.2 will either be in the balanced phase (second stopping condition) or all transactions will be processed and Algorithm 5.2 terminates (third stopping condition).

Lemma 5.3.7. *If* Divide *returns $R'_L \geq 0$ and j, then all almost-accepted transactions between i and j are accepted by Algorithm 5.2 and either all transactions are processed or $R_{R,j} \geq \frac{\sqrt{3}-1}{2}M$ and $R_{L,j} \geq \frac{\sqrt{3}-1}{2}M$.*

Proof. There are two cases when Divide returns $R'_L \geq 0$: either $R'_L \geq \frac{\sqrt{3}-1}{2}$ or $j = t$.

In both cases, we note that Divide simulated accepting all transactions from ϕ_A and U and rejecting all transactions from ϕ_R, and at no time R'_L went below 0. That means that Algorithm 5.2 just repeats decisions of Divide.

Finally, since $R_L + R_R = \sqrt{3}M$ and all transactions are smaller than M, this means that Algorithm 5.2 after emerging from left-phase cannot plunge to a right-phase right away.

□

As the penultimate step in our analysis, we show in the next lemma that RejectBig does not bring the approximation ratio of Algorithm 5.2 over $1 + \sqrt{3}$. We do this by computing the rejection cost incurred by Algorithm 5.2 on transactions from U_R and showing that it is always lower than $(1 + \sqrt{3})$ times the cost of the linear program on U. As the solution of the linear program is a lower bound on the cost of the optimal algorithm, this shows that Algorithm 5.2 maintains the approximation ratio even when rejecting transactions from U_R.

Lemma 5.3.8. *In Algorithm 5.2, for sets U and U_R the following inequality holds:*

$$(1+\sqrt{3})\sum_{x_i \in U} f \cdot (x_i - y_i) + m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} f x_i + m$$

Algorithm 5.3: Function DIVIDE to create sets ϕ_A, ϕ_R, and U.

Input: transaction sequence X_t, solution of LP :$S_{L,i}, S_{R,i}, y_i$, value R_L, capacity M
Output: sets ϕ_A, ϕ_R, U, resulting R_L

$R_L = R_L - (x_i - y_i)$
$\phi_A, \phi_R, U \leftarrow \{\}, \{\}, \{x_i\}$
$j = i$
while $R_L \geq 0$ *and* $R_L < \frac{\sqrt{3}-1}{2}$ *and* $j < t$ **do**
  $j = j + 1$
  if $x_j \in X_{\rightarrow}$ *and* x_j *is almost-accepted* **then**
    $R_L = R_L - (x_j - y_j)$
    $U \leftarrow U \cup x_j$
  else if $x_j \in X_{\rightarrow}$ *and* x_j *is little-accepted* **then**
    $R_L = R_L + y_j$
    $\phi_R \leftarrow \phi_R \cup x_j$
  else
    $R_L = R_L + (x_j - y_j)$
    $\phi_A \leftarrow \phi_A \cup x_j$
return $\phi_A, \phi_R, U, R_L, j$

Algorithm 5.4: Function REJECTBIG to prune out transactions from U.

Input: transaction sequence X_t, set U, value R'_L
Output: set U_R, value R'_L

$U_R \leftarrow \{\}$
while $R'_L < \frac{\sqrt{3}-1}{2}$ **do**
  $x_k \leftarrow$ biggest transaction from U
  $U \leftarrow U \setminus x_k$
  $U_R \leftarrow U_R \cup \{x_k\}$
  $R'_L = R'_L + x_k$
return U_R, R'_L

Proof. If $R'_L \geq 0$, we know that all almost-accepted transactions are accepted from Lemma 5.3.7. For $R'_L < 0$ we prove that $(1 + \sqrt{3}) \sum_{x_i \in U} (x_i - y_i) \geq \sum_{x_i \in U_R} x_i$, then we argue that the whole theorem holds.

Let $D = R_{L,i-1} - R'_L$ where R'_L is the value returned by DIVIDE in Algorithm 5.2. We know from the fact that that $R_{L,i-1}$ is the amount of reserves on the left *before* the

left phase and $R'_L < 0$, thus $D \geq \frac{\sqrt{3}-1}{2}M$.

By following the changes of R'_L in DIVIDE, we get

$$\sum_{x_i \in U} x_i - y_i = D + \sum_{x_i \in \phi_R} y_i + \sum_{x_i \in \phi_A} x_i - y_i$$

Therefore, we know that $\sum_{x_i \in U} x_i - y_i \geq D$.

Algorithm REJECTBIG removes transactions from U until $\sum_{x_i \in U_R} x_i \geq D$. If the condition is satisfied, we know that REJECTBIG returns U_R, because $R'_L \geq \frac{\sqrt{3}-1}{2}$.

If $|U_R| = 1$, we know that $\sum_{x_i \in U_R} x_i \leq M$, because every $x_i \leq M$. So in that case $\sum_{x_i \in U_R} x_i \leq M \leq (1+\sqrt{3})\frac{\sqrt{3}-1}{2}M \leq (1+\sqrt{3})D$.

If $|U_R| > 1$, we know that rejecting just one transaction is not enough. This means the biggest transaction has size at most D, so $\sum_{x_i \in U_R} x_i \leq 2D \leq (1+\sqrt{3})D$.

Now, we know that Algorithm 5.2 rejects less size than the linear program multiplied by $(1+\sqrt{3})$. This implies that $(1+\sqrt{3})\sum_{x_i \in U} f \cdot (x_i - y_i) \geq \sum_{x_i \in U_R} f x_i$, and leaves us to prove $(1+\sqrt{3})\sum_{x_i \in U} m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} m$.

But we know that the transactions are moved to U_R from the biggest size to smallest size. This means that for every $x_k \in U_R$ and $x_l \in U$, $x_k > x_l$, and so $\frac{x_l - y_l}{x_l} \geq \frac{x_l - y_l}{x_k}$ holds. Let x^* be the size of the smallest transaction in U_R. It suffices to show that $(1+\sqrt{3})\sum_{x_i \in U} \frac{x_i - y_i}{x^*} \geq \sum_{x_i \in U_R} \frac{x_i}{x^*} \geq \sum_{x_i \in U_R} x_i$. However, we know that $\sum_{x_i \in U_R} x_i \leq 2D \leq (1+\sqrt{3})D \leq (1+\sqrt{3})\sum_{x_i \in U} x_i - y_i$ since $\sum_{x_i \in U} x_i - y_i \geq D$. Thus, we conclude that $(1+\sqrt{3})\sum_{x_i \in U} m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} m$. □

We now have all the necessary ingredients to state and prove our main theorem, which is that transaction selection can be approximated with an approximation ratio of $(1+\varepsilon)(1+\sqrt{3})$.

Theorem 5.3.9. *The transaction selection for a channel problem can be approximated with a ratio* $(1+\varepsilon)(1+\sqrt{3})$ *in time* $\mathcal{O}(n^\omega \cdot \frac{1}{\varepsilon} \cdot \log \frac{M_{\max}}{x_{\min}})$*, where* ω *is the exponent of* n *in matrix multiplication.*

Proof. We perform a search for the capacity of the channel according to Lemma 5.3.2 and for every capacity M searched we solve the linear program with parameter M (as stated in Equation (5.1)) and run Algorithm 5.2. The solution is the output of Algorithm 5.2 with the smallest cost.

We know that $x_{\min}(1+\varepsilon)^{\frac{1}{\varepsilon} \cdot \log \frac{M_{\max}}{x_{\min}}} \geq M_{\max}$. That means we need to solve the linear program and run Algorithm 5.2 at most $\frac{1}{\varepsilon} \cdot \log \frac{M_{\max}}{x_{\min}}$ times.

From Lemma 5.3.3 we know that the solution of the linear program with parameter M is a lower bound for OPT_M^R.

From Lemma 5.3.4, we know that Algorithm 5.2 accepts all fully-accepted transactions by the linear program. The algorithm can reject any little-accepted transactions by Lemma 5.3.5. We also know from Lemma 5.3.6 that in the balanced phase Algorithm 5.2 accepts all almost-accepted transactions and never leaves the phase after processing little-accepted transactions. Finally, Lemma 5.3.8 shows that even in a left (or right) phase the approximation ratio of Algorithm 5.2 on almost-accepted transactions is $1+\sqrt{3}$. This means Algorithm 5.2 is $(1+\sqrt{3})$-approximation algorithm for the solution of the linear program. Moreover, the algorithm uses $(1+\sqrt{3})$ times more capacity that the linear program.

Using Lemma 5.3.2, we find that the selected solution is a $(1+\varepsilon)(1+\sqrt{3})$-approximation of the transaction selection for a channel problem. □

5.4 Online setting

In this setting, we present a $(7+2\lceil\log C\rceil)$-competitive online algorithm to admit transaction streams arriving at both sides of a payment channel, and also to recharge and rebalance the channel, in order to maximise the throughput over the channel while accounting for costs. In order to prove our main theorem, we decompose the problem into two simpler sub problems:

1. (Sub problem 1.) The first and most restrictive sub problem considers a transaction stream coming only from one direction across a payment channel, and users do not have the option to reject incoming transactions. We present a 2-competitive algorithm for this problem, which is optimal in the sense that no deterministic online algorithm can achieve a lower competitive ratio.

2. (Sub problem 2.) As a relaxation, our second sub problem allows users to reject transactions although all transactions are still restricted to come from one direction along a payment channel. We show that our algorithm achieves a competitive ratio of $2+\frac{\sqrt{5}-1}{2}$ for this sub problem. We stress that our lower bound of 2 we achieve in sub problem 1 also holds in this sub problem, hence our competitive ratio of $2+\frac{\sqrt{5}-1}{2}$ is close to optimal.

All intermediate and main results are summarised in Table 5.1.

We begin by first describing our transactions model, actions and costs in this setting, and as well as formally define our main and sub problems. Then we proceed to describe

Sub problem	Competitive ratio
Unidirectional stream without rejection	2
Unidirectional stream with rejection	$2 + \frac{\sqrt{5}-1}{2}$
Bidirectional stream	$\mathbf{7 + 2\lceil \log C \rceil}$

Table 5.1: Summary of the theoretical results in the online setting. The first column presents each sub problem we analyse in our paper and the second column shows the competitive ratio achieved by our algorithms for each sub problem

a tracking algorithm in Section 5.4.2 which is used as an algorithmic building block in all our online algorithms in all problem settings. The core idea of this tracking algorithm is to allow our online algorithms to track the funds locked inside the channel of the optimal *offline* algorithm, and to use the funds as cues to recharge the channel with more funds. We then present our online algorithms and analysis for each of the sub and main problem settings in Sections 5.4.3, 5.4.4 and 5.4.5. Finally, we conclude our study in this setting with an empirical evaluation of our main online algorithm in Section 5.4.6 to compare its average case performance on randomly generated transaction sequences to our worst-case analysis.

5.4.1 Model

Payment channels. We model the payment channel network as an undirected graph $G = (V, E)$. A payment channel between users ℓ (left) and r (right) in the network is an edge $(\ell, r) \in E$. We denote the balance of user ℓ (resp. r) in the channel (ℓ, r) by $b(\ell)$ (resp. $b(r)$). The *capacity* of the channel is the total amount of funds locked in the channel. That is, for a channel (ℓ, r), the capacity of (ℓ, r) is $b(\ell) + b(r)$.

Recharging and rebalancing payment channels. When a user in a channel does not have sufficient funds to accept a transaction, the user can either reject the transaction, recharge the channel, or rebalance the channel. Recharging the channel happens on-chain and corresponds to closing the payment channel on the blockchain and opening a new channel with more funds. In contrast, rebalancing the channel happens entirely off-chain (refer to Figure 2.3 in Chapter 2 for an example). Here, users find a cycle of payment channels to shift funds from one of their other channels to refund the depleted channel.

Transactions. We consider a transaction sequence $X_t = (x_1, ..., x_t)$, $x_i \in \mathbb{R}^+$, that arrives at a payment channel online. Each transaction x_i has both a size and a direction along a payment channel. The size of a transaction is simply the amount that is being transferred. The direction of a transaction across a payment channel (ℓ, r) determines

who is the sender and who is the receiver. When we have a sequence of transactions that go in both directions along a payment channel, we use $\vec{x}$ to denote a transaction that goes from left-to-right and $\overleftarrow{x}$ to denote a transaction that goes from right-to-left. We say a user, wlog ℓ, *accepts* a transaction of size x coming from the left to right direction along the channel (ℓ, r) if ℓ agrees to forward x to r. Similarly, we say a user ℓ *rejects* a transaction x coming from the left to right direction along the channel (ℓ, r) when ℓ does not forward the transaction to r. When it is clear which channel and direction we are referring to, we simply say ℓ accepts or rejects x.

Costs. We consider three types of costs in our problem setting:

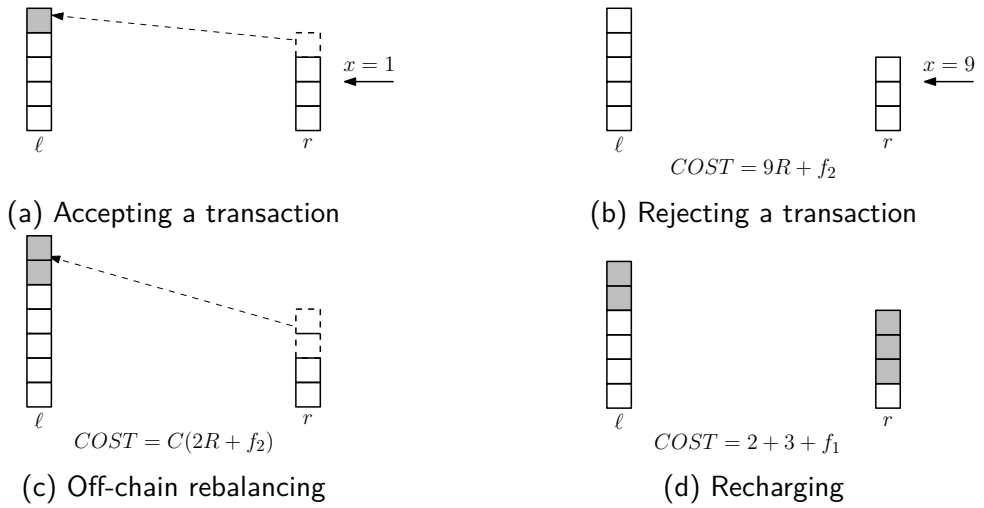
1. **Rejecting transactions:** For a user ℓ, the revenue in terms of transaction fees from forwarding a payment of size x is $Rx + f_2$, where $R, f_2 \in \mathbb{R}^+$. Consequently, the cost of rejecting a transaction of size x is simply the opportunity cost of gaining revenue from accepting the transaction, i.e. $Rx + f_2$.
2. **On-chain recharging:** For any user ℓ, the cost of recharging a channel on-chain is $F + f_1$, where F is the amount of funds ℓ puts into the new channel (this approximates the opportunity cost of locking in the funds in the channel) and $f_1 \in \mathbb{R}^+$ is an auxiliary cost independent of F which captures the on-chain recharging transaction fee.
3. **Off-chain rebalancing:** For any user ℓ, the cost of off-chain rebalancing for an amount x is $C \cdot (Rx + f_2)$, where C is the length of the cycle along which funds are sent -1. In the example of off-chain rebalancing in Figure 2.3, the length of the rebalancing cycle is 3 and thus $C = 2$.

Let us denote by OFF the optimal offline algorithm and ON an online deterministic algorithm. We denote by $\textsc{Cost}_{\mathrm{ON}}(X_t)$ (resp. $\textsc{Cost}_{\mathrm{OFF}}(X_t)$) the total cost of ON (resp. OFF) given the transaction sequence X_t.

Competitive ratio. We say an online algorithm ON is *c-competitive* if for every transaction sequence X_t generated by the adversary,

$$\textsc{Cost}_{\mathrm{ON}}(X_t) \leq c \cdot \textsc{Cost}_{\mathrm{OFF}}(X_t)$$

Main problem. Our main problem is to design a competitive deterministic online algorithm that determines when to accept/reject transactions and when to recharge or rebalance the channel given a bidirectional stream of transactions across a payment channel. More precisely, we consider a stream of transactions that can arrive from both



(a) Accepting a transaction

(b) Rejecting a transaction

(c) Off-chain rebalancing

(d) Recharging

Figure 5.1: Example of actions users ℓ and r can take in the general bidirectional stream setting. Each square represents 1 coin.

right to left or left to right in a given payment channel (ℓ, r). ℓ (resp. r) can choose to accept or reject transactions coming in the left-to-right (resp. right-to-left) direction in the stream. Either user would incur a cost of $Rx + f_2$ for rejecting a transaction of size x. Both users can also recharge the channel on-chain at any point, incurring a cost of $F_\ell + F_r + f_1$ where F_ℓ and F_r are functions of the funds put into the channel by ℓ and r respectively. Since transactions are streaming in both directions in this model, both users would incur costs in this setting. Thus, we seek to design an algorithm that minimises the cost of the *entire* channel. Refer to Figure 5.1 for examples of the actions that a user can take in our main bidirectional transaction stream setting.

To this end, we give a formal definition of two sub problems of decreasing restrictiveness on user actions. We present these sub problems as the algorithms and analysis used to solve these sub problems are used in developing the algorithm and analysis for our main problem.

Unidirectional stream without rejection. In this model, transactions stream only in one direction along a given payment channel. Here, we assume users cannot reject incoming transactions. Formally, given a channel (ℓ, r) and a transaction stream from wlog left to right, user ℓ only accepts a transaction x if $b(\ell) > x$. Otherwise, ℓ has to recharge the channel on-chain with more funds, incurring a cost of $F + f_1$ where F is some function of the amount of funds ℓ adds to the channel. As we only consider transactions streaming in one direction, only one user would incur costs in this setting (the user that has to decide whether to accept or reject transactions). A real world example that motivates this setting is a company which wants to position itself as a

"routing hub" in a payment channel network, providing a routing service in return for transaction fees. As such, the company would want to accept as many transactions possible to acquire the reputation of a hub that is constantly available.

Unidirectional stream with rejection. In this model, we still restrict the transaction stream from wlog left to right in a payment channel (ℓ, r). However, in addition to accepting transactions and recharging, ℓ can now also reject transactions, incurring a rejection cost of $Rx + f_2$ for a transaction of size x.

5.4.2 Algorithmic Building Blocks

Before we describe and analyse the performance of our algorithms in the various problem settings, we first introduce two algorithmic building blocks that we use extensively in our work. The first building block is an algorithm $\textsc{Funds}$. It takes a sequence of transactions as an input and returns the amount of funds that an optimal algorithm uses on this sequence. The purpose of the algorithm is to track the funds OFF has in their channel assuming that the sequence of transactions ends at this point. For the first two sub problems we show how to compute $\textsc{Funds}$. For the main problem, we propose a dynamic programming approach. The second building block is a general recharging online algorithm that calls $\textsc{Funds}$ as a subroutine and uses the output to decide when and how much to recharge the channel. The intuition behind the recharging online algorithm is to recharge whenever the amount of funds in OFF's channel "catches up" to the amount of funds ON has in their channel.

Building block 1: tracking funds of OFF. For a given transaction sequence $X_t = (x_1, \ldots, x_t)$, let us denote $A(X_i)$ to be the amount of funds OFF would use in the channel if OFF gets the sequence $X_i = (x_1, \ldots, x_i)$ (i.e. the length i prefix of X_t) as input. By appending subsequent transactions $x_{i+1}, \ldots, x_t$ from X_t to X_i, we can view $A(X_i)$ as a partial solution to the online optimisation problem that gets updated with any new transaction. In the unidirectional transaction stream (with or without rejection) setting, $A(X_i)$ refers to the funds a user locks into a payment channel. In the bidirectional transaction stream setting, $A(X_i)$ refers to the total balance of both users in the channel. We assume that given an input sequence X_t, $\textsc{Funds}(X_t)$ performs the necessary computations and returns $A(X_t)$. For our main problem, computing $\textsc{Funds}(X_t)$ is generally NP-hard, but we can approximate it to a constant factor. This is exactly described in Algorithm 5.2 described in Section 5.3.6.

Building block 2: using tracking for recharging. In Algorithm 5.5, we describe an online (γ, δ)-recharging algorithm ON that uses $\textsc{Funds}$ as a subroutine to decide when and how much to recharge the channel. ON is run by one user (wlog ℓ) in a

Algorithm 5.5: (γ, δ)-recharging

```
1 Initialise F_tracker, X ← 0, ∅
2 for transaction x in order of arrival do
3     concatenate x to X
4     F'_tracker ← FUNDS(X)
5     if F'_tracker > F_tracker then
6         F_tracker ← F'_tracker + δ
7         recharge to γF_tracker
```

payment channel (ℓ, r). ON calls $\textsc{Funds}$ after each transaction to check if the new transaction sequence results in a significant increase in the amount of funds OFF has in their channel. Whenever ON notices that OFF's funds have increased above a threshold (Line 5), ON recharges the channel with an amount of $\gamma(A(X_i) + \delta)$ where $A(X_i)$ is the amount of funds OFF has in their channel.

Let us denote $A_t := \max_{i \leq t} A(X_i)$. Now we state two important properties of the (γ, δ)-recharging algorithm.

Lemma 5.4.1. *Algorithm 5.5 with parameters (γ, δ) ensures that ON always has at least γ times the amount of funds OFF has and ensures that ON incurs a cost of at most $\gamma(A_t + \delta) + f_1 \cdot \lceil \frac{A_t}{\delta} \rceil$.*

Proof. The first part of the claim follows from the fact that the moment $A(X_i) > F_{\mathit{tracker}}$ for some i, F_{tracker} gets updated to $A(X_i) + \delta > A(X_i)$ and ON recharges the channel to $\gamma F_{\mathit{tracker}} > \gamma A(X_i)$.

For the second part of the claim, we note that the cost incurred by ON is simply the total amount of funds added to the channel with an additional cost of f_1 each time ON recharges the channel on-chain. The amount of funds locked in the channel for ON is always at most $\gamma(A_t + \delta)$ and the times when ON recharges the channel occurs whenever OFF increases its funds by an amount of at least δ. Thus, the number of rechargings for ON that can occur is at most $\lceil \frac{A_t}{\delta} \rceil$ with a cost of f_1 for each recharging instance. The total cost incurred by ON is therefore $\gamma(A_t + \delta) + f_1 \cdot \lceil \frac{A_t}{\delta} \rceil$. □

Next, we show a simple lower bound in terms of A_t for the cost of OFF given a sequence of transactions X_t.

Lemma 5.4.2. *If $A_t > 0$, then $\textsc{Cost}_{\mathrm{OFF}}(X_t)$ is at least $A_t + f_1$.*

Proof. We first note that the sequence of costs for OFF is monotonically increasing, i.e. $\textsc{Cost}_{\mathrm{OFF}}(X_t) \leq \textsc{Cost}_{\mathrm{OFF}}(X_{t+1})$. This comes from the fact that any action of

Algorithm 5.6: Unidirectional transaction stream without rejection

```
Initialise F_tracker, X ← 0, ∅
Initialise balance b = 0
for transaction x in order of arrival do
    concatenate x to X
    F'_tracker ← FUNDS(X)
    if F'_tracker > F_tracker then
        F_tracker ← F'_tracker + f_1
        recharge to F_tracker
    Accept x
```

OFF at step i of the sequence can only increase its cost (i.e. either rejecting x_{i+1} or recharging the channel and then accepting x_{i+1}), or it does not change the cost at all (i.e. by accepting x_{i+1} without recharging).

Since $A_t > 0$, we know that OFF recharged on-chain at some point to an amount A_t for a recharging cost of $A_t + f_1$. Since the sequence of costs for OFF is monotonically increasing, $\textsc{Cost}_{\mathrm{OFF}}(X_t) \geq A_t + f_1$. □

5.4.3 Unidirectional transaction stream without rejection

In this section we consider the first sub problem where, given a payment channel (ℓ, r), transactions stream along the channel in only one direction (wlog left to right). Moreover, ℓ has to accept an incoming transaction of size x and forward it to r if ℓ's balance $b(\ell) \geq x$. Otherwise, ℓ needs to recharge the channel on-chain (and accept the transaction after).

The optimal offline algorithm OFF follows a simple strategy: since it knows the entire stream of transactions in advance, it makes a single recharging action at the beginning of the transaction sequence X_t of size $\sum_{i=1}^{t} x_i$. The cost incurred by OFF is thus $f_1 + \sum_{i=1}^{t} x_i$.

Now, we present a 2-competitive online algorithm ON for this sub problem. The algorithm is described in Algorithm 5.6. ON uses (γ, δ)-recharging with parameters $\gamma = 1$ and $\delta = f_1$. The algorithm accepts all transactions and the recharging ensures that ON always has enough funds.

Theorem 5.4.3. *The algorithm described above is 2-competitive in the unidirectional transaction stream without rejection.*

Proof. From Algorithm 5.4.1, setting $\gamma = 1$ and $\delta = f_1$ gives ON a cost of at most $A_t + f_1 + f_1 \cdot \lceil \frac{A_t}{f_1} \rceil$. Since $f_1 \cdot \lceil \frac{A_t}{f_1} \rceil \leq f_1 \cdot (\frac{A_t}{f_1} + 1) = A_t + f_1$, the cost of ON is at most $2(A_t + f_1)$. From Algorithm 5.4.2, we know that the cost of OFF is at least $A_t + f_1$. Thus, ON is 2-competitive. □

In addition, we note that ON is optimal in this setting. The next theorem proves that no deterministic algorithm can achieve a strictly smaller competitive ratio compared to ON. In particular, our proof shows that ON cannot lock too much funds into the channel, otherwise ON's cost is too high, but if ON locks too little funds, it needs to recharge often.

Theorem 5.4.4. *There is no deterministic algorithm that is c-competitive for $c < 2$ in the unidirectional transaction stream without rejection sub problem.*

Proof. We prove the theorem by contradiction. For the sake of contradiction, suppose that there exists a c-competitive algorithm ON for $c = 2 - \varepsilon$ for some $\varepsilon > 0$. Consider the following sequence of transactions: $\frac{\varepsilon}{3}, \frac{\varepsilon}{3}, \frac{\varepsilon}{3}, \ldots$. We note that when the sequence of transactions is of length k, $\textsc{Cost}_{\mathrm{OFF}}(X_k) = f_1 + k \cdot \frac{\varepsilon}{3}$, as the optimal solution is to recharge the channel at the start of the sequence to the total sum of the transactions in the sequence.

For ON to remain $(2-\varepsilon)$-competitive after processing the first transaction, ON locked at most $f_1 - \varepsilon f_1 + \frac{\varepsilon}{3}$ in the channel ($\textsc{Cost}_{\mathrm{ON}}(X_1) \leq 2f_1 - \varepsilon f_1 + 2\frac{\varepsilon}{3}$).

We generalize the above idea and show that either ON has always smaller amount than $f_1 - \frac{\varepsilon}{3}$ in the channel or at some point it has at least $f_1 - \frac{\varepsilon}{3}$. In both cases, we derive a contradiction to the $(2 - \varepsilon)$ competitive ratio of ON.

First, suppose that ON always recharges to at most $f_1 - \varepsilon'$ for some $\varepsilon' > 0$. Then after t transactions, the number of rechargings is at least $\lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil$. So $\textsc{Cost}_{\mathrm{ON}}(X_t) \geq t \cdot \frac{\varepsilon}{3} + f_1 \lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil$. Setting $t = \frac{3k(f_1 - \varepsilon')}{\varepsilon}$ for some k gives $\textsc{Cost}_{\mathrm{OFF}}(X_t) = f_1 + k(f_1 - \varepsilon')$ and $\textsc{Cost}_{\mathrm{ON}}(X_t) = k(f_1 - \varepsilon') + kf_1$, but since $\lim_{k\to\infty} \frac{2kf_1 - k\varepsilon'}{(k+1)f_1 - k\varepsilon'} = \frac{2f_1 - \varepsilon'}{f_1 - \varepsilon'}$ for any ε', then the competitive ratio is at least 2.

Now, suppose that t is the first time that after processing a transaction, ON has at least $f_1 - \frac{\varepsilon}{3}$ locked in the channel. At time t, $\textsc{Cost}_{\mathrm{OFF}}(X_t) = f_1 + t\frac{\varepsilon}{3}$. Cost of ON is $f_1 + t\frac{\varepsilon}{3}$ for funds locked in the channel plus any additional recharging cost. But since it is the first time ON recharged by more than f_1, the cost for recharging is $f_1 \lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil \geq f_1 + \frac{t\varepsilon}{3}$ for some other positive ε'. So again, $\textsc{Cost}_{\mathrm{ON}}(X_t) \geq t\frac{\varepsilon}{3} + f_1 + t\frac{\varepsilon}{3} + f_1 = 2(f_1 + t\frac{\varepsilon}{3})$ which is twice of $\textsc{Cost}_{\mathrm{OFF}}(X_t)$.

In both cases the cost of ON is at least twice that of OFF which contradicts the assumption that ON is $(2 - \varepsilon)$-competitive. □

Algorithm 5.7: Unidirectional transaction stream with rejection

```
Initialise F_tracker, X ← 0, ∅
Initialise balance b = 0
for transaction x in order of arrival do
    concatenate x to X
    F'_tracker ← FUNDS(X)
    if F'_tracker > F_tracker then
        F_tracker ← F'_tracker + ((√5−1)/2) f_1
        recharge to F_tracker
    if b ≥ x and x is small then
        Accept x
    else
        Reject x
```

5.4.4 Unidirectional transaction stream with rejection

In this section we consider the second sub problem where transactions are still streaming along a given payment channel (ℓ, r) in one direction (wlog left to right). This time though, a user can choose to reject incoming transactions. We describe an algorithm Algorithm 5.7 with competitive ratio $2 + \frac{\sqrt{5}-1}{2}$. We note that the competitive ratio for this setting is larger than the competitive ratio we achieve in the previous setting as OFF has a wider range of decisions.

Let us call a transaction of size x *big* if $x > Rx + f_2$ and *small* otherwise. We first observe that OFF in this setting always rejects big transactions.

Lemma 5.4.5. OFF *rejects all big transactions in the unidirectional transaction stream with rejection.*

Proof. Accepting a transaction x incurs a cost of x for increasing funds. Rejecting a transaction x incurs a cost of $Rx + f_2$. So any big transaction should be rejected. □

Thus, the strategy of OFF in this setting is to simply reject all big transactions. Moreover, if there are sufficiently many small transactions in the sequence to offset the cost of recharging, OFF makes a single recharging action at the beginning of the sequence of size $\sum_{x \in X_t, x \text{ is small}} x$ for a cost of $f_1 + \sum_{x \in X_t, x \text{ is small}} x$.

The online algorithm performs $(1, \frac{\sqrt{5}-1}{2} f_1)$-recharging and it accepts a transaction x if it has enough funds and x is small. The following theorem states that ON is $(2 + \frac{\sqrt{5}-1}{2})$-competitive in this problem setting.

Theorem 5.4.6. *The algorithm described above is $(2 + \frac{\sqrt{5}-1}{2})$-competitive in the unidirectional transaction stream with rejection sub problem.*

Proof. From Lemma 5.4.5, OFF rejects big transactions. Thus, ON should also reject these transactions.

While $A_t = 0$, both OFF and ON reject all transactions in the sequence and both incur the same cost. The moment $A_t > 0$, we know that OFF recharged with an amount at least A_t to accept all small transactions in the sequence. Thus $\textsc{Cost}_{\mathrm{OFF}}(X_t) \geq A_t + f_1$.

At this time ON would have rejected the small transactions in the sequence for at most a cost of $A_t + f_1$ together with some additional recharging cost. From Lemma 5.4.1, we know that the recharging cost for ON is at most

$$A_t + \frac{\sqrt{5}-1}{2} f_1 + \left\lceil \frac{A_t}{\frac{\sqrt{5}-1}{2} f_1} \right\rceil f_1 \leq A_t + \frac{\sqrt{5}-1}{2} f_1 + \frac{A_t}{\frac{\sqrt{5}-1}{2}} + f_1.$$

Summing up both costs, we get

$$\begin{aligned} \textsc{Cost}_{\mathrm{ON}}(X_t) \leq & A_t + f_1 + A_t + \frac{\sqrt{5}-1}{2} f_1 + \frac{A_t}{\frac{\sqrt{5}-1}{2}} + f_1 \\ = & \left(2 + \frac{\sqrt{5}-1}{2}\right) \textsc{Cost}_{\mathrm{OFF}}(X_t) \end{aligned}$$

□

Before analysing the optimality of ON, we first observe, as a simple corollary of Theorem 5.4.4, that the lower bound of 2 also holds for this sub problem.

Corollary 5.4.6.1. *There is no deterministic algorithm that is c-competitive for $c < 2$ in the unidirectional transaction stream with rejection sub problem.*

We conjecture that no other deterministic algorithm can perform better that ON in this setting and sketch an approach to prove the conjecture below.

Conjecture 5.4.1. *conjecture]conjecture:rejection There is no deterministic algorithm that is c-competitive for $c < 2 + \frac{\sqrt{5}-1}{2}$ in the unidirectional transaction stream with rejection setting.*

Sketch. The best ON algorithm needs to recharge the channel when OFF does. If it recharges the channel later, it incurs some rejection cost that OFF is not incurring,

which worsens the competitive ratio. If it recharges sooner, there exists a sequence that either forces OFF to waste funds or incur a big cost.

After OFF recharges, it can reconsider and accept previously rejected transactions, but ON needs to reject them. Now, the situation is similar as in the case without rejection. ON needs to recharge, but it already paid for some rejections whereas OFF pays only for recharging and accepting very small transactions.

ON disadvantaged in this way cannot achieve a better competitive ratio than $2 + \frac{\sqrt{5}-1}{2}$ □

5.4.5 Bidirectional transaction stream

In this section, we consider the most general problem setting, where for a given payment channel (ℓ, r), transactions stream along the channel (ℓ, r) in both directions. A user ℓ (resp. r) can accept or reject incoming transactions that stream from left to right (resp. right to left). Either user would incur a cost of $Rx + f_2$ for rejecting a transaction of size x. ℓ does not need to take any action when encountering transactions that stream from right to left as they simply increase the balance of ℓ in the channel (ℓ, r). Both users can also decide at any point to recharge their channel on-chain, or rebalance their channel off-chain.

Our main online algorithm ON for the bidirectional transaction stream setting is detailed in Algorithm 5.10. For simplicity, we assume that $R = 0$ in the rejection cost. This means that the cost of rejecting a single transaction of size x is simply f_2, and rebalancing an amount of x off-chain now only incurs a cost of Cf_2. Our algorithm is run by both users on a payment channel and is composed of three smaller algorithms: the first is a recharging algorithm to determine when and how much to recharge the channel on-chain. The second algorithm (Algorithm 5.8) decides whether to accept or reject new transactions and when to perform off-chain rebalancing. The last algorithm (Algorithm 5.9) describes how to store the funds received from the other user of the channel.

$(4 + 2\lceil \log C \rceil, f_1)$-recharging. ON runs an on-chain recharging algorithm similar to Algorithm 5.5 (see Lines 5 and 8 in Algorithm 5.10) but with parameters $\gamma = 4+2\lceil \log C \rceil$ and $\delta = f_1$. Since we are in the bidirectional transaction stream setting, $\textsc{Funds}$ returns the amount of funds OFF has inside the entire channel (i.e. $b(\ell) + b(r)$) given a transaction sequence.

Let us look at the period between the on-chain recharging instances of ON. From Line 8 in Algorithm 5.10, we know that ON ensures that it has more than $4 + 2\lceil \log C \rceil$ times more funds than OFF locked in the channel. These funds are distributed in the following way: ON initialises $\lceil \log C \rceil + 2$ "buckets" on each end of the channel.

We denote set of left-side buckets as B^ℓ and it consists of $B^\ell_s, B^\ell_1, \ldots, B^\ell_{\lceil \log C \rceil}, B^\ell_o$. Likewise, the set of right-side buckets is B^r and it consists of $B^r_s, B^r_1, \ldots, B^r_{\lceil \log C \rceil}, B^r_o$.

After recharging, users decide how to distribute funds in the channel, so the buckets B^ℓ_s and B^r_s are filled with $2F_{\textit{tracker}}$ funds. Buckets B^ℓ_o and B^r_o are empty (0 funds). Other buckets contain $F_{\textit{tracker}}$ funds.

Looking ahead, the funds in the i-th bucket on both sides are used to accept transactions x with a size in the interval $\left[\frac{F_{\textit{tracker}}}{2^i}, \frac{F_{\textit{tracker}}}{2^{i-1}}\right)$. The funds in B_s are used to accept transactions with a size less than $\frac{F_{\textit{tracker}}}{C}$. Finally, B_o stores excess funds coming from payments from the other side when all other buckets are full.

Transaction handling. When a transaction arrives at the channel, based on the direction of the transaction, either ℓ or r executes Algorithm 5.8 to decide whether to accept the transaction. Wlog let us assume ℓ encounters transaction $\overrightarrow{x}$. If $\frac{F_{\textit{tracker}}}{2^i} < x \leq \frac{F_{\textit{tracker}}}{2^{i-1}}$ for some $i \in [\lceil \log C \rceil]$ and B^ℓ_i has sufficient funds, the funds from B^ℓ_i are used to accept the transaction. If B^ℓ_i lacks sufficient funds for accepting x, ℓ rejects x.

Now, we consider the case where $x \leq \frac{F_{\textit{tracker}}}{C}$. If B^ℓ_s has sufficient funds, ℓ uses the funds from B^ℓ_s to accept x. If B^ℓ_s has insufficient funds to accept x, ℓ performs off-chain rebalancing with an amount such that after deducting x from B^ℓ_s, there would still be $2F_{\textit{tracker}}$ funds left in B^ℓ_s. ℓ subsequently accepts x. The required funds for off-chain rebalancing are transferred from B^r_o and B^r_s (see Lines 15 and 16 in Algorithm 5.8). Whenever $B^\ell_o > 0$ and some bucket in B^ℓ gets under its original capacity, funds are reallocated from B^ℓ_o to fill the bucket.

Handling funds coming from the other side. When a transaction x is accepted by wlog ℓ, ON calls Algorithm 5.9 to distribute the transferred funds among r's buckets in the following way: r first uses x to fill B^r_s up to its capacity of $2F_{\textit{tracker}}$ (see Line 2 in Algorithm 5.9). If there are still funds left, r refills the B^r_i buckets in descending order from $i = \lceil \log C \rceil$ to $i = 1$. Intuitively, the reason why buckets are refilled in descending order is due to our simplified cost model for this problem where we assume the cost of rejection for any transaction is f_2. Thus, rejecting three small transactions size x costs thrice as much as rejecting a larger transaction of size $3x$. Finally, if there are still some funds left, they are added to B^r_o.

Our main theorem shows that our main algorithm is $7 + 2\lceil \log C \rceil$ competitive.

Theorem 5.4.7. *Algorithm 5.10 is $7 + 2\lceil \log C \rceil$ competitive.*

Proof. We know that for any i, $\mathrm{COST}_{\mathrm{OFF}}(X_i) \geq \mathrm{COST}_{\mathrm{OFF}}(X_{i-1})$. From Lemma 5.4.1 and Lemma 5.4.2, we know that cost of ON for recharging (in Algorithm 5.10) is

Algorithm 5.8: Decision on transaction

```
DECIDE(F_tracker, x, B^sdr, B^rcv)
    Status ← Accept
    if F_tracker/2^i < x ≤ F_tracker/2^(i−1) and x ≤ B_i^sdr then
        Accept x
        X ← min(F_tracker, B_i^sdr − x + B_o^sdr)
        B_o^sdr ← max(0, B_i^sdr − x + B_o^sdr − F_tracker)
        B_i^sdr ← X
    else if x_i ≤ F_tracker/C and x ≤ B_s^sdr then
        Accept x
        X ← min(2F_tracker, B_s^sdr − x + B_o^sdr)
        B_o^sdr ← max(0, B_s^sdr − x + B_o^sdr − 2F_tracker)
        B_s^sdr ← X
    else if x_i ≤ F_tracker/C and x > B_s^sdr then
        Do off-chain rebalancing to fill B_s and pay f_2 C
        B_o^rcv ← B_o^rcv − (2F_tracker − B_s^sdr)
        B_s^rcv ← B_s^rcv − x
        Accept x
        B_s^sdr ← 2F_tracker
    else
        Reject x
        Status ← Reject
    return (B^sdr, B^rcv, Status)
```

at most $(5 + 2\lceil \log C \rceil)\text{Cost}_{\text{OFF}}(X_t)$. Let t_1 and t_2 (with $t_2 > t_1$) be any two consecutive times ON recharges. We will show that the cost of ON for rebalancing and rejection is smaller than $2(\text{Cost}_{\text{OFF}}(X_{t_2}) - \text{Cost}_{\text{OFF}}(X_{t_1}))$. Then $\text{Cost}_{\text{ON}}(X_t) \leq (7 + 2\lceil \log C \rceil)\text{Cost}_{\text{OFF}}(X_t)$.

For every strategy of OFF and any two consecutive recharging times t_1 and t_2, we show that the rebalancing and rejection cost of ON between times t_1 and t_2 is at most twice that of OFF as defined by the strategy. Having the strategy of OFF, we split the time between rechargings even further, into epochs; we will show that the competitive ratio of 2 holds for every epoch.

The left epoch starts with the first transaction that makes some bucket in B^ℓ non-full (smaller than the original amount); the left epoch ends either before ON recharges, or $B_o^\ell > 0$. In a left epoch, every transaction from the right side is accepted; non-fullness of some buckets on one side means $B_o > 0$ on the other side. The right epoch is

Algorithm 5.9: Handling funds coming from the other side

```
HandleFunds(F_tracker, x, B)
    X ← min(2F_tracker, B_s + x)
    x ← max(x + B_s − 2F_tracker, 0)
    B_s ← X
    for i ∈ [⌈log C⌉] in decreasing order do
        if x > 0 then
            X ← min(F_tracker, B_i + x)
            x ← max(x + B_i − F_tracker, 0)
            B_i ← X
    B_o ← B_o + x
    return (B)
```

Algorithm 5.10: Main algorithm

```
Initialise left and right side buckets B^ℓ, B^r
Initialise tracker F_tracker, X ← 0, ∅
for transaction x in order of arrival do
    concatenate x to X
    F'_tracker ← Funds(X)
    if F'_tracker > F_tracker then
        F_tracker ← F'_tracker + f_1
        recharge to 2(2 + ⌈log C⌉)F_tracker
    sdr, rcv ← ℓ, r
    if x is from right to left then
        sdr, rcv ← r, ℓ
    B^sdr, B^rcv, Status ← Decide(F_tracker, x, B^sdr, B^rcv)
    if Status == Accept then
        B^rcv ← HandleFunds(F_tracker, x, B^rcv)
```

defined similarly, but since the epochs are disjoint, we can prove the statement for a left epoch only.

For transactions below $\frac{F_{tracker}}{C}$, we argue that the cost of ON is at most the cost of OFF. ON accepts everything, so ON pays only for rebalancing. OFF either rebalances too, in which case the cost is the same as ON; or it rejected some transactions. Since ON starts with $2F_{tracker}$ funds in B_s and refills the bucket with the highest priority, this means OFF rejected some transactions summing to at least $F_{tracker}$. There are at least

C of them, so OFF's cost is also above Cf_2. If there is a counterexample containing a small transaction that OFF rejects, then we can modify it to a counterexample where the transaction is increased to $\frac{F_{tracker}}{C}$. So we can show the ratio in the case that no small transactions are coming.

Now that we have the strategy for OFF (decisions before rebalancing), we define some variables that track the competitive ratio. We will look at incoming transactions, and prove that the competitive ratio is always below 2. We say that a transaction x belongs to a bucket B_i if $\frac{F_{tracker}}{2^i} < x \leq \frac{F_{tracker}}{2^{i-1}}$. A transaction is red if it is rejected by ON and accepted by OFF and it is blue if it is accepted by ON and rejected by OFF. Let ρ_i (β_i) be the number of red (blue) transactions in the bucket B_i. We can disregard transactions for which ON and OFF make the same decision. If the transaction is rejected by both, it improves the ratio. If the transaction is accepted by both, it can be simulated by decreasing $F_{tracker}$.

We prove by induction that $\sum_{k\leq j} \rho_k \leq 2\sum_{k\leq j} \beta_k$ for all $j \in [\lceil \log C \rceil]$. We show that if the equality holds and OFF has enough funds to accept incoming transactions, ON accepts too.

Let us examine $j = \lceil \log C \rceil$. We know that the ratio between any two transaction sizes in $B_{\lceil \log C \rceil}$ is less than 2, so any two red transactions are bigger than one blue. Moreover, all funds that arrived from the right side were put into $B_{\lceil \log C \rceil}$ (if it is not full). So if $\rho_{\lceil \log C \rceil} = 2\beta_{\lceil \log C \rceil}$, ON has at least the amount of funds in $B_{\lceil \log C \rceil}$ OFF has. To continue the induction for buckets with smaller indices, we reassign some red or blue transactions to different buckets. If $\rho_{\lceil \log C \rceil} < 2\beta_{\lceil \log C \rceil}$, we move at most one red and some blue transactions (in decreasing order of size) to $B_{\lceil \log C \rceil - 1}$, stopping just before $\rho_{\lceil \log C \rceil} \geq 2\beta_{\lceil \log C \rceil}$.

For general j, we know that, due to the reassignment, in every bucket smaller than j, ON rejected exactly twice the number of transactions OFF did. Moreover, OFF needs to use at least the same amount of funds to accept red transactions compared to funds needed by ON to accept blue ones. Now, in the bucket B_j holds $\rho_j = 2\beta_j$. Again, we pair every two red transactions to one blue, such that the sum of red is bigger than blue. Before the reassignment, the ratio between any two transactions is at most 2. The reassignment (if occurred) moved at most one red and at least one blue that is smaller than any original transaction in the bucket, so we can pair the moved red to moved blue. In transactions in buckets in j and bigger, OFF used more funds than ON. Any funds that arrived from the right side were put into some bucket in j or below, so if OFF has enough funds to accept, ON has too.

The same argument holds for a right-epoch, and we note that epochs are disjoint and cover the entire transaction sequence between times t_1 and t_2. Since we chose the consecutive recharging times t_1 and t_2 arbitrarily, the rebalancing and rejection cost of

ON between any two consecutive rechargings is at most twice that of OFF within the same period. Therefore, Algorithm 5.10 is $7 + 2\lceil \log C \rceil$ competitive. □

5.4.6 Empirical Evaluation

Methodology. We consider the performance of Algorithm 5.10 on randomly generated transaction sequences. We compare it with the optimal offline algorithm OFF. Since computing the optimal solution is NP-hard (refer to Section 5.3.2), we use dynamic programming to compute the cost. For clarity of presentation, we defer the full algorithm to Appendix A.1.

Average performance of ON

We first report on the competitive ratio our online algorithm achieves under random transaction sequences. We begin by sampling 50 random transaction sequences of length 50 each. In each sequence, transaction sizes are first sampled independently from the folded normal distribution with mean 0 and standard deviation 3, and then we sample the direction of the transaction (left-to-right or right-to-left) uniformly at random. Finally, we quantise the size of the transaction to the closest integer. We then run both OFF and ON on the generated sequences and compute the average of the following:

1. cost (sum of rejection/off-chain rebalancing/on-chain recharging costs)
2. sum of funds locked in the channel ($A(X)$)
3. acceptance rate (fraction of transactions accepted)
4. funds that are transferred across the channel by off-chain rebalancing per sequence
5. number of on-chain rechargings per sequence

We present our results in Table 5.2. As we can see from the cost of ON vs OFF in Table 5.2, the competitive ratio is generally significantly lower than the $7 + 2\lceil \log C \rceil$ bound as suggested by our conservative worst-case analysis in Theorem 5.3.9.

Limitations of ON. We notice in our experiments that ON seems to overcharge the channel. This is most noticeable when we observe the effect of C on the performance of ON. From Table 5.2, increasing C in a range of medium (not too small) values does not change OFF's cost noticeably. In contrast, both the average cost and total amount of locked funds of ON grows with C. This is due to the fact that ON uses $(4 + 2\lceil \log C \rceil, f_1)$-recharging to ensure that it always has significantly more funds

Param		OFF					ON				
C	f_2	Cost	$A(X)$	Acc	Reb	Rech	Cost	$A(X)$	Acc	Reb	Rech
2	0.5	**15.02**	6.4	0.78	0.8	1	**63.3**	44.26	0.50	0.9	2.18
8	0.5	**15.21**	6.38	0.77	0	1	**87.79**	69.06	0.50	0	2.04
2	2	**23.6**	14.26	0.95	5.36	1	**127.02**	100.2	0.91	0.38	5.86
8	2	**24.5**	13.9	0.92	0	1	**184.32**	156.6	0.9	0	5.84
Param		ON-I					ON-II				
C	f_2	Cost	$A(X)$	Acc	Reb	Rech	Cost	$A(X)$	Acc	Reb	Rech
2	0.5	**30.74**	6.86	0.44	11.18	2.18	**26.52**	5.56	0.41	10.22	1.2
8	0.5	**39.98**	19.86	0.48	0	2.04	**32.94**	15.9	0.46	0	1.18
2	2	**66.16**	14.42	0.84	25.48	5.86	**60.38**	11.3	0.78	27.16	2.2
8	2	**81.35**	42.72	0.89	1.5	5.84	**58.38**	33.3	0.83	1.56	2.16

Table 5.2: Comparison between the performance of OFF, ON, ON-I and ON-II on randomly generated transaction streams. The result is averaged over 50 sequences each of length 50. The size of each transaction is independently sampled from the folded normal distribution with mean 0 and standard deviation 3, then quantised to the closest integer. We set $f_1 = 3$ and $R = 0$. $A(X)$ is the total amount of funds in the channel from recharging the channel. "Acc" shows the average fraction of transactions that were accepted. "Reb" shows how much funds on average was moved along the channel using off-chain rebalancing. "Rech" shows the average number of rechargings performed. Note that since OFF knows the entire sequence in advance, it only recharges the channel once at the beginning of each sequence.

than OFF, even though a big fraction of these funds remain unspent. ON is also limited by the fact that it does not borrow funds from other buckets when a bucket is depleted. For instance, ON always charges B_s to $2F_{tracker}$ and only uses these funds to accept transactions that are smaller than $\frac{F_{tracker}}{C}$. Thus, as C increases, the number of transactions that fall into the B_s bucket decreases and the funds in B_s remain unspent.

Heuristics to improve the performance of ON These observations motivate us to design a less pessimistic version of ON that we expect will perform better than ON. We introduce ON-I which is a slightly altered version of ON: ON-I follows the $(\lceil \log C \rceil, f_1)$-recharging algorithm and does not divide the funds into separate buckets. Instead, ON-I accepts all the transactions smaller than $F_{tracker}$ as long as it has funds. Otherwise, if the transaction is small ($< \frac{F_{tracker}}{C}$) it off-chain rebalances to fill the bucket and accepts the transaction. Similar to ON, ON-I rejects a transaction if it is larger than $F_{tracker}$.

Our empirical results in Table 5.2 confirms that the average cost of ON-I is significantly

smaller than ON. The acceptance rate of ON-I is slightly smaller than ON, which is expected as ON-I does not have separate funds for each range of transactions (as defined by the buckets), and as a result might miss some transactions. We observe that ON-I performs more off-chain rebalancings compared to ON on average because ON-I does not reserve separate funds for small transactions. However, one issue with both ON and ON-I is that they recharge the channel too often (as soon as $F_{tracker} < A(X_t)$). As can be seen from Table 5.2 ($f_2 = 2$), both algorithms perform more than 5 rechargings on average for transaction sequences of length 50. This increases the cost of both algorithms significantly as each recharging instance incurs a cost of at least f_1.

We thus design another version of ON-I to address the aforementioned problem. ON-II works exactly as ON-I except that it does not recharge the channel as frequently as ON-I does. ON-II only recharges the channel if $\alpha \cdot F_{tracker} < A(X)$, where $\alpha > 1$ is some constant that controls the how often the algorithm recharges the channel and can be fine-tuned empirically based on f_1 and f_2. If we set $\alpha = 1$, ON-II becomes equivalent to ON-I and has higher acceptance rate. This is favourable when f_2 is large and f_1 is small. Conversely, by increasing α, ON-II recharges the channel less frequently but the acceptance rate falls. This is favourable when f_1 is large and f_2 is small. In our experiments, we observe that for the case $f_2 = 2, C = 2$, when all the other parameters are as Table 5.2, $\alpha = 2$ yields the lowest average cost. Thus, in our evaluation of ON-II, we use $\alpha = 2$ and from Table 5.2 we note that this choice of α halves the number of rechargings compared to ON-I, which consequently leads to lower average cost. Additionally, we note that the total amount of funds in the channel of ON-II is close to OFF.

Transaction size variance. We also evaluate how the performance of our algorithms is affected by the variance of the transaction size. We sample 50 sequences each of length 50 with each transaction in the sequence independently sampled from the folded normal distribution with mean 0 and standard deviation σ, for a range of σ values across $[3, 20]$. The transactions are quantised to the closest integer, and the direction of each transaction is sampled uniformly at random. We then observe the cost of ON, ON-I,ON-II and OFF. As can be seen from in Figure 5.2, the cost of all 4 algorithms rises as σ increases. This is due to the fact that increasing the variance of the sampled transactions reduces the probability of getting a similarly sized transaction coming from the other side, thus increasing the speed at which the balance on one side gets depleted. We note, however, that Figure 5.2 shows that even for large values of σ, ON's average cost remains a lot smaller than the worst case upper bound of $7 + 2\lceil \log C \rceil$. We also observe that the cost of ON-I and ON-II is noticeably smaller than ON and grows at a much slower pace with respect to increases in σ.

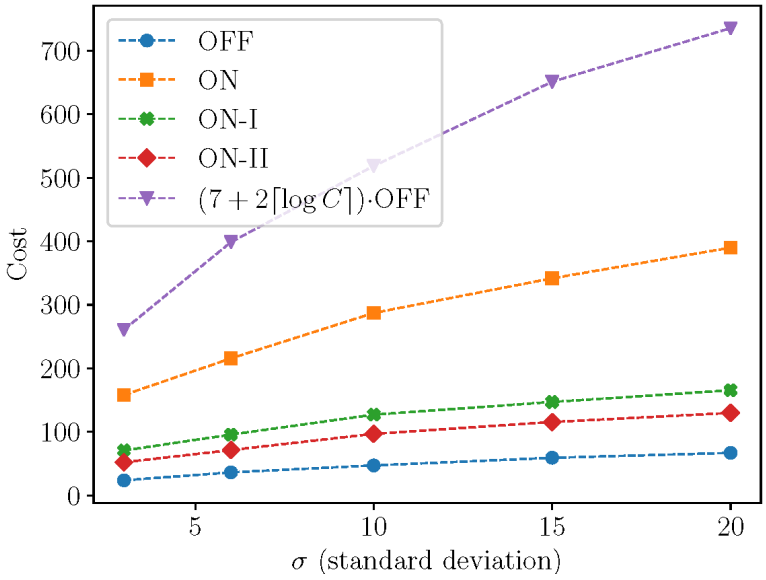


Figure 5.2: Average cost of our algorithms over 50 randomly generated transaction streams, each of length 50. The size of each transaction is independently sampled from the folded normal distribution with mean 0 and standard deviation σ. We use the parameters $f_1 = 3, f_2 = 2, R = 0, C = 4, \alpha = 2$.

Transaction flow symmetry. Another factor we look at is how the asymmetry of the transaction flow along a channel can affect the performance of our algorithms. To do so, we generate 50 sequences each of length 50, and sample the size of each transaction from a folded normal distribution with mean 0 and standard deviation 3, and then quantise the transaction to the nearest integer. We then sample the direction of the transactions according to a Bernoulli distribution with parameter p, where p represents the probability of sampling a left-to-right transaction. We see from Figure 5.3 that the cost of all algorithms decrease as p increases from 0 to 0.5. As p increases from 0.5 to 1, the cost function increases again. This conforms to our intuition that extremely asymmetric sequences are harder to handle as the lack of sufficiently many transactions from one side just increases the speed at which the balance on the other side gets depleted. We observe that both ON-I and ON-II nevertheless perform comparatively better than ON when given these asymmetric transaction sequences (see Figure 5.3).

Case Study: Lightning Network

Lightning Network parameters. We conclude our evaluation section with a case study of the Lightning network. We first run our experiments with realistic parameters

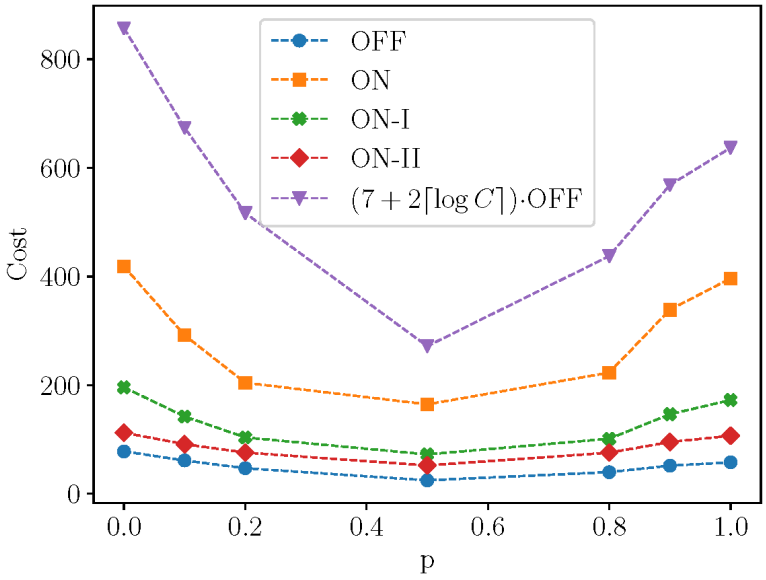


Figure 5.3: Average cost of our algorithms over 50 randomly generated transaction streams each of length 50. The size of each transaction is sampled from the folded normal distribution with mean 0 and standard deviation 3. p is the probability of sampling a left-to-right transaction. We use the parameters $f_1 = 3, f_2 = 2, R = 0, C = 4, \alpha = 2$.

taken from Lightning Network data. In the Lightning Network, f_1 is the on-chain transaction fee (roughly around 1000 satoshi) which is a lot larger than f_2, the base fee one receives when forwarding a payment (around 1 satoshi).

Average rebalancing cycle length. Next, we conduct an empirical estimation of the average rebalancing cycle length in the Lightning Network using the latest snapshot (September 2021) of the Lightning Network from the Lightning Network gossip repository [Dec]. This is important as C is an integral component of the competitive ratio in our main theorem, Theorem 5.3.9. The Lightning Network snapshot contains $117,894$ channels, including pairs of users with more than one channel between them. After merging these channel pairs, we get $63,820$ channels. We compute the length of the shortest cycle containing both ℓ and r for every pair of channel holders (ℓ, r). Table 5.3 shows the frequency of each cycle length (including the channels that are not part of any cycle at all). From Table 5.3, we see that a large fraction of channel holders are part of cycles of length at least 4. We note, however, that actual cycles might be longer as precise balance information is hidden in the Lightning Network snapshot.

Cycle length	≤ 4	5	6	7	N.A.
Frequency	$49,424$	$7,758$	469	12	$6,157$

Table 5.3: Frequency of the length of shortest cycle between all users in the Lightning Network. The last column shows the frequency of channels that are not part of any cycle. The average cycle length is 4.15

5.5 Future Work

This chapter analyses the problem of optimising the decision making process of users in a PCN in both the offline and online setting. In the offline setting, we present a constant approximation algorithm for transaction selection for a channel problem. We then show how the offline and online settings are linked, by showing how our main algorithm in the offline setting can be used as an oracle for our algorithms in the online setting. In the online setting, we provide several deterministic online algorithms for various problem settings that minimise cost while maximising liquidity and transaction throughput in a payment channel. Our algorithms come with formal worst-case guarantees, and also perform well in realistic scenarios in simulations.

The analysis in this chapter opens several interesting avenues for future research. In both the offline and online settings, it would be good to extend our algorithms from a single channel to an entire network. On the theoretical front, it would be interesting to close the gap in the achievable competitive ratio, and to explore the implications of our approach on other classic online admission control problems. Finally, we have mainly focused on deterministic algorithms in the online setting. Thus, it would be interesting to study the power of randomised approaches in this context, or to consider different adversarial models.

CHAPTER 6

Algorithms for optimising decisions for new users in PCNs

> All models are wrong, but some are useful.
>
> George Box

This chapter is based on the following publication:

- Zeta Avarikioti, Tomasz Lizurej, Tomasz Michalak, Michelle Yeo.† *Lightning Creation Games*. In 43rd IEEE International Conference on Distributed Computing Systems, **ICDCS 2023**

6.1 Introduction

Optimisation problem for a new user. In Chapter 5 we looked at the optimisation problem faced by an *existing user* in a PCN. In this chapter, we will adopt a different but complementary perspective and look at the optimisation problem for a *new user* who has not yet joined the PCN. From the perspective of a new user, two important considerations when joining a PCN are *where to connect* and *what portion of a budget should be locked* to distinct channels. This is important as their choice not only affects the situation of individual nodes but also influences the resulting network as a whole. However, this issue has been weakly studied in the literature. In fact, most PCN implementations (e.g., the Lightning Network) still propose a simple heuristic for new nodes, suggesting connecting to a trusted peer or a hub.

Key challenge. We answer this question by first presenting several attachment strategies for newly-joining nodes in a PCN. The first key challenge to this task is to model the new node's utility function that accurately reflects the key objectives of new PCN users. A newcomer has to weigh the cost of creating channels and locking up capital against the profits stemming from these connections and the node's position in the network. Furthermore, the utility function should be efficiently computable, so that it can be used in practice by new nodes, posing a second challenge.

Prior approaches and drawbacks. Unfortunately, the models of the utility function considered so far in the literature do not take all the above aspects into account. In particular, Guasoni et al. [GHS21] analyse the cost of channel creation, and establish conditions under which two parties would create unidirectional or bidirectional channels between themselves, as opposed to transacting on-chain. However, the utility function in [GHS21] only accounts for the cost of channel creation but neglects profits from routing transactions and fees a user could encounter. Avarikioti et al. [ASW19, AHWW20] and Ersoy et al. [ERE20b], on the other hand, account for fees and profits from routing transactions through the PCN but neglect the opportunity costs from the locked capital and consider only a simplified transaction model where users transact with each other with uniform probability.

Our approach. We take up the first challenge to model a utility function that accurately depicts the gains and costs of newly joining nodes. In particular, we account for on-chain costs for opening channels, routing fees paid to and by the node due to its position in the network, and opportunity costs for locked capital. We further leverage a realistic transaction distribution where nodes transact with other nodes with probability proportional to their degree, inspired by the well-known Barabási-Albert preferential attachment model [BA99]. We believe this transaction distribution approximates well real-life scenarios where nodes transact more often with big vendors and service providers. We further address the second challenge by providing a series of approximation algorithms to efficiently compute the optimal connection strategy for newly-joining nodes. The approximation ratio and runtime of each algorithm depend on how much freedom the node has to distribute its budget on the channels, highlighting an interesting trade-off.

Stable network topologies. Apart from the myopic analysis for a single joining node, we also examine the effect our strategies may have on the topological structure of a PCN. In particular, we examine simple graph structures, i.e., circle and star graphs, to determine under which conditions these constitute stable graphs, where no node may increase its utility by changing its strategy (Nash equilibrium). Naturally, which topologies are stable or not heavily depends on the parameters of the transaction

distribution. We thus identify the exact parameter space in which each topology constitutes a Nash equilibrium. Our results in this domain further advances a growing line of work that attempts to understand the game-theoretic aspects of these networks either with respect to security e.g., [RAKM21, AKWZ21b, AL22, MBB+19a, ALW20], or economics, e.g., [WGL+22, vER21b], in addition to purely analysing the cryptographic underpinnings of the PCN's security proofs [KL19, MMSK+17].

Contribution. Our contributions in this chapter can be summarised as follows:

- We extend the utility function of [AHWW20] to incorporate a *realistic transaction model and opportunity costs*. To that end, we consider transaction distributions where users transact with other users in proportion to their degree instead of uniformly at random as in [AHWW20, ERE20b, ASW19].
- Applying standard optimisation algorithms to optimise our utility function does not provide guarantees on the approximation ratio, as our utility function is submodular but non-monotone and not necessarily non-negative. We provide a series of *approximation algorithms* that maximize our utility function under different constraints. In particular, we identify a *trade-off between the runtime of the algorithm and capital distribution constraints*, i.e., how much capital is locked in each channel at its creation.
- Finally, we examine simple graph topologies and determine the parameter space of the transaction distribution in which they form *Nash equilibria*.

6.2 Related work

Strategic aspects of cryptocurrencies, and more generally the blockchain technologies, have attracted a lot of attention in the literature [CXG+21, AGBPBTP20, LBS+15] as by their very nature, they are created to facilitate interactions between self-interested parties in a decentralised manner.

Network creation games. Apart from the works discussed in the introduction ([AHWW20, ASW19, ERE20b, GHS21]), perhaps the closest research line to which our paper contributes is the one on creation games. In a well-known work by Fabrikant et al. [FLM+03b], players choose a subset of other players to connect to in order to minimise their total distance to all others in the network. The result of Fabrikant et al. was later strengthened by Albers et al. [AEE+14], and also extended to the weighted network creation game setting. Ehsani et al. [EFM+11] considers the network creation game with a fixed budget for each player, thus constraining the number of connections

each player can make. Another well-known body of research of this kind are network formation games [BG00, Jac05]. All of these works, however, consider the problem of network creation in general networks which do not take into account fees and channel collateral which are specific to PCNs.

Optimising channel creation. There are several works that focus on optimising channel creation in the PCN setting [GHS21, AHWW20, ERE20b]. Avarikioti et al. [AHWW20] considers the problem of who to create channels with in a PCN, taking into account graph centrality measures like betweenness and closeness. Ersoy et al. [ERE20b] also considers the same problem, but focuses on choosing the fees for each created channel so as to maximise profit. However, both these works only consider a limited transaction model where users transact uniformly with all other users in the network. In contrast, our work considers a more realistic transaction model where users transact with other users in proportion to their degree. Guasoni et al. [GHS21] provides a rich model and analysis of the cost of channel creation, and establish conditions under which two parties would create unidirectional or bidirectional channels between themselves, as opposed to transacting on-chain. However, the utility function in [GHS21] only accounts for the cost of channel creation, whereas the utility function in our work not only considers the channel creation cost, but also the potential profit from routing transactions and fees a user could encounter.

Stable network topologies. Our work is also closely related to the study of stable network topologies for real-world networks (e.g. social and communication networks) that are formed by the interaction of rational agents [DHMZ07, BFL$^+$21]. Demaine et al. [DHMZ07] show that all equilibrium networks satisfy the small world property, that is, these networks have small diameters. Bilo et al. [BFL$^+$21] establish properties on the diameter, clustering and degree distribution for equilibrium networks. In [ASW19, AHWW20], Avarikioti et al. consider stable graph topologies in the context of PCNs. Our work extends the analysis of Avarikioti et al. [AHWW20] and considers stable graph topologies in PCNs under a non-uniform distribution of transactions between users.

6.3 The Model

In this section, we outline our model which is an extension of the model introduced in [AHWW20]. We alleviate several unrealistic assumptions introduced in [AHWW20], thus providing more meaningful insights on the connection strategies and expected network structure of PCNs. We indicate these assumptions below. We model a PCN with a directed graph $G = (V, E)$ with $|V| = n$. Each (bidirectional) channel among two parties is represented by 2 directed edges (one in each direction) connecting the two vertices corresponding to the parties. We model each bidirectional channel as 2

directed edges to take into account the balance on both ends of the channel which can be different and thus impose different limits on the payment amount that can be sent in each direction. For node $u \in V$, let $Ne(u)$ denote the set of in- and out-neighbours of u.

6.3.1 PCN transactions

In the following, we alleviate the assumption of [AHWW20] that transactions are uniformly distributed among the PCN users, and introduce a more realistic transaction model.

Transactions. We assume transactions (tx) are of size at most $T > 0$ and all intermediary nodes impose the same (global) fee function $F : [0, T] \longrightarrow \mathbb{R}^+$. We denote by f_{avg} the value of the average (over all transaction sizes) fee when using the global fee function F. That is, $f_{avg} = \int_0^T p_{tx \text{ size}=t} \cdot F(t)dt$, where $p_{\text{tx size}=t}$ is a global probability of occurrence of a transaction with size t. We assume that f_{avg} is publicly known (recall that the fee functions are publicly announced in PCNs).

Let N_u denote the average number of transactions sent from user u over a unit of time. We denote by $N = \sum_{u \in V} N_u$ the sum of all transactions sent by all users in the PCN in a unit of time. We assume a user u joining the network knows the distribution of transactions in the network. These assumptions equally allow each user to estimate the mean rate (denoted by λ_{uv}) of transactions going along any directed edge (u, v) which, we assume, follows a Poisson process with rate λ_{uv}. We also stress that this estimation can be done efficiently in time $\mathcal{O}(n^2)$, by calculating shortest paths using e.g., Dijkstra's algorithm [Dij59] for each pair of nodes in the network.

Reduced subgraph with updated capacities. The topology of the PCN can change with the size of transactions due to balance constraints: some directed edges do not have enough capacity to forward transactions when the transaction size is too large. However, given that we assume users know the distribution of transactions in the network, and that the capacity and time of channel creation are publicly posted on the blockchain, users can estimate the expected balance on each end of all channels in the network. Thus, for the rest of the paper, we consider that all our proposed algorithms for a given transaction of size x are computed on a subgraph G' of the original PCN G that only takes into account directed edges that have enough capacity to forward x.

Transaction distribution. In this work, we assume that the probability that any two users transact with each other is proportionate to their degree. Specifically, we use the *Zipf distribution* [Zip49] to model the probability of any two users transacting with each other. In the original definition of the Zipf distribution, these probabilities are

computed for a given user u in the network by first ranking of all other users according to their degree, breaking ties arbitrarily. That is, the user with the highest degree is given rank 1, the user with the second highest degree is given rank 2, etc. Then, for some user-specific parameter $s_u > 0$, the probability $p^{tx}_{u,v}$ that u transacts with another user $v \in V \setminus \{u\}$ with rank k is:

$$p^{tx}_{u,v} = \frac{1/k^{s_u}}{\sum_{i=1}^{n} 1/i^{s_u}}. \tag{6.1}$$

We note that the Zipf distribution is widely used in the natural and social sciences for modelling data with a power law distribution [SAPP15, ACL16]. It is also frequently used in the context of social networks [BFL$^+$21] and thus seems a natural model for approximating the probability of any 2 users transacting in a payment channel network.

At this point, a careful reader might observe that the probability of a given user transacting with two other users of the same degree, as computed using the original definition of the Zipf distribution, is not the same. Indeed, these probabilities can differ to a large extent, depending on how these users are ranked as well as how many users have the same degree. This seems unrealistic to model transactions in a PCN as it allows an implicit bias in transacting with users with same degree (and in PCNs like the Lightning Network the majority of users have degree 1 or 2 [Gita]). In order to make our model *indifferent* to rankings of users with similar degrees, we modify the original Zipf distribution to ensure that the probability of any user transacting with two other distinct users with the same degree is equal. We do this by simply averaging the Zipf probability of transacting with every user with the same degree, which we detail below:

Given a network $G = (V, E)$, we first consider the subgraph $G' = (V' = V \setminus \{u\}, E')$ which is created by removing the node u and all its incident edges from G. Then, we sort all nodes in V' by their *in-degree* and then assign a *rank-factor* (denoted by $rf(v)$) to each node v in V'. Since we want to ensure that every node with the same in-degree has the same rank-factor, we simply average the ranks (as computed using the original Zipf definition) of nodes with the same in-degree. In more detail, let $r_0(v)$ denote the smallest rank of a node $v' \in V'$ such that the in-degree of v' is equal to the in-degree of v. Let $n(v)$ be the number of nodes in V' with the same in-degree as v. The rank factor of v can be computed as follows:

$$rf(v) = \frac{\frac{1}{r_0^{s_u}(v)} + \ldots + \frac{1}{(r_0(v)+n(v))^{s_u}}}{n(v)}$$

The probability that u transacts with $v \in V'$ is then:

$$p^{tx}_{u,v} = \frac{rf(v)}{\sum_{v' \in V'} rf(v')}$$

Estimating the transaction rate. The *edge betweenness centrality* is a natural measure of the "importance" of a given edge in terms of routing transactions in a PCN. Let the edge betweenness centrality be defined as:

$$EBC(e) := \sum_{s,r \in V; s \neq r; m(s,r) > 0} \frac{m_e(s,r)}{m(s,r)},$$

where $m_e(s,r)$ is the number of shortest paths that traverse through the edge e and $m(s,r)$ is the total number of shortest paths from s to r. The transaction rate λ_e for all directed edges e in E can be estimated by the edge betweenness centrality of the edge e weighted by the probability of any two vertices s and r transacting with each other. That is, for a directed edge e, we first define the probability p_e that the edge e is chosen in a single transaction:

$$p_e = \sum_{s,r \in V; s \neq r; m(s,r) > 0} \frac{m_e(s,r)}{m(s,r)} p^{tx}_{s,r}. \tag{6.2}$$

Recall that N is the average number of transactions that happen in a unit of time sent by all users in the network, and we assume these transactions are independent. The average number of times a directed edge $e = (u,v)$ is chosen in N transactions is the transaction rate λ_e and is simply $N \cdot p_e$.

6.3.2 Utility function of a new user

When a new user joins a PCN, they must decide *which channels to create and how much capital to lock in each channel*, while respecting their own budget. In their decision, the user must factor the following: (a) the on-chain costs of the channels they choose to open, (b) the opportunity cost from locking their capital for the lifetime of each channel, (c) the potential gains from routing transactions of others (routing fees), (d) the routing fees they must pay to route their own transactions through the PCN, (e) their budget. Intuitively, the more channels a user opens and the higher the amount of the total capital locked, the more fees they will obtain from routing and the less cost they will bear for routing their own transactions. In other words, increasing the initial costs also increases the potential gains. Our goal is to analyze these trade-offs and find the sweet spot that maximizes the benefits for a newly-joining user with a specific budget. We account for all these factors in a realistic manner when we design the utility function of the user, in contrast to previous work [AHWW20, ASW19] where the opportunity cost was omitted, and the routing fees were calculated naively (i.e., constant fees and uniform transaction distribution).

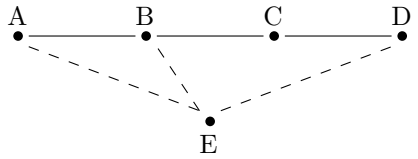


Figure 6.1: E joins a PCN with existing users A, B, C, D. E plans to transact with B once a month, and A usually makes 9 transactions with D each month. We assume the transactions are of equal size, and transaction fees and costs are of equal size. E has enough budget only for 2 channels, with the spare amount of funds to lock equaling 19 coins. E should create channels with A and D of sizes 10 and 9 to maximize the intermediary revenue and minimize E's own transaction costs, instead of creating a channel only with B. Although E would now have to pay 1 satoshi for routing the payment to B through A, A would now choose to route their payments through E as the fee per payment for A would be 1 satoshi instead of 2. E would gain 10 satoshi from routing A's payment to D for a net revenue of 9 satoshi.

User strategy and constraints. Consider a fixed PCN $G = (V, E)$ and a new user u that wants to join G. We denote the set of possible actions that are available to u by $\Omega := \{(v_i, l_i)\}_i$, where each element $(v_i, l_i) \in \Omega$ represents a node v_i that u wants to connect to by locking in an amount of $l_i > 0$ on the corresponding channel with v_i. The *strategy* of u is to select a set of users $S \subseteq \Omega$ that u wants to connect to and how much funds to deposit in these channels. Note that both Ω and S may contain more than one channel with the same endpoints, but different amounts of locked funds on each end. We also assume u has some budget $B_u > 0$ to create and fund channels and u's budget constraint imposes the requirement that for the strategy $S \subseteq \Omega$ chosen by u, $\sum_{j=1}^{|S|}[C + l_j] \leq B_u$, where C represents the channel creation fee. Finally, we remark that we can either constrain the amount locked inside channels to be discrete or continuous. We call the case where each l_i can take on continuously many values in the range $[0, B_u]$ a *continuous action set*. Figure 6.1 highlights a simple example of the decision-making process of a new user that wants to join an existing PCN.

Now we define the *utility function* for a new user u that wants to join a PCN G and has a fixed budget B_u. The goal of u is to *choose any strategy* $S = \{(v_i, l_i)\}_i \subseteq \Omega$ *to maximize their expected profit within a given budget* B_u. The expected profit (utility) of u is essentially the expected revenue from forwarding transactions through its channels and collecting the routing fees, minus the costs of creating the channels (on-chain fees and opportunity cost) and the expected fees encountered by u when sending transactions to other users in the network.

Channel costs. Typically, two on-chain transactions are needed to open and close a channel *. Recall that each blockchain transaction costs a fee to the miners, denoted C.

The cost of the *opening* transaction can be shared by two parties, and we assume that parties only agree to open channels if they share this cost equally ($C/2$ each). The cost of the closing transaction, on the other hand, is paid by both parties when the channel closes in collaboration, or by the party that closes the channel when the channel is closed unilaterally. To model the costs of a channel between u and v, we assume that it is equally probable that the channel closes in one of the three ways: unilaterally by v, unilaterally by u, in collaboration of u and v. Thus, the cost of the closing transaction is in expectation $C/2$ for each party. Hence, in total, the channel cost for each party is C.

We also account for the opportunity cost of locking funds (as opposed to using or storing them elsewhere) in a channel for the lifetime of the channel. Suppose two users u and v wish to open a channel locking c_u and c_v amount of coins respectively. Let l_u be the opportunity cost defined by user u; which is, typically a function of the amount of coins c_u, e.g., $l_u = r \cdot c_u$, r constant (a standard economic assumption due to the non-specialized nature of the underlying coins [HL98]). We denote the total cost for opening the channel (u, v) for user u by $L_u(v, l) = C + l_u$. The cost of user v is symmetric.

We direct the reader to the work by Guasoni et al. [GHS21] for a more detailed model of channel costs. We note that our computational results still hold in this extended model of channel cost. We further note that while the utility function in [GHS21] only accounts for the cost of channel creation, in our work we also consider the potential profit from routing transactions and fees a user could encounter.

Revenue from routing fees. Each user of the PCN may route transactions of others through their channels in exchange for a routing fee. Each time a user u provides such service, u gains revenue equal to f_{avg} as described in section 6.3.1. Specifically, the expected revenue gained by a user u (denoted by $\mathbb{E}_u^{rev}$) over a unit time interval from routing transactions in the PCN is the sum of the fees weighted by the average transaction rate from all of u's incident channels:

$$\mathbb{E}_u^{rev} = \sum_{v_i \in Ne(u)} \lambda_{uv_i} \cdot f_{avg}. \tag{6.3}$$

*We omit the case when one of the parties may commit fraud, and a third transaction is necessary to award the cheated party all the channel funds. This case is outside the scope of the paper as the costs for publishing such a transaction may be covered by the total funds of the channel

Fees encountered by the user. Whenever a user in the network u makes a payment to another user v, u has to pay some amount of fees to all the intermediary nodes in the payment path from u to v. Let $d(u, v)$ be one less than the length of the shortest path from any two distinct users u to v in the network. That is, for any two users u, v directly connected by a payment channel $d(u, v) = 0$. The expected fees encountered by u with a stream of N_u output transactions is the sum of costs which increases proportionally with the distance between any two users:

$$\mathbb{E}_u^{fees} = N_u \cdot \sum_{v \in V; v \neq u} d(u, v) \cdot f_{avg} \cdot p_{u,v}^{tx}$$

We note that when two users u and v are not connected, then $d(u, v) = +\infty$.

Objective. Here, we combine all the costs calculated above and compute the utility function of a newly joining node. We denote the expected utility of a user u under a given strategy $S \subseteq \Omega$ by $\mathcal{U}_{u_S}$ and it is the profit gained from collecting the fees, minus the fees paid for sending out transactions, and minus the costs of the channels. Formally,

$$\mathcal{U}_{u_S} = \mathbb{E}_u^{rev} - \mathbb{E}_u^{fees} - \sum_{(v,l) \in S} L_u(v, l)$$

We assume the utility of a disconnected node (i.e. a node that is not connected to any other node in the network) is $-\infty$.

The objective of u is to select a subset of users to connect to as well as the amount of funds to lock into these channels that maximises their expected utility subject to the budget constraints. Formally:

$$\max_{S \in \Omega} \mathcal{U}_{u_S} \text{ s.t. } \sum_{(v,l_u) \in S} [C + l_u] \leq B_u$$

6.4 Optimisation algorithms

Having defined the utility and objective for a new user u in Section 6.3.2, we now propose several algorithms to optimise the objective in this section. We begin by establishing some properties of our objective function. We first show that our utility function is submodular but not necessarily monotone and not necessarily non-negative. Thus, we cannot apply standard algorithms to maximise it efficiently with good guarantees on the approximation ratio [FMV11]. We thus propose a series of constraints on the actions of the new user u and define a solution for the objective in each constrained setting.

We then provide a corresponding optimisation algorithm for each setting that comes with guarantees on the approximation ratio. In the following, let $[k]$ denote $\{1, \ldots, k\}$.

6.4.1 Properties of the objective function

We first show that the objective function is submodular. Let $S \subset \Omega$ be a strategy. Note that we allow the algorithm to add more than one channel with the same endpoint v but different amounts of funds l_i to the strategy set S.

Theorem 6.4.1. *The expected utility function $\mathcal{U}_{u_S}$ is submodular.*

Proof. We split $\mathcal{U}_{u_S}$ into three components that sum to $\mathcal{U}_{u_S}$ and show that each component is submodular. Since the sum of submodular functions is submodular, the claim follows.

We first rewrite $\mathcal{U}_{u_S}$ as

$$\mathcal{U}_{u_S} = \mathbb{E}_u^{rev} + \left(-\mathbb{E}_u^{fees}\right) + \left(-\sum_{(v,l)\in S} L_u(v,l)\right). \tag{6.4}$$

Consider two strategies $S_1 \subseteq S_2$, and consider a pair $X = (x, l_x) \notin S_2$.

We now analyse the first component of $\mathcal{U}_{u_S}$, $\mathbb{E}_u^{rev}$. We use $\mathbb{E}_{u_S}^{rev}$ to denote the expected revenue of user u given the strategy S.

Now observe that

$$\mathbb{E}_{u_{S_1\cup X}}^{rev} - \mathbb{E}_{u_{S_1}}^{rev} = \mathbb{E}_{u_X}^{rev} = \lambda_{xu} \cdot f_{avg} = \mathbb{E}_{u_{S_2\cup X}}^{rev} - \mathbb{E}_{u_{S_2}}^{rev}$$

Hence the expected revenue function $\mathbb{E}_{u_S}^{rev}$ is submodular. Note that we assume that λ_{xu} is fixed.

Now we show that the second component $-\mathbb{E}_u^{fees} = -\lambda_u \sum_{v\in V; v\neq u} d(u,v) \cdot f_{avg} \cdot p_{u,v}^{tx}$ is submodular. Let us denote the marginal gain in terms of the expected fees of adding $X = (x, l_x)$ to strategy S as $MG_S(X) := -\mathbb{E}_{u_{S\cup X}}^{fees} - (-\mathbb{E}_{u_S}^{fees}) = \mathbb{E}_{u_S}^{fees} - \mathbb{E}_{u_{S\cup X}}^{fees}$. We note that $MG_S(X)$ only changes if there is a shortest path from u to some $v \in V$ that goes through the new vertex x, i.e.:

$$MG_S(X) = \lambda_u f_{avg} \sum_{\substack{v\in V; v\neq u; \\ x\in sp_{S\cup\{X\}}(u,v)}} p_{u,v}^{tx}\Big[d_S(u,v) - d_{S\cup\{X\}}(u,v)\Big]$$

Recall that $d(u,v)$ as defined for two disconnected nodes u, v is $+\infty$. Thus, $d_{S_1\cup\{X\}}(u,v) - d_{S_1}(u,v) \leq 0$ as $X \notin S_1, S_2$. Moreover, as all vertices $v \in S_1, S_2$ are direct neighbours

of u, then $d_{S_1}(u,v) - d_{S_1 \cup \{X\}}(u,v) \geq d_{S_2}(u,v) - d_{S_2 \cup \{X\}}(u,v)$. Hence, we conclude that $MC_{S_1}(X) \geq MC_{S_2}(X)$. Note that in our calculations we assume that $p^{tx}_{u,v}$ is fixed.

Finally, we show that the last component $-\sum_{(v,l)\in S} L_u(v,l)$ in 6.4 is submodular. The marginal gain of $X = (x, l_x)$ to the channel costs given u_{S_1} is simply the cost of a single bidirectional channel between u and x, i.e. $L_u(v,x)$. This is exactly equal to the marginal gain given u_{S_2}. □

Now, we show that although the objective function is submodular, it is unfortunately non-monotone. That is, for any two strategy sets S_1, S_2 with $S_1 \subset S_2$, it is not necessarily the case that $\mathcal{U}_{u_{s_1}} \leq \mathcal{U}_{u_{s_2}}$.

Theorem 6.4.2. *The expected utility function $\mathcal{U}_{u_S}$ is not necessarily monotone*

Proof. We analyse each component of $\mathcal{U}_{u_S}$ separately. First, we note that a direct application of [ERE20b] shows that $\mathbb{E}^{rev}_u$ is monotone increasing. Next, we look at expected fees:

$$-\mathbb{E}[\text{fees encountered by } u_S] = -\lambda_u \sum_{v \in V; v \neq u} d(u,v) \cdot f_{avg} \cdot p^{tx}_{u,v}.$$

The monotonicity of this function directly follows from the fact that for any $S_1 \subseteq S_2$, $d_{S_1}(u,v) \geq d_{S_2}(u,v)$. Thus, the function is monotone increasing. Note that in the calculations we assume that $p^{tx}_{u,v}$ is a fixed value.

Finally, $-\sum_{(v,l)\in S} L_u(v,l)$ is clearly a monotone decreasing function. Since two components of $\mathcal{U}_{u_S}$ are monotonically increasing and one component is monotonically decreasing, $\mathcal{U}_{u_S}$ is non-monotone. □

It is straightforward to see that the proof of Theorem 6.4.2 leads to the following corollary.

Corollary 6.4.2.1. *The modified utility function $\mathcal{U}'_{u_S} = \mathbb{E}^{rev}_u - \mathbb{E}^{fees}_u$ is monotone.*

Proof. Follows as a direct consequence from the fact that the expected revenue and expected fees are both monotone increasing. □

The final property we show about our objective function is that it is not necessarily non-negative.

Theorem 6.4.3. *The expected utility function $\mathcal{U}_{u_S}$ is not necessarily non-negative.*

Proof. This follows from the observation that the sum of the cost of creating channels and the expected fees $\sum_{(v,l)\in S} L_u(v,l) + \mathbb{E}_u^{fees}$ might easily get bigger than the expected revenue $\mathbb{E}_u^{rev}$ when choosing some strategy $S \subseteq \Omega$. □

6.4.2 Fixed amounts of funds per channel

We first show that if we restrict the amount of funds (say l^*) that the new user u can lock in each created channel, we can achieve an approximation ratio of $1 - \frac{1}{e}$. This setting is useful for users who join the PCN to transact regularly with other users and with fixed amounts (for instance, u buys coffee from a different cafe every day in the week with each cafe selling similarly priced coffee), and so the precise amount of funds allocated in each channel is not as important who the user should connect to. The algorithm (described formally in Algorithm 6.1) that achieves this ratio in this setting is simple – the new user just greedily picks the k best channels to connect with that maximises their expected revenue minus their expected fees.

Algorithm 6.1: Greedy algorithm

Input: Ω, M

$\mathbf{P_S} \leftarrow$ array indexed $1, \ldots, \mathrm{M}$ initialised with $\mathbf{P_S}[i] = \emptyset$
$\mathbf{P_U} \leftarrow$ array indexed $1, \ldots, \mathrm{M}$ initialised with $\mathbf{P_U}[i] = -\infty$
$S \leftarrow \emptyset$
$A \leftarrow \Omega$
while $|S| \leq \mathrm{M}$ **do**
  $X \leftarrow argmax_{X \in A}[\mathcal{U}'_{u_{S\cup\{X\}}} - \mathcal{U}'_{u_S}]$
  $S \leftarrow S \cup \{X\}$
  $\mathbf{P_S}[|S|] \leftarrow S$
  $\mathbf{P_U}[|S|] \leftarrow \mathcal{U}'_{u_S}$
  $A \leftarrow A \setminus \{X\}$
$i \leftarrow argmax_{i\in\{1,\ldots,M\}}[\mathbf{P_U}[i]]$
return $\mathbf{P_S}[i]$

Formally, let us restrict our analysis to the modified utility function $\mathcal{U}'_{u_S}$ which is the sum of the expected revenue and the expected fees: $\mathcal{U}'_{u_S} = \mathbb{E}_u^{rev} + (-\mathbb{E}_u^{fees})$. We note that the modified utility function $\mathcal{U}'_{u_S}$ is submodular and monotone, as shown in Corollary 6.4.2.1. Let us denote the maximum number of channels that can be created given u's budget B_u by $\mathrm{M} := \lfloor \frac{B_u}{C+l^*} \rfloor$. Algorithm 6.1 searches for the optimal subset of vertices to connect to that maximises $\mathcal{U}'_{u_S}$ for each possible subset of size k, $k \in \{1, 2, \ldots, \mathrm{M}\}$. Once all the M optimal subsets are found, Algorithm 6.1 then compares the utility given by $\mathcal{U}'_{u_S}$ for each subset, and returns the subset that gives the maximal utility. Since the channel creation cost is now fixed for any choice of k new

channels, the $(1-\frac{1}{e})$-approximation we achieve when we greedily maximize $\mathcal{U}'_{u_S}$ simply follows from the result in [Wil19] since $\mathcal{U}'_{u_S}$ is submodular and monotone.

The next theorem shows that Algorithm 6.1 returns a $(1-\frac{1}{e})$-approximation and runs in time linear in M.

Theorem 6.4.4. *Algorithm 6.1 with inputs $\Omega = \{(v, l^*) \in V : v \neq u\}$ and M returns a $(1-\frac{1}{e})$-approximation of the optimum of $\mathcal{U}'_{u_S}$. The result is computed in at most $\mathcal{O}(\mathrm{M}n)$ number of estimations of the λ_{uv} parameter.*

Proof. To see that Algorithm 6.1 returns a $(1-\frac{1}{e})$-approximation of the optimum of $\mathcal{U}'_{u_S}$, we need to see that in the algorithm for each possible k we compute a $(1-\frac{1}{e})$-approximation of $\mathcal{U}'$ (in $\mathcal{O}(n)$ time), because the function $\mathcal{U}'$ is submodular and monotonically increasing, then the overall solution that compares partial results gives a $(1-\frac{1}{e})$-approximation ratio for a fixed k. This in turn gives a $(1-\frac{1}{e})$-approximation ratio for each $k \in \{1, 2, \ldots, \mathrm{M}\}$. □

6.4.3 Varying amount of funds per channel, discrete version

Next, we relax the previous constraint that the new user u can only lock a fixed amount of funds in each channel and allow u to now lock varying amounts of funds in each channel. Enabling varying capital on channels more accurately depicts the realistic model of transaction distribution that we leverage. However, in order to achieve the same approximation ratio of $1-\frac{1}{e}$ as in the previous setting, we have to discretise the capital that can be locked into a channel to some minimal amount $m > 0$. That is, opening a channel would require injecting funds of the form km for some $k \in \mathbb{N}$. We impose this discretisation constraint in order to perform an exhaustive search over all possible assignments of the budget B_u capitals in each channel.

We again operate on the modified utility function $\mathcal{U}'_{u_S}$ and present an algorithm (described in Algorithm 6.2) that achieves the same approximation ratio of $1-\frac{1}{e}$, albeit with a larger runtime. In more detail, given a parameter m, Algorithm 6.2 firstly divides the budget B_u to $\frac{B_u}{m}$ units that can be spent. Then, the algorithm divides these budget units into $k+1$ parts (where $k := \lfloor \frac{B_u}{C} \rfloor$ is a bound on the number of channels that u can possibly create with a budget of B_u). Finally, for each possible division of the budget into $k+1$ parts, it runs Algorithm 6.1 as a subroutine in each step locking the capital assigned to this channel in the division. Let us denote $\tau := \binom{\frac{B_u}{m}}{k+1}$.

Theorem 6.4.5. *Algorithm 6.2 with inputs V, budget B_u, and parameter m returns a $(1-\frac{1}{e})$-approximation of the optimum of $\mathcal{U}'_{u_S}$. The result is computed in at most $\mathcal{O}(\tau k n)$ steps.*

Algorithm 6.2: Exhaustive search over channel funds

Input: V, B_u, m

$k = \lfloor \frac{B_u}{C} \rfloor$

$\mathbf{D} =$ array of all divisions of $[\lfloor \frac{B_u}{m} \rfloor]$ to $k+1$ parts

$\mathbf{D_S} \leftarrow$ array indexed $1, \ldots, |\mathbf{D}|$ initialized with $\mathbf{D_S}[i] = \emptyset$

for $i \in [|\mathbf{D}|]$ **do**

  $(l_1, \ldots, l_{k+1}) \leftarrow \mathbf{D}[i]$

  $\mathbf{D_S}[i] \leftarrow$ the output of Algorithm 6.1 run on $\mathrm{M} = k$ with a restriction that in every step j of the while loop in the algorithm a channel with capacity l_j is selected

$i \leftarrow argmax_{i \in \{1,\ldots,|\mathbf{D}|\}} \mathcal{U}'_{u_{\mathbf{D_S}[i]}}$

return $\mathbf{D_S}[i]$

Proof. The budget $\frac{B_u}{m}$ can be split to at most $k+1$ parts in at most $\tau = \binom{\frac{B_u}{m}}{k+1}$ cases. Algorithm 6.1 is run as a subroutine of Algorithm 6.2 and thus the runtime of using Algorithm 6.1 for each case is $\mathcal{O}(kn)$. The main routine iterates through all possible combinations of amounts locked to channels, each of them giving the $(1 - \frac{1}{e})$-approximation for the selected assignments of funds. □

We note a natural trade-off between the choice of m and the runtime of Algorithm 6.2: a larger choice of m would reduce the search space and hence the runtime of the algorithm. However, it would reduce the control over the capital the user could lock into any particular channel.

6.4.4 Varying amount of funds per channel, continuous version

In this section, we remove the previous constraint that the capital the new user u can inject into new channels have to be discretised, that is, u can now inject funds of the form m where m lies in the real interval $[0, B_u]$ into any channel. We sketch a polynomial-time $\frac{1}{5}$-approximation algorithm for the optimisation problem: let us first denote the total expected on-chain transaction cost for a user u with an average output stream of N_u transactions as $C_u := \frac{N_u \cdot C}{2}$. That is, C_u represents the total expected cost for user u when u transacts entirely on the blockchain and not on the PCN. One can now consider what we term the benefit function, which is simply the sum of C_u and the utility of u when u joins the PCN with strategy S. Formally, we denote this function by $\mathcal{U}^b_{u_S} := C_u + \mathcal{U}_{u_S}$. Intuitively, the benefit function captures the potential benefit u would gain from transacting with other users over the PCN rather than on the blockchain, thus it is the sum of the utility the u gets from joining the PCN, as well as the expected saved costs C_u of not transacting on the blockchain.

We observe that $\mathcal{U}^b_{u_S}$ will stay submodular and positive whenever the user chooses channels (u, v), such that

$$\mathbb{E}^{fees}_u + \frac{B_u}{C} \cdot L_u(v, l) < C_u.$$

As such, we can apply the algorithm and result of Lee et al. [LMNS09] for optimising submodular and non-negative functions to $\mathcal{U}^b_{u_S}$ to achieve a $\frac{1}{5}$-approximation of $\mathcal{U}^b_{u_S}$.

6.5 Structural properties of simple graph topologies

In this section, we complement our study of optimisation algorithms for users in the payment channel network (Section 6.4) with a study of structural properties of simple graph topologies given the transaction model between users as defined in Section 6.3.1. We are particularly interested in properties of stable networks, that is, networks that are in a Nash Equilibrium where no user can increase their utility by any unilateral change in strategy. Stability is an important notion in the context of PCNs as this has implications not only on the choice of which nodes to connect to for a new user [AHWW20] but also on payment routing and finding off-chain rebalancing cycles for existing users to replenish depleted channels [APS+22b]. We are also interested in the parameter space of our model under which specific graph topologies form a Nash Equilibrium.

6.5.1 Upper bound on the longest shortest path containing a hub

An interesting question is how large is the diameter of stable networks with highly connected nodes. In the context of PCNs, this has implications on efficient payment routing algorithms[RMKG18, SVR+20b, PSSY21a, GHP00]. As a first step to answering this question, we derive an upper bound on the longest shortest path in a stable network that contains a hub node, i.e., an extremely well-connected node that transacts a lot with other nodes in the network. Let us select a hub node h and consider the longest shortest path that h lies on (if there are multiple we simply select one of them arbitrarily). We denote the length of the path by d. Let $\varepsilon > 0$. The following theorem derives an upper bound on d for a stable network.

Theorem 6.5.1. *d is upper bounded by* $2(\frac{\frac{C+\varepsilon}{2} - \lambda_e \cdot f_{avg}}{p_{\min} \cdot N \cdot f_{avg}}) + 1$.

Proof. Let $P = (v_0, v_1, \ldots, v_d)$ be the path. Consider the addition of an edge e between $v_{\lfloor \frac{d}{2} \rfloor - 1}$ and $v_{\lfloor \frac{d}{2} \rfloor + 1}$. Denote by λ_e the minimum rate of transactions going through the edge e in both directions, i.e. $\lambda_e := \min\{\lambda_{(v_{\lfloor \frac{d}{2} \rfloor - 1}, v_{\lfloor \frac{d}{2} \rfloor + 1})}, \lambda_{(v_{\lfloor \frac{d}{2} \rfloor + 1}, v_{\lfloor \frac{d}{2} \rfloor - 1})}\}$.

Now consider the set of directed shortest paths SP such that each path $s_i \in SP$ is a subsequence of P and one end point of s_i lies in $\{v_0, \ldots, v_{\lfloor \frac{d}{2} \rfloor - 1}\}$ and the other end point of s_i lies in $\{v_{\lfloor \frac{d}{2} \rfloor + 1}, \ldots, v_d\}$. Let p_i be the probability that s_i is selected, with probabilities of directed paths being selected as defined by the probability of the source of the path transacting with the sink (refer to Equation 6.1 for more details). Let $p_{\min} := \min_i p_i$.

We know the cost (split equally) of creating the edge e with some ε amount of funds locked into the edge is at least $\frac{C+\varepsilon}{2}$. Since the network is stable, this implies that the cost of creating e is larger than any benefits gained by the 2 users $v_{\lfloor \frac{d}{2} \rfloor - 1}$ and $v_{\lfloor \frac{d}{2} \rfloor + 1}$ by creating e. That is,

$$\frac{C+\varepsilon}{2} \geq \lambda_e \cdot f_{avg} + N \cdot p_{\min} \cdot f_{avg} \cdot \lfloor \frac{d}{2} \rfloor, \tag{6.5}$$

where the first term on the RHS of the inequality is the minimum (among the two parties $v_{\lfloor \frac{d}{2} \rfloor - 1}$ and $v_{\lfloor \frac{d}{2} \rfloor + 1}$) of the average revenue gained by adding the edge e. The second term on the RHS of the inequality is a lower bound on the average amount of fees saved by $v_{\lfloor \frac{d}{2} \rfloor - 1}$ and $v_{\lfloor \frac{d}{2} \rfloor + 1}$. Rearranging, this implies that $d \leq 2(\frac{\frac{C+\varepsilon}{2} - \lambda_e \cdot f_{avg}}{p_{\min} \cdot N \cdot f}_{avg}) + 1$. $\square$

Note that since a hub node is on the path, as long as it is not directly in the middle of the path (i.e. vertex $v_{\lfloor \frac{d}{2} \rfloor}$), $p_{\min}$ should be fairly large as hubs are typically high degree vertices. Moreover, if a hub node is on a diametral path, we extract a meaningful bound on the diameter of a stable network.

6.5.2 Stability of simple graph topologies

In this section, we study some simple graph topologies, and the parameter spaces of the underlying transaction distribution under which they form a Nash Equilibrium. We restrict our analysis to these simple topologies because computing Nash Equilibria for a general graph using best response dynamics is NP-hard (see Theorem 2 in [AHWW20]). As mentioned in Section 6.3.1, we assume the underlying transaction distribution that gives the probability of any two nodes transacting with each other is the Zipf distribution.

Our analysis in this section relies on the following set of assumptions and notation:

1. Recall from Equations 6.2 and 6.3 in Section 6.3 that the expected revenue of a user u can be written as:

$$\begin{aligned}\mathbb{E}_u^{rev} &= \sum_{v_i \in Ne(u)} \lambda_{uv_i} \cdot f_{avg} \\ &= \sum_{\substack{v_1 \neq v_2 \\ v_1, v_2 \in V \setminus \{u\}}} \frac{m_u(v_1, v_2)}{m(v_1, v_2)} \cdot N_{v_1} \cdot p^{tx}_{v_1, v_2} \cdot f_{avg}.\end{aligned}$$

We denote $b := N_{v_1} \cdot f_{avg}$ and assume it is constant for $v_1 \in V \setminus \{u\}$.

2. We denote $a := N_u \cdot f_{avg}$.
3. We assume a *global* scale parameter s for the Zipf distribution to model the underlying transaction distribution. For any $s > 0$ and $n \in \mathbb{N}$, we denote $H_n^s := \sum_{k=1}^{n} \frac{1}{k^s}$.
4. We assume all players create channels locking in the same amount of funds. We use l to denote the cost of creating channels in this case.

We first establish the necessary conditions that make the star graph a Nash Equilibrium. Let us denote by $K_{1,n}$ the star graph with 1 central vertex and n leaves.

Theorem 6.5.2. *$K_{1,n}$ with $n \geq 2$ is a Nash Equilibrium when nodes transact with each other according to the Zipf distribution with parameter $s \geq 0$ whenever the following conditions hold:*

1. $\frac{a}{H_n^s} \leq l \cdot 2^s$,
2. $b \cdot \frac{i}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} + a \frac{H_{i+1}^s - 1}{H_n^s} \leq l \cdot (i-2)$ *(for $2 \leq i \leq n-1$)*

Due to its length, and for clarity of presentation, we defer the proof of Theorem 6.5.2 to Appendix B.1. The main observation in the proof is that the central node would not want to switch strategies as its expected revenue under the default strategy is already maximised. Thus, any switch in strategy would simply add additional cost to the node. For leaf nodes, we enumerate all possible classes of deviations, and then show for each deviating class the expected utility of deviating is less than the expected utility under the default strategy, as long as the two conditions in the theorem statement hold.

We also show that the circle graph is not a Nash Equilibrium when the number of vertices is sufficiently large. For clarity of presentation, we defer the full proof to Appendix B.2. The main observation is that for large n, there is a better strategy for any vertex. This strategy deletes a connection with one of the existing neighbours, and connects to the opposite node.

Theorem 6.5.3. *The Circle graph is not a Nash Equilibrium for $s \geq 0$ and for large n.*

6.6 Conclusion and Future Directions

In this chapter, we modeled and analysed the incentive structure behind the creation of PCNs. We first focused on the perspective of a new user who wants to join the network in an optimal way. To this end, we defined a new user's utility function in terms of expected revenue, expected fees, on-chain cost of creating channels, and opportunity costs, while accounting for realistic transaction distributions.

We also introduced a series of approximation algorithms under specific constraints on the capital distribution during the channel creation: (a) We first presented a linear time $(1 - \frac{1}{e})$-approximation algorithm when a user locks a fixed amount to all channels; thus, providing an efficient approach for users who wish to lower computational costs. (b) We further provided a pseudo-polynomial time $(1 - \frac{1}{e})$-approximation algorithm when users may lock varying, but discretised by m, amounts to different channels. This setting applies to most real-life scenarios but comes with a computational overhead that depends on m. (c) Finally, we proposed a $1/5$ approximation solution when a user can pick the amounts from a continuous set. We used a modified utility function, the benefit function, which may be leveraged by a user to test whether assuming continuous funds yields unexpected profits.

Altogether, our results in this section show that depending on the number of assumptions a new user joining a PCN wants to make, the user has a range of solutions to deploy to optimize the way they connect to the network.

Lastly, we analysed the parameter spaces in our underlying model and conditions under which the star and circle graph topologies form a Nash Equilibrium. Our analysis indicates that under a realistic transaction model, the star graph is the predominant topology, enhancing the results of [AHWW20].

We highlight three interesting directions for future work. First, it would be beneficial to develop more advanced algorithms for maximizing the general utility function that also come with guarantees on the approximation ratio. Second, we believe there are still avenues in which our model can be made more realistic, for instance, by considering a more realistic cost model that takes into account interest rates as in [GHS21]. Lastly, as the accuracy of our model depends on estimations of the underlying PCN parameters, for instance, the average total number of transactions and the average number of transactions sent out by each user, developing more accurate methods for estimating these parameters may be helpful.

CHAPTER 7

Conclusion

Now cometh the age of the stars.

Ranni, Elden Ring

In this thesis, we studied algorithms for routing, rebalancing, and optimisation in payment channel networks, with a focus on the privacy and efficiency aspects.

To recap the motivation of the thesis, one of the main challenges that existing route discovery and rebalancing protocols face is that they either focus on privacy or efficiency but not on both. As such, our first goal was to advance the study of routing and rebalancing algorithms in payment channel networks by proposing methods which are both private and efficient. We also observed that in the optimisation frontier, existing models for actions, costs, and utility are either too simplistic or rely on strong assumptions that are unrealistic. Thus, our second goal is to develop richer and more realistic models as well as optimisation algorithms in these settings that come with good guarantees.

We summarise the main problems and our solutions below:

- In Chapter 3, we studied the problem of efficient and private route discovery in PCNs. We introduced a notion of privacy that hides the route query from servers that receive the query. We presented a protocol that is private and efficient, based on the crucial observation that hub-labelling algorithms combine well with private information retrieval to boost asymptotic efficiency. We also developed a heuristic that leverages the topology of payment networks to discover hubs more efficiently. To conclude our findings, we validated all our theoretical results with

simulations on historical snapshots of the Lightning Network, and show that the evaluations imply that our method can be run frequently with little overhead, making it easy to use in practice.

- In Chapter 4, we studied the problem of private and efficient rebalancing on PCNs. We first outlined the shortcomings of previous rebalancing proposals, namely the focus on either efficiency or privacy but not both. To the best of our knowledge, we then presented a protocol that is both fully private and efficient, overcoming all the shortcomings of previous solutions. Our method formulates the rebalancing problem as a linear program and solves it using MPC to achieve a fully private, globally optimal solution.
- In Chapter 5, we studied the problem of optimising the utility of an existing user in the PCN. We look at the problem in both the offline and online setting. In both of these settings, we advanced existing models of utility by considering a richer model of actions and costs. In the offline setting, we presented a constant approximation algorithm that decides how much capacity to inject into a channel and which transactions to accept and reject, given an input transaction sequence. In the online setting, we present competitive algorithms in various settings. We also validated our theoretical results in the online setting with simulations on randomly generated transaction sequences. Our simulations show that the average performance of our algorithm is significantly lower than our theoretical bound, which implies that our algorithms can be deployed in practice.
- In Chapter 6, we studied the problem of optimising the utility of a new user that wants to join the PCN. We advanced prior models of utility in this setting by adopting more realistic models of transactions and costs. We showed that the objective function under our more complex model is difficult to optimise with good guarantees on the approximation ratio using existing optimisation algorithms. Nevertheless, we presented several meaningful ways to constrain the optimisation setting so that we can still develop good approximation algorithms in each setting. We conclude our findings by with an analysis of stable network properties and topologies.

The hope of this thesis is not only to advance the privacy and efficiency frontiers of PCN research, but also to spark interest in future research that continues the ideas and directions presented in this thesis. In particular, there are various immediate as well as broader research directions that form interesting and natural progressions of the ideas proposed in the thesis. We identify some of these directions as follows:

Privacy efficiency trade-off. One of the reasons why most prior work have focused on either privacy or efficiency exclusively is the existence of an inherent trade-off between

privacy and efficiency. For instance, a lot of private algorithms for rebalancing and routing either use MPC [APS+21, MMSKM17], or some noising mechanism [TWFO20, DTZG22] to hide channel balances, which makes achieving private solutions come at an additional cost to utility and scalability. The scalability of most fully private solutions naturally depends on the advances in the implementation of the underlying cryptographic primitives, e.g., the development of faster MPC libraries or specialised hardware. An important and natural question is whether we can develop private solutions that do not depend on sufficient advancements in MPC implementations in order to create scalable private solutions. A promising approach is to leverage the specific topological structure of PCNs (close to star-like with very few high-degree hub nodes), and develop optimisation algorithms that scale exponentially in the number of hub nodes instead of the input [TYA+22]. This restricts the computational complexity of the optimisation problem, and hence the amount underlying MPC computations, to scale in the number of hubs which can be assumed to be very small. Another approach is to combine specific topological properties of the network with other private algorithms that are efficient. Thus, our proposed routing solution in Chapter 3 can also be seen as a step in the right direction.

Optimisation of whole network. In Chapter 5 we restricted the scope of our optimisation problem to a single channel with both users cooperating to reduce the total cost of their shared channel. A natural extension of this problem is to expand the scope of the optimisation problem to include the entire network. Some important considerations in this setting would be to firstly ensure the atomicity of the decisions on transactions on an entire payment path – either a payment is accepted along all channels in the payment path, or rejected by all channels. Another important consideration is network creation, that is, how to create a network that minimises the total cost. As the length of payment paths clearly make the problem more complex in this setting, an interesting approach for future work is to restrict the analysis to low-diameter topological structures like the star.

Cross-PCN protocols. Thus far we have limited the focus of our study to protocols and problems over a single PCN. However, just as there are now several blockchains, and correspondingly several PCNs built on top of these blockchains, future PCN research would naturally have to include the design of private and efficient cross-PCN protocols. To the best of our knowledge, there has been only a single work so far that proposed how to create cross-PCN channels [JYS+23]. Thus, there is still a plethora of open questions to be answered, such as designing efficient cross-chain payment routing protocols, efficient cross-chain conflict resolution protocols, cross-chain transaction aggregation etc. Future work on these might perhaps draw ideas from the current protocols and algorithms for single PCNs.

Bibliography

[AAF+97] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[ABM+21] Zeta Avarikioti, Mahsa Bastankhah, Mohammad Ali Maddah-Ali, Krzysztof Pietrzak, Jakub Svoboda, and Michelle Yeo. Route discovery in private payment channel networks. *IACR Cryptol. ePrint Arch.*, page 1539, 2021.

[ABWW19] Georgia Avarikioti, Kenan Besic, Yuyi Wang, and Roger Wattenhofer. Online payment network design. *CoRR*, abs/1908.00432, 2019.

[ACK+21] Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Inverse-sybil attacks in automated contact tracing. In Kenneth G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 399–421. Springer, 2021.

[ACL16] Laurence Aitchison, Nicola Corradi, and Peter E. Latham. Zipf's law arises naturally when there are underlying, unobserved variables. In Olaf Sporns, editor, *PLoS computational biology*, 2016.

[ADF+11] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Vc-dimension and shortest path algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 690–699. Springer, 2011.

[ADGW11] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011.

[ADGW12] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Hierarchical hub labelings for shortest paths. In Leah Epstein and Paolo Ferragina, editors, *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

[AEE+14] Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On nash equilibria for a network creation game. *ACM Trans. Economics and Comput.*, 2(1):2:1–2:27, 2014.

[AEE+20] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Bitcoin-compatible virtual channels. Cryptology ePrint Archive, Report 2020/554, 2020. `https://eprint.iacr.org/2020/554`.

[AFGW10] Ittai Abraham, Amos Fiat, Andrew Goldberg, and Renato Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA10)*. Society for Industrial and Applied Mathematics, January 2010.

[AGBPBTP20] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Rational vs byzantine players in consensus-based blockchains. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 43–51, 2020.

[AGOT92] Ravindra Ahuja, Andrew Goldberg, James Orlin, and Robert Tarjan. Finding minimum-cost flows by double scaling. *Math. Program.*, 53:243–266, 02 1992.

[AHWW20] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the lightning: The game theory of payment channels. In *FC*, 2020.

[AKWZ21a] Zeta Avarikioti, Eleftherios Kokoris Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous incentive-compatible payment channels. In *FC*, 2021.

[AKWZ21b] Zeta Avarikioti, Eleftherios Kokoris-Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous incentive-compatible payment channels. In *Financial Cryptography and Data Security FC*, 2021.

[AL22] Zeta Avarikioti and Orfeas Stefanos Thyfronitis Litos. Suborn channels: Incentives against timelock bribes. In *Financial Cryptography and Data Security FC*, 2022.

[ALW20] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. In *FC*, 2020.

[Aly15] A. Aly. Network flow problems with secure multiparty computation, 2015.

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[APS+21] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. HIDE & SEEK: privacy-preserving rebalancing on payment channel networks. *CoRR*, abs/2110.08848, 2021.

[APS+22a] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2022.

[APS+22b] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In *Financial Cryptography and Data Security FC*, 2022.

[APS+22c] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. Hide and seek: Privacy-preserving rebalancing on payment channel networks. In *Proc. Financial Cryptography and Data Security (FC)*, 2022.

[ASW19] Georgia Avarikioti, Rolf Scheuner, and Roger Wattenhofer. Payment networks as creation games. In *DPM/CBT@ESORICS*, 2019.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[BCM+22] Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali Maddah-Ali, Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Online admission control and rebalancing in payment channel networks. *CoRR*, abs/2209.11936, 2022.

[BDG15] N. Borisov, G. Danezis, and I. Goldberg. Dp5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015:24 – 4, 2015.

[BEY05] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[BFL+21] Davide Bilò, Tobias Friedrich, Pascal Lenzner, Stefanie Lowski, and Anna Melnichenko. Selfish creation of social networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event*, pages 5185–5193. AAAI Press, 2021.

[BFM+07] H. Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In transit to constant time shortest-path queries in road networks. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007.

[BG00] Venkatesh Bala and Sanjeev Goyal. A noncooperative model of network formation. *Econometrica*, 68(5):1181–1229, 2000.

[BI01] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In *In Proceedings of 28th ICALP*, pages 912–926, 2001.

[BIKR02a] A. Beimel, Y. Ishai, E. Kushilevitz, and J. Raymond. Breaking the o(n/sup 1/(2k-1)/) barrier for information-theoretic private information retrieval. *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 261–270, 2002.

[BIKR02b] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the o(n1/(2k-1)) barrier for information-theoretic private information retrieval. In *43rd Symposium on Foundations of Computer Science (FOCS 2002)*, pages 261–270. IEEE Computer Society, 2002.

[BNT22] Alex Biryukov, Gleb Naumenko, and Sergei Tikhomirov. Analysis and probing of parallel channels in the lightning network. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 337–357. Springer, 2022.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 562–592. Springer, 2020.

[Cap13] Justin Cappos. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 386–394. Springer, 2013.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In *FC*, 2016.

[CDG+21] Suvradip Chakraborty, Stefan Dziembowski, Malgorzata Galazka, Tomasz Lizurej, Krzysztof Pietrzak, and Michelle Yeo. Trojan-resilience without cryptography. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 397–428. Springer, 2021.

[CdH10] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security – ESORICS 2010*, pages 134–150, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In Eli Biham, editor, *Advances*

in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer, 2003.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CKS04] Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 156–165, 2004.

[CLS21] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1):3:1–3:39, 2021.

[CP19] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. Whitepaper, 2019. `https://docs.chia.net/assets/files/Precursor-ChiaGreenPaper-82cb50060c575f3f71444a4b7430fb9d.pdf`.

[CXG+21] Lin Chen, Lei Xu, Zhimin Gao, Ahmed Sunny, Keshav Kasichainula, and Weidong Shi. A game theoretical analysis of non-linear blockchain system. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 2021.

[Dec] Christian Decker. Lightning network research; topology, datasets. `https://github.com/lnresearch/topology`. Accessed: 2020-10-01.

[DEFM19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 106–123. IEEE, 2019.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):1–15, 2016.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[DHMZ07] Erik D. Demaine, MohammadTaghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in network creation games. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 292–298, 2007.

[DHS14] Daniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: practical multi-server PIR. In Gail-Joon Ahn, Alina Oprea, and Reihaneh Safavi-Naini, editors, *Proc. 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014*, pages 45–56. ACM, 2014.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

[DSW16] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *ICDCN*, 2016.

[DTZG22] Maya Dotan, Saar Tochner, Aviv Zohar, and Yossi Gilad. Twilight: A differentially private payment channel network. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 555–570. USENIX Association, 2022.

[DW15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, 2015.

[EFM+11] Shayan Ehsani, MohammadAmin Fazli, Abbas Mehrabian, Sina Sadeghian Sadeghabad, MohammadAli Safari, Morteza Saghafian, and Saber ShokatFadaee. On a bounded budget network creation game. *CoRR*, abs/1111.0554, 2011.

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

[ERE20a] Oğuzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to profit from payments channels. In *FC*, 2020.

[ERE20b] Oğuzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to profit from payments channels. In *Financial Cryptography and Data Security FC*, page 284–303, 2020.

[FLM+03a] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H Papadimitriou, and Scott Shenker. On a network creation game. In *PODC*, 2003.

[FLM+03b] Alex Fabrikant, Ankur Luthra, Elitza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 347–351, 2003.

[FMV11] Uriel Feige, Vahab Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40:1133–1153, 01 2011.

[FNS21] MohammadAmin Fazli, Seyed Moeen Nehzati, and MohammadAmin Salarkia. Building stable off-chain payment networks. *CoRR*, abs/2107.03367, 2021.

[GH05] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: *A* search meets graph theory. In *Proc. Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, 2005.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, 2017.

[GHP00] M. Gerla, Xiaoyan Hong, and Guangyu Pei. Landmark routing for large ad hoc wireless networks. In *Globecom '00 - IEEE. Global Telecommunications Conference. Conference Record (Cat. No.00CH37137)*, volume 3, pages 1702–1706 vol.3, 2000.

[GHS21] Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Channels. *Michael J. Brennan Irish Finance Working Paper Series Research Paper No. 21-7*, 2021.

[Gita] Github. Lightning network daemon. `https://github.com/lightningnetwork/lnd`.

[Gitb] Github. Rebalance plugin. `https://github.com/lightningd/plugins/tree/master/rebalance`.

[GK00] Piyush Kumar Gupta and Panganamala Ramana Kumar. The capacity of wireless networks. *IEEE Trans. Inf. Theory*, 46:388–404, 2000.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.

[GMSR+19] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. *IACR Cryptol. ePrint Arch.*, 2019:360, 2019.

[GMSR+20] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020.

[GRKC15] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705, 2015.

[Gut04] Ronald J. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 100–111. SIAM, 2004.

[HL98] R.E. Hall and M. Lieberman. *Macroeconomics: Principles and Applications*. Accounting Principles S. South-Western College Pub., 1998.

[HNR+19] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquín García-Alfaro. On the difficulty of hiding the balance of lightning network channels. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *Proceedings of the 2019 ACM*

Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019, pages 602–612. ACM, 2019.

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[Jac05] Matthew O Jackson. A survey of network formation models: stability and efficiency. *Group formation in economics: Networks, clubs, and coalitions*, 664:11–49, 2005.

[JP] Thaddeus Dryja Joseph Poon. The bitcoin lightning network: Scalable off-chain instant payments.

[JYS+23] Xiaofeng Jia, Zhe Yu, Jun Shao, Rongxing Lu, Guiyi Wei, and Zhenguang Liu. Cross-chain virtual payment channels. *IEEE Transactions on Information Forensics and Security*, 18:3401–3413, 2023.

[KER21] Satwik Prabhu Kumble, Dick H. J. Epema, and Stefanie Roos. How lightning's routing diminishes its anonymity. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 13:1–13:10. ACM, 2021.

[KG17a] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *CCS*, 2017.

[KG17b] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 439–453. ACM, 2017.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[KL19] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. *Cryptology ePrint Archive*, 2019.

[KPW+21] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 268–284. IEEE, 2021.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, 2017.

[KYP+20] George Kappos, Haaroon Yousaf, Ania M. Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An empirical analysis of privacy in the lightning network. *CoRR*, abs/2003.12470, 2020.

[Lau06] Ulrich Lauther. An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem, Proc. of a DIMACS Workshop, 2006*, 2006.

[LBS+15] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *International conference on autonomous agents and multiagent systems*, pages 919–927, 2015.

[LMNS09] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In Michael Mitzenmacher, editor, *ACM Symposium on Theory of Computing, STOC*, 2009.

[LMZ20a] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. Secure balance planning of off-blockchain payment channel networks. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1728–1737, 2020.

[LMZ20b] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. Secure balance planning of off-blockchain payment channel networks. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pages 1728–1737. IEEE, 2020.

[LNZ+16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open

blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 17–30. ACM, 2016.

[LS15] Tamás Lukovszki and Stefan Schmid. Online admission control and embedding of service chains. In *International Colloquium on Structural Information and Communication Complexity*, pages 104–118. Springer, 2015.

[MBB+19a] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration outsourcing for state channels. In *AFT*, 2019.

[MBB+19b] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *FC*, 2019.

[MMKM17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[MMS+19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*, 2019.

[MMSK+17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[MMSKM17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *NDSS*, 2017.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

[NFSD20] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. In *Proc. 6th International Conference on Information Systems Security and Privacy (ICISSP)*, 2020.

[Orl96] James Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Math. Prog.*, 78:109–129, 01 1996.

[PD15] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2015.

[PN20a] Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*, pages 1–5. IEEE, 2020.

[PN20b] Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*, pages 1–5. IEEE, 2020.

[PSSY21a] Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. Lightpir: Privacy-preserving route discovery for payment channel networks. In Zheng Yan, Gareth Tyson, and Dimitrios Koutsonikolas, editors, *IFIP Networking Conference, IFIP Networking 2021, Espoo and Helsinki, Finland, June 21-24, 2021*, pages 1–9. IEEE, 2021.

[PSSY21b] Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. Lightpir: Privacy-preserving route discovery for payment channel networks. In *IFIP Networking*, 2021.

[PZS+16a] Pavel Prihodko, S. N. Zhigulin, Mykola Sahno, A B Ostrovskiy, and Olaoluwa Osuntokun. Flare : An approach to routing in lightning network white paper, 2016.

[PZS+16b] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper*, 2016.

[Rai17] Raiden Network. Raiden network. `https://raiden.network/`, 2017.

[RAKM21] Sophie Rain, Zeta Avarikioti, Laura Kovács, and Matteo Maffei. Towards a game-theoretic security analysis of off-chain protocols. *arXiv preprint arXiv:2109.07429*, 2021.

[RBS16] Stefanie Roos, Martin Beck, and Thorsten Strufe. Voute-virtual overlays using tree embeddings. *arXiv preprint arXiv:1601.06119*, 2016.

[RMKG18] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[RMSKG17] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[RMT19] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 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)*, pages 347–356. IEEE, 2019.

[RT85] Prabhakar Raghavan and Clark D Thompson. Provably good routing in graphs: regular arrays. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 79–87, 1985.

[SAPP15] Christoph Salge, Nihat Ay, Daniel Polani, and Mikhail Prokopenko. Zipf's law: Balancing signal usage cost and communication efficiency. In Neil R. Smalheiser, editor, *PLoS ONE*, 2015.

[Spi] Jeremy Spilman. Anti dos for tx replacement. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html`. Accessed: 2020-11-22.

[SSY23] Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Weighted packet selection for rechargeable links in cryptocurrency networks: Complexity and approximation. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 576–594. Springer, 2023.

[SVA+18] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. Routing cryptocurrency with the spider network. In *Proc. 17th ACM Workshop on Hot Topics in Networks*, pages 29–35, 2018.

[SVR+20a] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and

Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 777–796, 2020.

[SVR+20b] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2020.

[TM20] Somanath Tripathy and Susil Kumar Mohanty. Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In *International Conference on Financial Cryptography and Data Security*, pages 481–495. Springer, 2020.

[Tof09] Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*, volume 5628 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.

[TS20] Saar Tochner and Stefan Schmid. On search friction of route discovery in offchain networks. In *Proc. IEEE International Conference on Blockchain (Blockchain)*, 2020.

[TWFO20] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(2):29:1–29:39, 2020.

[TYA+22] Samarth Tiwari, Michelle Yeo, Zeta Avarikioti, Iosif Salem, Krzysztof Pietrzak, and Stefan Schmid. Wiser: Increasing throughput in payment channel networks with transaction aggregation. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 217–231. ACM, 2022.

[TZS20a] Saar Tochner, Aviv Zohar, and Stefan Schmid. Route hijacking and dos in off-chain networks. In *AFT*, 2020.

[TZS20b] Saar Tochner, Aviv Zohar, and Stefan Schmid. Route hijacking and dos in off-chain networks. In *Proc. ACM Conference on Advances in Financial Technologies (AFT)*, 2020.

[vER20] Yuup van Engelshoven and Stefanie Roos. The merchant: Avoiding payment channel depletion through incentives. *CoRR*, abs/2012.10280, 2020.

[vER21a] Yuup van Engelshoven and Stefanie Roos. The merchant: Avoiding payment channel depletion through incentives. *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 59–68, 2021.

[vER21b] Yuup van Engelshoven and Stefanie Roos. The merchant: Avoiding payment channel depletion through incentives. In *IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS*, 2021.

[Wee19] Bitcointech Weekly. Outsourcing route computation with trampoline payments. In *https://bitcointechweekly.com/front/outsourcing-route-computation-with-trampoline-payments/*, 2019.

[WG16] Karl Wüst and Arthur Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.

[WGL+22] Xiaojian Wang, Huayue Gu, Zhouyu Li, Fangtong Zhou, Ruozhou Yu, and Dejun Yang. Why riding the lightning? equilibrium analysis for payment hub pricing. In *ICC*, pages 5409–5414. IEEE, 2022.

[Wil19] David P. Williamson. Bridging continuous and discrete optimization. `https://people.orie.cornell.edu/dpw/orie6334/lecture23.pdf`, 2019.

[WXJW19] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 370–381, 2019.

[WZPM16] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. Privacy-preserving shortest path computation. *CoRR*, abs/1601.02281, 2016.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1), February 2008.

[YXK+18] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks. In *2018 27th International Conference*

on Computer Communication and Networks (ICCCN), pages 1–9. IEEE, 2018.

[ZHS07] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Two-servers PIR based DNS query scheme with privacy-preserving. In *The 2007 International Conference on Intelligent Pervasive Computing, IPC 2007*, pages 299–302. IEEE Computer Society, 2007.

[Zip49] George Kingsley Zipf. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.

APPENDIX A

Supplementary material for Chapter 5

A.1 Computing the cost of OFF using dynamic programming

In this section, we describe a dynamic programming algorithm ON that solves the main problem. We assume that the size of transactions is integral (moreover the sum of transactions should be small).

Let $\textsc{Cost}_i(F_\ell, F_r)$ be the minimum cost for rejecting and off-chain rebalancing in processing sequence X_i that ends with $b(\ell) = F_\ell$ and $b(r) = F_r$ (For values of F_ℓ and F_r smaller that 0, we define it to be ∞). $\textsc{Cost}_i(F_\ell, F_r)$ can be derived from $\textsc{Cost}_{i-1}$ given the decision on the ith transaction.

Let us assume wlog that the ith transaction is from ℓ to r. OFF has three choices when encountering the transaction x_i. The first option is to reject x_i, then the cost is $A_1 = \textsc{Cost}_{i-1}(F_\ell, F_r) + Rx_i + f_2$. The second option is to accept x_i which gives the cost $A_2 = \textsc{Cost}_{i-1}(F_\ell + x_i, F_r - x_i)$. The last option is to rebalance the channel off-chain before accepting the transaction x_i. Note that any off-chain rebalancing before rejecting or accepting (while having enough funds) can be postponed. This gives a cost of $A_3 = \min_{a \leq F_\ell + x_i} \textsc{Cost}_{i-1}(a, F_r + F_\ell - a) + C \cdot R(F_\ell + x_i - a) + Cf_2$. OFF chooses the best option, that means $\textsc{Cost}_i(F_\ell, F_r) = \min\{A_1, A_2, A_3\}$

Note that we handle right to left transactions in the same way.

Thus, ON computes $\textsc{Cost}_t$ for all valid pairs (F_ℓ, F_r) and the final cost is

$$\textsc{Cost}_{\mathrm{OFF}}(X_i) = \min_{F_\ell, F_r} \textsc{Cost}_t(F_\ell, F_r) + F_\ell + F_r + f_1$$

To bound the time complexity of ON, we observe some bounds for $S = F_\ell^* + F_r^*$, where F_ℓ^* and F_r^* are the values of F_ℓ and F_r achieving the minimal cost. Observe that $S \leq \sum_{i \leq t} x_i$, it is not worth to have more money than the sum of the trasactions. We can strenghten the inequality and instead of $\sum_{i \leq t} x_i$, we can compute the minimal amount needed to accept every transaction. The other option is to reject everything, so we know that $f_1 + S \leq \sum_{i \leq t} Rx_i + f_2$.

Now we can prove the theorem about the described algorithm ON.

Theorem A.1.1. ON *computes the optimal cost in time* $\mathcal{O}(tS^3)$*, where* S *is the bound on the maximal funds in the channel and* t *is the number of transactions.*

Proof. In the dynamic programming, we take into account all possible decisions ON can make, by this, correctness follows.

The algorithm ON tries all possible amounts between 1 and S and starting distributions. There are S^2 of them. While computing one value, it needs to look at at most S precomputed values. And it needs to do it at most t times. □

Using dynamic programming for calculating OFF has two advantages. First, we can easily recover the decisions of OFF. Secondly, dynamic programming provides us with the optimal solution for all subsequences of X_t. This is useful for implementing queries to the offline oracle in Algorithm 5.10.

APPENDIX B

Technical proofs for Chapter 6

B.1 Proof of Theorem 6.5.2

Theorem 6.5.2. *$K_{1,n}$ with $n \geq 2$ is a Nash Equilibrium when nodes transact with each other according to the Zipf distribution with parameter $s \geq 0$ whenever the following conditions hold:*

1. $\frac{a}{H_n^s} \leq l \cdot 2^s$,
2. $b \cdot \frac{i}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} + a\frac{H_{i+1}^s - 1}{H_n^s} \leq l \cdot (i-2)$ *(for $2 \leq i \leq n-1$)*

Proof. We will first show that the central node has no incentive to deviate from the star topology. Then, we analyse the leaves and list all possible deviations that could happen, then show that these deviations would lead to lower expected utility under the conditions stated above. Let us call the star topology the default strategy.

Consider the central node of the star. The possible deviations that can occur are either to add additional channels between the central node and some other nodes, or to delete edges to disconnect the central node from other nodes. Since the central node is connected to all other nodes, adding an additional channel to any node just increases the channel creation cost and thus decreases the expected utility for the central node. Removing a single edge disconnects the central node from some user and thus leads to infinite cost. Altogether, since any change in strategy results in a lower expected utility, the central node has no incentive to switch to any different strategy.

Now consider a leaf node u. Apart from sticking to the default strategy, the only possible deviations for a leaf node are:

1. connecting to all other $n-1$ leaves
2. connecting to all other $n-1$ leaves and removing the connection to the central node
3. connecting to 1 other leaf
4. connecting to $2 \leq i \leq n-2$ leaves
5. connecting to $2 \leq i \leq n-2$ leaves and removing the connection to the central node

Expected utility under default strategy. We now compute the expected utility of a leaf node u under the default strategy, as well as under each of these deviating strategies. Under the default strategy, the expected revenue of u is 0 as u is a leaf and so does not route transactions. Since the rank factor of the central node is 1, and each leaf node has rank factor $\frac{H_n^s-1}{n-1}$, the expected fees of u is $\mathbb{E}_u^{fees} = a(n-1)\frac{\frac{H_n^s-1}{n-1}}{H_n^s} = a \cdot \frac{H_n^s-1}{H_n^s}$. Finally, the cost of creating 1 channel is l. Altogether, the expected utility of u under the default strategy is $-a \cdot \frac{H_n^s-1}{H_n^s} - l$.

Expected utility under deviating strategy 1. Under deviating strategy 1, i.e., connecting to all the other $n-1$ leaves, the expected fees of u is $\mathbb{E}_u^{fees} = 0$ since u is connected to all nodes. The cost of creating these channels is nl. Under this strategy, the rank factor of u and the central node will both be $\frac{1+1/2^s}{2}$, and the rank factor of all other nodes will be $\frac{H_n^s-1-1/2^s}{n-2}$. Apart from the central node and u, there are $\binom{n-1}{2}$ pairs of nodes that, when transacting with each other, could route their transactions through u or the central node, with equal probability of $\frac{1}{2}$ for each. Thus, the expected revenue of u under this strategy would be $\mathbb{E}_u^{rev} = 2b \cdot \frac{1}{2}\binom{n-1}{2}\frac{\frac{H_n^s-1-1/2^s}{n-2}}{H_n^s} = b \cdot \frac{n-1}{2} \cdot \frac{H_n^s-1-1/2^s}{H_n^s}$. Altogether, the expected utility of u under this strategy is $b \cdot \frac{n-1}{2} \cdot \frac{H_n^s-1-1/2^s}{H_n^s} - ln$.

Expected utility under deviating strategy 2. Under deviating strategy 2, i.e., connecting to all other leaves and removing the connection to the central node, the expected fees incurred by u would be $\mathbb{E}_u^{fees} = \frac{a}{H_n^s}$ as the only way u will incur any fees is to transact with the central node which has a rank factor of 1 from the perspective of u. The cost of channel creation is $l \cdot (n-1)$. The rank factor of both u and the central node is still $\frac{1+1/2^s}{2}$, and the rank factor for all other nodes is also still $\frac{H_n-1-1/2^s}{n-2}$. Thus, the expected revenue is $\mathbb{E}_u^{rev} = 2b \cdot \frac{1}{2}\binom{n-1}{2}\frac{\frac{H_n^s-1-1/2^s}{n-2}}{H_n^s} = b \cdot \frac{n-1}{2} \cdot \frac{H_n^s-1-1/2^s}{H_n^s}$. The expected utility is thus $b \cdot \frac{n-1}{2} \cdot \frac{H_n^s-1-1/2^s}{H_n^s} - \frac{a}{H_n^s} - l(n-1)$.

Expected utility under deviating strategy 3. Under deviating strategy 3, i.e., adding only 1 connection to another leaf, the expected revenue of u is still going to be 0 as the extra connection does not reduce shortest paths that go through u and thus does not give u any benefit in terms of added fees. The cost of channel creation in this case is $2l$. The central node still maintains its rank factor of 1, and the leaf that u connects to will have rank factor $\frac{1}{2^s}$. All other $n-2$ leaves would have rank factor $\frac{H_n^s-1-\frac{1}{2^s}}{n-2}$. Thus, $\mathbb{E}_u^{fees} = a(n-2)\cdot\frac{\frac{H_n^s-1-\frac{1}{2^s}}{n-2}}{H_n^s} = a\cdot\frac{H_n^s-1-\frac{1}{2^s}}{H_n^s}$. Altogether, the expected utility of u under this strategy is $-a\cdot\frac{H_n^s-1-\frac{1}{2^s}}{H_n^s} - 2l$.

Expected utility under deviating strategy 4. Under deviating strategy 4, i.e., adding connections to $2 \leq i \leq n-2$ other leaves, the cost of channel creation is $l(i+1)$. From the perspective of u, the central node has rank factor 1, the leaves that u connects to have rank factor $\frac{H_{i+1}^s-1}{i}$, and all the other nodes have rank factor $\frac{H_n^s-H_{i+1}^s}{n-i-1}$. Thus, $\mathbb{E}_u^{fees} = a(n-i-1)\frac{\frac{H_n^s-H_{i+1}^s}{n-i-1}}{H_n^s} = a\cdot\frac{H_n^s-H_{i+1}^s}{H_n^s}$. From the perspective of a leaf that u connects to, the central node has rank factor 1, u has rank factor $\frac{1}{2}$, and the other $i-1$ leaves that u connects to have rank factor $\frac{H_{i+1}^s-1-1/2}{i-1}$. Thus, the expected revenue of u is $\mathbb{E}_u^{rev} = 2b\cdot\frac{1}{2}\binom{i}{2}\frac{\frac{H_{i+1}^s-1-1/2}{i-1}}{H_n^s} = b\cdot\frac{i}{2}\cdot\frac{H_{i+1}^s-1-1/2^s}{H_n^s}$. Altogether, the expected utility of u under this strategy is $b\cdot\frac{i}{2}\cdot\frac{H_{i+1}^s-1-1/2^s}{H_n^s} - a\cdot\frac{H_n^s-H_{i+1}^s}{H_n^s} - l(i+1)$.

Expected utility under deviating strategy 5. Finally, we analyse the utility of u under deviating strategy 5, which is to add connections to $2 \leq i \leq n-2$ other leaves and remove the connection to the central node. Under this strategy, the cost of channel creation is li. From the perspective of u, the central node has rank factor 1, the leaves that u connects to have rank factor $\frac{H_{i+1}^s-1}{i}$, and all other nodes have rank factor $\frac{H_n^s-H_{i+1}^s}{n-i-1}$. Thus, the expected fees of u is $\mathbb{E}_u^{fees} = a\cdot[(n-i-1)\frac{\frac{H_n^s-H_{i+1}^s}{n-i-1}}{H_n^s} + \frac{1}{H_n^s}] = a\cdot\frac{H_n^s-H_{i+1}^s+1}{H_n^s}$. From the perspective of a leaf that u connects to, the central node has rank factor 1, u has rank factor $\frac{1}{2}$, and all $i-1$ other leaves that u is connected to have rank factor $\frac{H_{i+1}-1-1/2}{i-1}$. Thus, $\mathbb{E}_u^{rev} = 2b\cdot\frac{1}{2}\binom{i}{2}\frac{\frac{H_{i+1}^s-1-1/2^s}{i-1}}{H_{i+1}^s} = b\cdot\frac{i}{2}\cdot\frac{H_{i+1}^s-1-1/2^s}{H_n}$. Altogether, the expected utility of u under this strategy is $b\cdot\frac{i}{2}\cdot\frac{H_{i+1}^s-1-1/2^s}{H_n} - a\cdot\frac{H_n^s-H_{i+1}^s+1}{H_n^s} - li$.

Comparison of strategies. Now we compare the expected utilities of each deviating strategy, and show that under the conditions stated in the theorem claim, the expected utility under the default strategy is the highest. We first compare the expected utility under default strategy and deviating strategy 1. For the default strategy to be a Nash

Equilibrium, we need

$$-a \cdot \frac{H_n^s - 1}{H_n^s} - l \geq b \cdot \frac{n-1}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} - ln$$
$$\iff$$
$$a \cdot \frac{H_n^s - 1}{H_n^s} + b \cdot \frac{n-1}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} \leq l(n-1)$$

The inequality holds due to condition 2 in the theorem claim.

We now compare the expected utility under default strategy and deviating strategy 2. For the default strategy to be a Nash Equilibrium, we need

$$-a \cdot \frac{H_n^s - 1}{H_n^s} - l \geq b \cdot \frac{n-1}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} - l(n-1) - \frac{a}{H_n^s}$$
$$\iff$$
$$a \cdot \frac{H_n^s - 2}{H_n^s} + b \cdot \frac{n-1}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} \leq l \cdot (n-2)$$

The inequality holds due to condition 2 in the theorem claim.

We now compare the expected utility under default strategy and deviating strategy 3. For the default strategy to be a Nash Equilibrium, we need

$$-a \cdot \frac{H_n^s - 1}{H_n^s} - l \geq -a \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} - 2l$$
$$\iff$$
$$\frac{a}{H_n^s} \leq 2^s \cdot l$$

The inequality holds due to condition 1 in the theorem claim.

We now compare the expected utility under default strategy and deviating strategy 4. For the default strategy to be a Nash Equilibrium, we need

$$-a \cdot \frac{H_n^s - 1}{H_n^s} - l \geq b \cdot \frac{i}{2} \cdot \frac{H_{i+1}^s - 1 - 1/2^s}{H_n^s} - \frac{a \cdot (H_n^s - H_{i+1}^s)}{H_n^s} - l(i+1)$$
$$\iff$$
$$b \cdot \frac{i}{2} \cdot \frac{H_{i+1}^s - 1 - 1/2^s}{H_n^s} + a \cdot \frac{H_{i+1}^s - 1}{H_n^s} \leq li$$

The inequality holds due to condition 2 in the theorem claim.

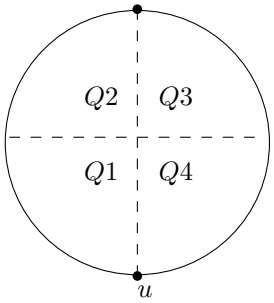


Figure B.1: Circle topology with vertices divided into quadrants

Finally, we compare the expected utility under default strategy and deviating strategy 4. For the default strategy to be a Nash Equilibrium, we need

$$-a \cdot \frac{H_n^s - 1}{H_n^s} - l \geq b \cdot \frac{i}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} - a \cdot \frac{1 + H_n^s - H_{i+1}^s}{H_n^s} - li$$

$$\iff$$

$$b \cdot \frac{i}{2} \cdot \frac{H_n^s - 1 - 1/2^s}{H_n^s} + a\frac{H_{i+1}^s - 2}{H_n} \leq l(i-1)$$

The inequality holds due to condition 2 in the theorem claim. □

B.2 Proof of Theorem 6.5.3

Theorem 6.5.3. *The Circle graph is not a Nash Equilibrium for $s \geq 0$ and for large n.*

Proof. Suppose we have a circle graph with $n+1$ nodes. Let u be an arbitrary vertex in the circle graph. Let us call the circle topology the default strategy.

Under the default strategy, u is connected to 2 other vertices, thus the cost of channel creation for u is $2l$. From the perspective of any node in the graph, all nodes under the default strategy have the same degree, and thus the same rank factor of H_n^s/n. Also, u will only gain revenue from forwarding transactions only if u is in the shortest path between the sender and receiver nodes. Figure B.1 shows how the vertices in the circle can be divided into quadrants $Q1, Q2, Q3, Q4$.

There are $\binom{n}{2}$ total pairs of vertices (excluding u). Out of these vertices, we need to exclude $\binom{n/2}{2}$ pairs of vertices (the vertices that lie in $Q2$ and $Q3$ in Figure B.1) when computing the expected revenue of u as these pairs will never route transactions through u. For every vertex in quadrants $Q1$ and $Q4$, we also need to exclude the $\frac{n}{2}$

other vertices that they transact with that u is not on the shortest path and so will never be routed through u. Thus, the expected revenue of u under the default strategy is $\mathbb{E}_u^{rev} = 2b \cdot \frac{\frac{H_n^s}{n}}{H_n^s} \cdot (\binom{n}{2} - \binom{n/2}{2} - (2 \cdot \frac{n}{4} \cdot \frac{n}{2})) \approx \frac{b}{n} \cdot \frac{n^2}{4} = \frac{bn}{4}$. From the perspective of u, there are 2 nodes of distance 0 to u (its direct neighbours), 2 nodes of distance 1 etc. Finally, depending on the parity of n, there are at most 2 nodes of distance $\lfloor \frac{n}{2} \rfloor$. Thus, $\mathbb{E}_u^{fees} = a \cdot \frac{\frac{H_n^s}{n}}{H_n^s} \cdot 2 \cdot (1 + 2 + \ldots + \frac{n}{2}) \approx \frac{a}{n} \cdot \frac{n^2}{4} = \frac{an}{4}$.

There is, however, a strictly better strategy for u, given that all other nodes stick to the default strategy. This is the strategy that deletes a connection to one of the neighbour nodes and creates a new connection to the node directly opposite to it (i.e., the node with distance $\lfloor \frac{n}{2} \rfloor$ from u, picking one at random if there are 2 of them). Under this new strategy, the cost of channel creation remains $2l$. From the perspective of the node opposite u, all nodes have a rank factor of $\frac{H_n^s}{n}$. For all other nodes apart from the node opposite of u and u itself, the node opposite u has rank factor 1 and all other nodes have rank factor $\frac{H_n^s - 1}{n-1}$. To compute the expected revenue of u under this new strategy, we use the smallest rank factor of $\frac{H_n^s - 1}{n-1}$ as this lower bounds the expected revenue.

Now we note that all vertices in $Q1$ will use u to route to vertices in $Q3$ and $Q4$. The vertices in $Q2$ will not use u to route to $Q3$ and $Q4$, but half of the vertices in $Q2$ will use u to route to half of the vertices in $Q1$. The vertices in $Q3$ will not use u to route to the vertices in $Q4$ and $Q2$ but will use u for vertices in $Q1$. Finally, the vertices in $Q4$ will only use u to route to vertices in $Q1$. Thus, $\mathbb{E}_u^{rev} \geq 2b \cdot \frac{\frac{H_n^s - 1}{n-1}}{H_n^s} \cdot (\frac{n}{4} \cdot \frac{n}{2} + \frac{n}{8} \cdot \frac{n}{8} + \frac{n}{4} \cdot \frac{n}{4} + \frac{n}{4} \cdot \frac{n}{4}) \approx bn(\frac{17}{32})$. From the perspective of u, the opposite node has rank factor 1 and all other nodes have rank factors of $\frac{H_n^s - 1}{n-1}$. Thus, $\mathbb{E}_u^{fees} = a \cdot \frac{\frac{H_n^s - 1}{n-1}}{H_n^s} (\frac{3\frac{n}{4}(\frac{n}{4}-1)}{2} + \frac{(\frac{n}{2}(\frac{n}{2}-1))}{2} - \frac{(\frac{n}{4}(\frac{n}{4}-1))}{2}) \leq \frac{a}{n} \frac{3}{16} n^2 = \frac{3an}{16}$. Since the expected fees under this new strategy is smaller than the expected fees under the default strategy, and the expected revenue under the new strategy is larger than the expected revenue under the default strategy, switching to the new strategy gives u larger expected utility. $\square$