Diss. ETH No. 27442

A Theoretical Treatment of Blockchain Scaling Protocols

A dissertation submitted to attain the degree of

Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by

Georgia Avarikioti
M.Sc., National and Kapidostrian University of Athens

born on 14 July 1988

citizen of Greece

accepted on the recommendation of

Prof. Dr. R. Wattenhofer, examiner
Prof. Dr. M. Maffei, Prof. Dr. A. Miller, co-examiners

2021

First Edition 2021

Series in Distributed Computing, Volume 35
TIK-Schriftenreihe-Nr. 187

In loving memory of my dearest friend Marisa,
keeping the promise we once made to change this unjust world.

Rather than love, than money, than fame, give me truth.
- Henry David Thoreau

Acknowledgements

First and foremost, I want to thank my PhD advisor, Prof. Roger Wattenhofer, for this beautiful journey. Roger was more than an advisor, a mentor and a friend, through these years. Our meetings were always a time of joy and clarity, be it in scientific matters, or simply discussions about society and life. Apart from the academic guidance, Roger instilled in me confidence and trust in myself, which I was severely lacking coming from a different background (especially when he was losing in Tichu!). In my 3,5 years in the group I never felt mistreated or that my opinion mattered less; quite the opposite, Roger is a rare personality that actually listens to arguments, which is extremely gratifying and enjoyable when working with someone. And in my time of need, Roger was always there, supporting and championing me, even when he had no reason to do so. It is with deep sorrow that this journey came to an end and I leave his group; I enjoyed every moment working with Roger (even writing the CoTi script!).

Second, I am particularly grateful to Lefteris. He has been my second mentor, the best collaborator, a loving friend, and an academic partner. Lefteris has supported me and encouraged me, both professionally and personally. He has guided me in my research, my career, my future, and my personal life. I could not have wished for a better "partner in crime". I can only express my deepest gratitude for his patience, friendship, support, and endorsement.

The completion of my PhD would not have been the same without Chrysa, Dionysis, Alexei, Orfeas, Tejaswi, Kobi, Roland, and Yuyi. I thank you all from the depth of my heart for the fruitful discussions and productive feedback on research (and more). A special thanks to Alexei for our memorable and adventurous work trips around the world – these fond memories of almost losing the plane on our way to Miami, the police picking up our car in San Francisco, the fever and the urchin in the Caribbean, the karaoke, the night dance, the drag show, and so many others, fill my soul with joy.

I cannot express enough thanks to my mother Vaso, and my sister Anastasia, as well as my extended family, Victor, Thanasis, and Aliki, for tolerating me all these years. They were there for me always, showing their unconditional love by listening to me whine for everything, being happy with my successes and holding me in my failures; and always always believing in me and encouraging me to "fly higher". But most importantly I am forever beholden to my sister for her blind loyalty and most notably for listening to every single talk I ever gave throughout my academic studies and waking up to see my live presentation in Tokyo at 5am. I could never hope for a better person to support me through this journey and the rest of my life.

Moreover, my warmest thanks to Vaggelis and Vaggelio, for showing me what family is really about: checking up on me every single day in the beginning of my journey, sharing their thoughts and new memories, so I would never miss home.

I would further like to sincerely thank my friends Athina, Haris and Maria for their mental support and academic guidance. As close friends with a strong research drive and academic careers, they inspired and influenced me, paving the path I follow today. Especially, Athina, or Tichu partner, for being my number one supporter, my academic "mother"; Haris for the peace of mind every Tuesday at Manuel's, and for his swimming-dance moves every Saturday at Paddy's; Maria for being by my side to help me rise after my deepest fall.

A heartfelt thanks to my friends Myrto and Giannis for bearing with me the stress and intensity of conducting a PhD, for the numerous calls regarding every matter possible during the last years. My deepest thanks to my everlasting friends from all around the world, Martha, Zeta, and Dimitra, that were there for me both in my sunniest and darkest hours, as well as my new friends, Sahar, Zoi, Mira, Chrysa, Kostas, Simos, Angeliki, and Jan, that made me feel like home in a foreign country. Furthermore, a big thanks to Roland, Lukas, Henri, and Beni for finally making the workplace a place where friendship can nurture. I hope all the best to all of you and may our paths cross again.

Lastly, I want to thank my friends, Yiannis, Dimitra, Michalis, Christos, and Mania, as well as my previous advisors, Prof. Stathis Zachos, Prof. Aris Pagourtzis, Prof. Dimitris Fotakis, and Prof. Ioannis Emiris, without whom I would have never conducted this PhD. They encouraged me, championed me, and believed in me when there was no evidence to do so. It is because of them I had the courage to follow this path. I will be forever grateful, and carry with me always the love and loyalty they showed me. I only hope one day I will get the chance to do the same.

Abstract

Scaling decentralized blockchains has been in the spotlight of the blockchain research community due to the immediate consequences on the widespread adoption of cryptocurrencies. In this thesis, we examine different scaling solutions of blockchain protocols mainly from a theoretical perspective.

We first present a *formalization of sharding protocols*, the most promising on-chain scaling solution. Our goal is to provide formal "common grounds" for systematically evaluating the security and efficiency of sharding systems. To that end, we define the necessary properties sharding protocols should satisfy. To demonstrate the power of our framework, we evaluate the most prominent sharding systems found in literature, and further provide bounds and limitations of what sharding protocols can achieve in general. We conclude by identifying the critical components of sharding.

We then focus on off-chain scaling solutions, and in particular, *payment channels*. We highlight and address *security* concerns on the fundamental construction of payment channels. Specifically, previous payment channel designs demand participants to be *online monitoring the blockchain*, while the blockchain should be *live* and the network *synchronous*. We alleviate the first assumption for Bitcoin-compatible constructions by presenting Cerberus channels. Cerberus channels enable participants to go securely offline for an extended period of time as monitoring the blockchain is outsourced to third-parties that are financially incentivized to faithfully follow the protocol. Assuming smart contracts we later present Brick, the first incentive-compatible asynchronous payment channel construction, effectively eliminating both security assumptions. Brick remains secure with offline participants under network asynchrony, and concurrently provides correct incentives.

Finally, we study the *creation of payment channel networks* under the lens of theory. On the one hand, we investigate the design of capital-efficient payment channel networks assuming a central coordinator. For this purpose, we introduce an *algorithmic framework* and present numerous results, either efficient (approximation) algorithms or hardness results. On the other hand, we model payment channel networks as *network creation games*, where participants act selfishly. We analyze prominent topologies for Nash equilibria, determine the social optima, and bound the price of anarchy when possible. Our objective is to determine the optimal topologies for payment channel networks, both under central coordination and when the network is decentralized.

Zusammenfassung

Die Skalierbarkeit dezentraler Blockchains steht aufgrund der unmittelbaren Auswirkungen für die voranschreitende Einführung von Kryptowährungen im Fokus der Blockchain-Forschungsgemeinschaft. In dieser Arbeit untersuchen wir verschiedene Skalierungslösungen von Blockchain-Protokollen, überwiegend von einem theoretischen Blickwinkel ausgehend.

Zunächst präsentieren wir eine Formalisierung von Sharding-Protokollen, der bislang vielversprechendsten "on-chain" Skalierungslösung. Unser Ziel war es, eine formale Grundlage für die systematische Bewertung der Sicherheit und Effizienz von Sharding-Systemen zu schaffen. Zu diesem Zweck definieren wir die erforderlichen Eigenschaften, welche Sharding-Protokolle erfüllen sollten. Um die Leistungsfähigkeit unseres Frameworks zu demonstrieren, evaluieren wir die bekanntesten Sharding-Systeme der Literatur. Darüber hinaus präsentieren wir Grenzen und Einschränkungen von Sharding Protokollen im Allgemeinen. Wir schliessen diesen Abschnitt mit der Identifikation der kritischen Komponenten des Sharding.

Anschliessend konzentrieren wir uns auf "off-chain" Skalierungslösungen, insbesondere auf Zahlungskanäle. Wir weisen auf Sicherheits-Bedenken hinsichtlich des grundlegenden Aufbaus von Zahlungskanälen hin und präsentieren Lösungen für diese. Bisherige Designs für sichere Zahlungskanäle setzen voraus, dass die Teilnehmer die Blockchain jederzeit online überwachen können, während die Blockchain "live" und das Blockchain-Netzwerk synchron sein muss. Mit Cerberus-Kanälen präsentieren wir eine Lösung, um diese erste Annahme für Bitcoin-kompatible Konstruktionen zu umgehen. Cerberus-Kanäle erlauben es den Teilnehmern über einen längeren Zeitraum sicher offline zu gehen, da die Überwachung der Blockchain an Dritte ausgelagert wird, welche einen finanziellen Anreiz zur präzisen Einhaltung des Protokolls haben. Unter der Annahme sogenannter "smart contracts" stellen wir später Brick vor, die erste anreizkompatible, asynchrone Zahlungskanalkonstruktion, bei der beide Sicherheitsannahmen effektiv eliminiert werden können. Brick bleibt für Offline-Teilnehmer auch in asynchronem Netzwerk sicher und bietet gleichzeitig korrekte Anreize.

Schliesslich untersuchen wir das Erstellen von Zahlungskanal-Netzwerken aus theoretischer Sicht. Einerseits untersuchen wir das Design kapitaleffizienter Zahlungskanal-Netze unter der Annahme eines zentralen Koordinators. Zu diesem Zweck führen wir ein algorithmisches Framework ein und präsentieren zahlreiche effiziente (Approximations-) Algorithmen oder Schwierigkeits-Reduktionen. Andererseits modellieren wir Netzwerke von Zahlungskanälen spieltheoretisch mit Netzwerkerstellungs-Spielen, in denen die Teilnehmer egoistisch handeln.

Wir analysieren prominente Topologien auf Nash-Gleichgewichte, bestimmen die sozialen Optima und geben Schranken für den Preis der Anarchie an, wenn möglich. Unser Ziel ist es, die optimalen Netzwerkstrukturen für Zahlungskanal-Netzwerke sowohl unter zentraler Koordination, als auch bei dezentraler Netzwerknutzung, zu ermitteln.

Contents

Acknowledgements viii

Abstract xi

Part II Introduction and Background 1

1 Introduction 3

1.1 Motivation . . . 3

1.2 Contribution and Publications . . . 5

1.3 Organization and Structure . . . 7

2 Background 9

2.1 Bitcoin . . . 9

2.1.1 Bitcoin Architecture . . . 9

2.1.2 Smart Contracts . . . 11

2.1.3 UTXO Model . . . 12

2.2 What is a Blockchain? . . . 13

2.3 Consensus Variants . . . 14

2.4 The Scalability Challenge . . . 17

2.5 Sharding a Blockchain . . . 21

2.6 Payments Channels (Layer 2) . . . 22

2.6.1 Foundations . . . 22

2.6.2 Payment Channel Networks (PCN) . . . 24

2.6.3 Shortcomings . . . 25

2.7 Other Scaling Solutions . . . 28

Part III Sharding Blockchains **31**

3 Formalization of Sharding Protocols **33**

3.1 Introduction . . . 33
3.1.1 Motivation . . . 33
3.1.2 Contribution . . . 35
3.2 The Sharding Framework . . . 36
3.2.1 The Model . . . 36
3.2.2 Sharded Transaction Ledgers . . . 37
3.2.3 (Sharding) Blockchain Protocols . . . 41
3.3 Limitations of Sharding Protocols . . . 42
3.3.1 General Bounds . . . 43
3.3.2 Bounds under Uniform Shard Creation . . . 45
3.3.3 Bounds under Random Party to Shard Assignment . . . 47
3.4 The Sharding Crux . . . 50
3.4.1 Sharding Components . . . 50
3.4.2 Protocol Abstraction . . . 52
3.4.3 Analysis . . . 52
3.5 Evaluation of Existing Protocols . . . 54
3.5.1 Elastico . . . 54
3.5.2 Monoxide . . . 57
3.5.3 OmniLedger . . . 60
3.5.4 RapidChain . . . 67
3.5.5 Chainspace . . . 73
3.5.6 Comparison of Sharding Protocols . . . 74
3.6 Conclusion, Limitations, and Extensions . . . 76

Part IV Payment Channels **79**

4 Cerberus: Incentive-Compatible Channels for Bitcoin **81**

4.1 Introduction . . . 81
4.1.1 Motivation . . . 81
4.1.2 Contribution . . . 82

4.2 Protocol Overview . . . 82
4.2.1 System Model . . . 82
4.2.2 Cerberus Overview . . . 83
4.2.3 Payment Channel Properties . . . 84
4.3 Cerberus Design . . . 84
4.3.1 Phase: Open . . . 84
4.3.2 Phase: Update . . . 87
4.3.3 Phase: Close . . . 89
4.3.4 Phase: Abort . . . 90
4.4 Security Analysis . . . 91
4.4.1 Security . . . 91
4.4.2 Performance . . . 95
4.5 Conclusion, Limitations and Extensions . . . 96

5 Brick: Asynchronous Payment Channels 99
5.1 Introduction . . . 99
5.1.1 Motivation . . . 99
5.1.2 Contribution . . . 101
5.2 Protocol Overview . . . 101
5.2.1 System Model . . . 101
5.2.2 Brick Overview . . . 102
5.2.3 Brick+ Overview . . . 103
5.2.4 Reward Allocation & Collateral . . . 103
5.2.5 Protocol Goals . . . 104
5.3 Brick Design . . . 105
5.3.1 Architecture . . . 105
5.3.2 Incentivizing Honest Behavior . . . 108
5.4 Brick Analysis . . . 111
5.4.1 Security Analysis under one Honest Participant and t Honest wardens . . . 111
5.4.2 Incentivizing Rational Players . . . 113
5.5 Brick+ Design . . . 119
5.5.1 Architecture . . . 119

5.5.2 BRICK+ Security Analysis 121
5.6 Conclusion, Limitations and Extensions 123

Part V Payment Channel Networks 127

6 Algorithmic Design of PCNs 129
6.1 Introduction 129
6.1.1 Motivation 129
6.1.2 From Payment Channels to Network Design 129
6.1.3 Contribution 130
6.2 PCN Design with Fees 132
6.2.1 Model with Diverse Fees 132
6.2.2 A Linear Program for Trees 133
6.2.3 Dynamic Program for Paths 134
6.2.4 Payment Hub: a Near-Optimal Solution 137
6.3 Model with Fixed Fees 138
6.3.1 Graph Model 138
6.3.2 Problem Variants 138
6.3.3 Adversarial Model 140
6.3.4 Assumptions 140
6.4 Offline PCN Design 140
6.4.1 Graph Topology 141
6.4.2 Single Channel 142
6.4.3 Channel Design for Maximum Profit 145
6.4.4 Channel Design with Graph Restrictions 149
6.5 Online Single Channel 150
6.5.1 Algorithms against Adaptive Adversaries 151
6.5.2 Algorithms with Advice 152
6.5.3 Randomized Algorithms against Oblivious Adversaries . . 153
6.5.4 Resource Augmentation 154
6.5.5 Minimizing the Number of Rejected Transactions 155
6.6 Online PCN Design for Maximum Profit 157

6.7 Conclusion, Limitations, and Extensions . . . 158

7 Game-Theoretic Analysis of PCNs 161
7.1 Introduction . . . 161
7.1.1 Motivation . . . 161
7.1.2 Contribution . . . 162
7.2 Preliminaries and Model . . . 162
7.2.1 PCN Creation Games . . . 162
7.2.2 Formal Model . . . 163
7.3 PCN Creation Game . . . 165
7.3.1 Social Optimum . . . 165
7.3.2 Nash Equilibria . . . 167
7.3.3 Price of Anarchy . . . 177
7.3.4 Price of Stability . . . 180
7.4 Conclusion, Limitations, and Extensions . . . 181

Part VI Related Work and Concluding Remarks 183

8 Related Work 185
8.1 Formalization of Sharding . . . 185
8.2 Payment Channel Constructions . . . 186
8.3 Algorithmic Design of PCNs . . . 188
8.4 PCN Creation Games . . . 190

9 Conclusion and Future Work 193
9.1 Summary and Insights . . . 193
9.2 Future Work . . . 196

Bibliography 201

Part II

Introduction and Background

Chapter 1

Introduction

1.1 Motivation

Bitcoin [1] and other cryptocurrencies [2–4] are electrifying the world. Thanks to a distributed data structure known as the blockchain, cryptocurrencies can execute financial transactions without a trusted central authority. However, every computer participating in a blockchain must exchange, store, and verify each and every transaction, and as such the transaction throughput of blockchains is embarrassingly low. The Bitcoin blockchain for instance does not process more than seven transactions per second [5].

With seven transactions per second, Bitcoin cannot rival established payment systems such as Visa, WeChatPay, or PayPal that process thousands. Consequently, various research groups have proposed numerous solutions that enable higher transaction throughput, and allow blockchain systems to *scale*. The scaling solutions for blockchains can be categorized in two groups: (a) On-chain solutions that aim to create highly efficient blockchain protocols, either by improving the performance of the consensus algorithm, or via sharding the blockchain; (b) Off-chain solutions that use the blockchain only as a fail-safe mechanism and move the transaction load offline, where the bottleneck is the network speed. On-chain solutions lead to the design of new promising blockchain systems, but they typically require stronger trust assumptions and they are not applicable to existing blockchain systems (without a hard fork). Off-chain (layer 2) solutions, on the other hand, are built on top of the consensus layer of the blockchain and operate independently. In this thesis, we explore both avenues.

Sharding. On the on-chain front, we focus on protocols that shard the blockchain. Sharding protocols partition the state (e. g., transactions) into multiple chains, called the shards, that operate in *parallel* with different subsets of participants. As a result, sharding systems reduce the transaction processing load of every

participant and increase the total processing capacity of the system, ideally proportionally to the number of participants.

Multiple sharding protocols have been proposed in the literature [6–9], all claiming advantages over the other. However, there is no *formal "common ground"* to evaluate and compare these systems, as there is no formal definition of what sharding is and achieves. In this thesis, we take up the challenge to fill in this gap and provide a formal framework for the security and efficiency analysis of sharding protocols.

Payment channels. Off-chain solutions, or so-called *payment channels*, enable transactions to be executed securely off-chain. Many such proposals exist [10–12], yet all of them follow the same basic principle: Instead of sending every transaction to the blockchain, transactions are only exchanged between the involved parties. If two parties expect to exchange multiple payments, they can establish a payment channel. The channel is set up with a blockchain funding transaction. Once the channel is available, the parties exchange all payments directly, by sending each other digitally signed payment messages. If a party tries to cheat, the counterparty can show the signed payment messages as a proof to the blockchain, using the original funding transaction as security.

Payment channels, although a revolutionary idea, have major drawbacks. The locked funds in the channel can only be safe if both parties are often *online monitoring the blockchain*. Furthermore, the blockchain must be *synchronous* and *live*, i. e., a transaction broadcast in the blockchain network must be included on-chain within a specific time period. If either of these assumptions is violated, a party might successfully steal funds from the channel. In this thesis, we tackle these problems by introducing two novel payment channel constructions, one applicable to Bitcoin and one operating with smart contracts.

Payment channel networks. Instead of establishing a payment channel to every other person and company in the world, thanks to a technique called Hash Time Locked Contracts (HTLCs) [11, 12], payments can also be sent atomically through a path of payment channels. More precisely, each payment channel is now an edge in a *payment channel network* (PCN), and payments can be routed along a path of payment channels in the payment network. Such a payment network is called the layer 2 of the blockchain, the blockchain itself being the layer 1.

Payment channel networks have many significant advantages over vanilla blockchains: With payment channels, the transaction throughput becomes unlimited, as each transaction is only seen by the nodes on the path between sender and receiver of a payment. This is like sending a packet in the internet instead of sending every packet to a central server. Solving the throughput problem will also drastically decrease transaction fees. In addition, payments will be instantaneous, as one does not have to wait multiple minutes before the blockchain verifies a

transaction. Payment networks also allow for more privacy as transactions are only seen by the parties involved. On the negative side, to set up a channel, the channel owner(s) must lock some capital, and pay the fee to publish the funding transaction of the channel on-chain. However, whenever a payment channel routes a transaction on behalf of other parties, the channel owner(s) can collect a transaction fee.

With the rising interest in decentralized payment networks, like Lighting [12], we study these trade-offs from a game-theoretic perspective. Our goal is to discover which network structures are stable assuming selfish participants. For this purpose, we model payment channel networks as network creation games.

Lastly, a major obstacle in the widespread adoption of payment channel networks is the vast amount of capital that needs to be locked in the channels. In this thesis, we want to understand how to optimally allocate this capital assuming a central coordinator. We further ask which is the optimal strategy (network topology, transactions' selection, fees) for a central coordinator to maximize their profit under capital restrictions. These questions spark an algorithmic study of payment channel networks.

1.2 Contribution and Publications

This thesis explores the scalability solutions for blockchain protocols, mainly from a theoretical perspective, by introducing formal models for sharding protocols and payment channel networks, as well as novel payment channel constructions to address the challenges mentioned above.

More specifically, the contributions of this thesis are the following:

- We define and analyze the properties sharded distributed ledgers should fulfill, extending the Bitcoin backbone protocol [13]. We provide the bounds and limits of secure and efficient sharding under our model, and we further highlight the necessary and critical components for designing a secure and efficient sharding system. To demonstrate the effectiveness of our protocol, we evaluate the most prominent sharding blockchain systems and either prove their correctness (OmniLedger [9], RapidChain [6]) or identify their failures (Elastico [7], Monoxide [8]) under our model.

- We introduce Cerberus channels, a novel Bitcoin-compatible payment channel construction that enables participants to employ third-parties, so-called watchtowers, that are financially incentivized to adhere to the protocol execution, i. e., both participate and act upon fraud. Thus, participants can go securely offline for an extended period of time. To that end, we define the desired properties for payment channel solutions and prove Cerberus channels are secure against collusion and bribing.

- We introduce Brick, the first payment channel construction that remains secure with offline participants under network asynchrony, and concurrently provides correct incentives. The core idea is to incorporate the conflict resolution process within the channel by introducing a rational committee of watchtowers. We present elaborate incentive-mechanisms to defend against collusion and bribing, and show that Brick is secure under a hybrid model of both rational and byzantine players (channel parties and watchtowers). We further provide a permissioned extension of Brick that adds auditability without conflicting with its privacy guarantees.

- We study the design of capital-efficient payment channel networks through the lens of a central coordinator, in the offline as well as the online setting (knowing the future transactions or not). We want to know how to compute an efficient payment network topology, how capital should be assigned to the individual edges, and how to decide which transactions to accept. We also consider the fees one might ask to route the transactions through the payment network. Towards this end, we present a flurry of interesting results, basic but generally applicable insights on the one hand, and hardness results and approximation algorithms on the other hand.

- Finally, we analyze payment channel networks in the context of network creation games. We study the topologies that emerge when players act selfishly and determine the parameter space in which they constitute a Nash equilibrium. We determine the social optima, bound the price of anarchy when possible, and briefly discuss the price of stability.

Most of the work presented in this thesis is based on the following co-authored publications:

- Georgia Avarikioti, Yuyi Wang, and Roger Wattenhofer. Algorithmic Channel Design.
  International Symposium on Algorithms and Computation, 2018.

- Georgia Avarikioti, Gerrit Janssen, Yuyi Wang, and Roger Wattenhofer. Payment Network Design with Fees.
  Data Privacy Management, Cryptocurrencies and Blockchain Technology, 2018.

- Georgia Avarikioti, Kenan Besic, Yuyi Wang, and Roger Wattenhofer. Online Payment Network Design.
  Data Privacy Management, Cryptocurrencies and Blockchain Technology, 2019.

- Georgia Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus Channels: Incentivizing Watchtowers for Bitcoin.

International Conference on Financial Cryptography and Data Security, 2020.

- Georgia Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the Lightning: The Game Theory of Payment Channels.
  International Conference on Financial Cryptography and Data Security, 2020.

- Georgia Avarikioti, Eleftherios Kokoris-Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous Incentive-Compatible Payment Channels.
  International Conference on Financial Cryptography and Data Security, 2021.

- Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and Scale: Formalization of Distributed Ledger Sharding Protocols.
  Under Submission.

1.3 Organization and Structure

The thesis is organized in five parts and nine chapters.

Chapter 2 introduces the necessary background to accommodate the reader, wrapping up Part I of the thesis. First, we discuss what a blockchain is and its connection to distributed computing. Then, we thoroughly explain the scalability problem of blockchains, its origins and limitations, and last we cover scaling solutions, focusing on sharding and payment channel networks.

We dive into the core of the thesis with Part II that consists of Chapter 3. In this chapter, we provide a general framework for the security and performance analysis of sharding protocols, explore the limitations of sharding under our framework, evaluate existing sharing systems to exhibit the utility of our model, and finally pinpoint the essential ingredients – the crux – of sharding.

Chapter 4 commences Part III of the dissertation that focuses on the fundamentals of payment channels. In this chapter, we introduce Cerberus channels, a novel Bitcoin-compatible payment channel construction that remains secure when parties go offline for an extended period of time by employing financially-incentivized watchtowers to act on their behalf.

We proceed with Chapter 5, where we introduce Brick, the first payment channel construction that remains secure under offline participants in full asynchrony with low latency. With this chapter we conclude Part III, as our two systems, Cerberus and Brick, successfully address the challenges discussed previously.

Part IV of this thesis, includes a theoretical treatment of payment channel networks, and it consists of Chapter 6 and Chapter 7.

First, in Chapter 6, we view payment channel networks through the eyes of a central coordinator, e. g. a company or a bank, in the context of algorithmic network design.

Then, in Chapter 7, we explore the payment channel network design problem under the umbrella of game-theory. In particular, we examine the network creation game the emerges with selfish participants, and evaluate which network topologies are stable under this model.

Finally, we conclude this thesis with Part V that surveys related work in Chapter 8, and presents concluding remarks and potential avenues for future research in Chapter 9.

Chapter 2

Background

2.1 Bitcoin

2.1.1 Bitcoin Architecture

With the inception of Bitcoin by Satoshi Nakamoto in 2008, a financial revolution began. Bitcoin is the first open distributed system that allows users to transact securely and efficiently. In other words, Bitcoin enables users to join and leave the system on-the-fly without the need for identities (open/permissionless), does not require a trusted third party (decentralized), and guarantees that all transactions will be permanently "written" in an accessible and verifiable transaction ledger via an efficient consensus mechanism that is resilient to participants that deviate arbitrarily from the protocol specification, called byzantine (secure). To achieve all these properties, Bitcoin employs a combination of tools, which we analyze below. From now on, we will use the terms users, nodes, participants, and parties interchangeably.

Addresses. Users may generate any number of private/public key pairs. A Bitcoin address is derived from a public key and is used as a pseudonymous identity. A private/public key pair can uniquely identify the owner of funds of an address. The addresses are used to send and receive bitcoins.

Transactions. Transactions consist of one or more inputs and one or more outputs. An output is a tuple consisting of an amount of bitcoins and a spending condition. Most commonly the spending condition requires a valid signature associated with the private key of an address. An output, on the other hand, is a tuple consisting of a reference to a previously created output and arguments to the spending condition (e. g., a digital signature created with the corresponding private key), proving that the creator of the transaction is allowed to spend the

referenced output. Therefore, a sender can transfer coins to a recipient (create a transaction) by appending to an output the digital signature on the reference of the previous transaction and the public key of the recipient. The recipient can consequently be convinced of the chain of ownership by verifying the signatures.

An output can have two states: spent and unspent. The unspent transaction outputs or UTXOs define the state of the system and are shared among the users. Transactions take UTXOs as inputs, destroy them, and create new UTXOs, the outputs. Thus, the system transitions to a new state when a transaction is executed. We discuss transactions and output conditions in more detail in Sections 2.1.2 and 2.1.3.

Proof-of-work & longest chain rule. Proof-of-work is a form of cryptographic zero-knowledge proof in which one party proves to others that a certain amount of computational effort has been expended for some purpose. One can efficiently confirm this expenditure of computational power.

Bitcoin employs proof-of-work to validate batches of transactions, or so-called *blocks*. The participants of the proof-of-work algorithm, called *miners*, create a block and then repeatedly hash it (by changing a nonce) along with their public key, and a reference to the previous valid block, until they find a hash that is lower than a predefined number, the target. Therefore, the valid blocks form a hash-chain, known as the *the blockchain*, that indicates the order in which transactions are confirmed. A block is valid only if the contained transactions are valid (i. e., the transactions spend only UTXOs), the hash is lower than the target, and the block is part of the longest blockchain. To create valid blocks, miners try to extend the longest chain.

In Bitcoin, proof-of-work has dual functionality. First, it allows for a secure permissionless setting; participants can create multiple keys but their "power" in the system is capped by the computational power they spend in it. Hence, the system is resilient to Sybil attacks. Second, proof-of-work along with the longest chain selection rule guarantee all transactions are totally ordered; if one attempts to double-spend a UTXO (transfer the same coin to two different addresses) only the first transaction in the ordering will be considered valid. Both these properties are essential to maintain a consistent view of the state among the distributed participants under byzantine adversaries.

Incentives. Miners are automatically rewarded with a fee when they append a block to the longest blockchain. In addition, the transactions in Bitcoin typically include a fee that is also awarded to the miner. Both these reward mechanisms incentivize miners to waste computational resources to operate and progress the system. As the system progresses, the amount of bitcoins awarded by the system for mining a block decreases, and eventually only transaction fees will be awarded.

Peer-to-peer network. The Bitcoin network is a peer-to-peer dynamic payment network. Its nodes are randomly connected, and they all perform the same operations. Transactions and blocks are propagated through the network, thus are shared among the participants. However, the time and order in which nodes receive the information might differ. Hence, each miner might have locally a different longest chain. The Bitcoin protocol only enforces that each miner tries to extend their longest chain. In case of a tie, another block will be eventually appended to one of the chains and the tie will be naturally broken.

Nevertheless, the quality of the network is vital for the correct operation of the Bitcoin protocol. Indeed, the propagation delay of the network is critical for the security of the proof-of-work algorithm [5, 13], as one can only start mining a new valid block if they have received the longest valid chain. As a result, the Bitcoin protocol is secure only if the propagation delay is bounded, i. e., the network is *synchronous* [14, 15]. It has been shown that in the proof-of-work setting, building a secure system that makes progress is only possible in synchronous networks [14, 15]. In Section 2.2, we will discuss such trade-offs in detail.

2.1.2 Smart Contracts

A contract is an agreement that can be enforced on the blockchain. Enforcing such a contract depends on the operations the programming language allows with respect to transaction outputs. Most recent cryptocurrencies, such as Ethereum [2, 16], support a Turing-complete language and thus can enforce arbitrary rules with a *smart contract*. However, Bitcoin (as well as other cryptocurrencies) has strict limitations on the scripting language and allows only specific operations. As a result, Bitcoin's contracts are simpler and with limited functionality. Next, we discuss some operations allowed in script with respect to transaction outputs.

Signatures. A signature is the most basic form of contract and is essentially a proof of ownership of a transaction output. We denote by σ_A the signature that corresponds to the public key A. (We omit the signed message, as it always is the transaction which contains the signature, with a placeholder in the location of the signature). Further, an m-of-n *multisignature* is a contract that demands at least m signatures which correspond to any m of the n predefined public keys. If m valid signatures are provided then the output is immediately spendable. We denote with $\sigma_{A,B}$ a 2-of-2 multisignature for the public keys A and B, meaning that the output of the transaction can only be spent with both the signatures of A and B.

Timelocks. Timelocks are another type of contract. When a transaction or a transaction output is timelocked it cannot be included in the blockchain until the specified time has elapsed. There are two types of timelocks, *absolute* [17]

and *relative* [18]. Transactions or transaction outputs with an absolute timelock become valid when the specified timestamp or block height is reached. On the other hand, a relative timelock allows a transaction output to be locked for a time relative to the block that included that output. We denote by Δt a relative timelock that expires t blocks (confirmations) after the transaction is included in the blockchain. After this time the output of the transaction is spendable.

Hashlocks. If a transaction output is hashlocked, it requires the pre-image of the specific hash to become valid and thus spendable. For instance, suppose $h(s)$ denotes the hash of a secret s. If an output is hashlocked with $h(s)$, it is valid only if the secret s is revealed.

2.1.3 UTXO Model

Bitcoin is UTXO-based (UTXO: Unspent Transaction Output), meaning that transactions consist of inputs and outputs. A transaction connects its inputs to UTXOs (removing the latter from the UTXO set) and creates new UTXOs, its outputs. Each UTXO can only be spent as a whole. A UTXO consists of a monetary value and the conditions under which it can be spent, e. g., a signature corresponding to a public key, a timelock, etc.

We denote by $o = (x \mid C)$ the UTXO that holds a monetary value of x coins that can be spent when conditions C are met. For example, $o = (10 \mid \sigma_A)$ means that the signature that matches the public key A can spend the output o which is equivalent to 10 coins.

A transaction in this model is a mapping from a set of UTXOs, the inputs, to a (new) set of UTXOs, the outputs. We define a transaction as follows:

$$TX_i = [o_j, o_k, \dots] \mapsto [o_i^1, o_i^2, \dots]$$

where o_j, o_k, etc. are the inputs of the transaction and $o_i^1, o_i^2, \dots$ are the first, second, etc. outputs, respectively. If transaction TX_i has a single output we simply write o_i. Moreover, if the specific UTXO that is input to a transaction is irrelevant to the protocol design, we refer to it as $\#$. If we demand the input to belong to a specific public key A and hold a specific value of x coins, but which of the UTXOs owned by A is spent by the input is irrelevant to the protocol design, we refer to it as $(x \mid \#_{\sigma_A})$. For instance,

$$TX_i = \left[(0.8 \mid \#_{\sigma_A}), o_k^2\right] \mapsto \left[(1 \mid h(s)), (0.5 \mid \sigma_B)\right]$$

denotes the transaction TX_i that spends a UTXO from the party A with value 0.8 coins and the second output of transaction TX_k and creates two new UTXOs. The first output holds the value of 1 coin and can be spent with the secret s, while the second holds 0.5 coins and can be spent with B's signature.

2.2 What is a Blockchain?

Definition 2.1 (Blockchain)**.** *A blockchain is a distributed append-only data structure where data are batched in blocks which are linked using cryptography.*

Blockchains are generally byzantine fault tolerant and immutable in practice. Due to their design, blockchains are commonly used in cryptocurrencies, i. e., to maintain a public immutable distributed transaction ledger.

Properties of blockchain protocols. In a groundbreaking work, Garay et al. [13] formally define the properties of blockchain protocols, and consequently what a public transaction ledger achieves. For this purpose, they assume a synchronous communication model (the protocols proceed in rounds) and a computationally bounded adversary that can corrupt t out of n participants. Note that in proof-of-work based blockchains this fraction expresses the limitation of the computational power of the adversary in the system. We present a simplified version of those properties below.

Definition 2.2 (Blockchain Protocol)**.** *A blockchain protocol must generally achieve the following properties:*

- *Common Prefix (parametrized by $k \in \mathbb{N}$): For any pair of honest parties P_1, P_2 adopting chains C_1, C_2 at rounds $r_1 \leq r_2$ respectively, it holds that $C_1^{\lceil k} \preceq C_2$, where $\preceq$ denotes the prefix relation and $C^{\lceil k}$ denotes chain C with the last k blocks pruned.*
- *Chain Growth (parametrized by $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$): For any honest party P with chain C, it holds that for any s rounds there are at least $\tau \cdot s$ blocks added to chain C of P.*
- *Chain Quality (parametrized by $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$): For any honest party P with chain C, it holds that for any l consecutive blocks of C the ratio of honest blocks in C is at least μ.*

Intuitively, the common prefix property states that all parties in the system share the same state with probability dependent on the depth of the "common prefix" k. Note that this property holds due to the synchrony assumptions.

Chain growth captures the need for the system to make progress, i. e., append new blocks. Chain quality, on the other hand, expresses the number of honest-party contributions that are contained in a sufficiently long and continuous part of an honest party's chain. This property is necessary to make sure *meaningful progress* is been made. Without chain quality the system can progress but produce for instance empty blocks (only from the adversary), or blocks that do not include specific transactions (censoring).

Public distributed transaction ledger. Blockchain protocols typically aim to solve one specific problem, that is, to maintain an efficient and secure public distributed transaction ledger. For this purpose, we present below the properties of a *robust* public distributed transaction ledger as defined in [13].

Definition 2.3 (Robust Public Distributed Transaction Ledger)**.** *A robust public distributed transaction ledger must generally achieve the following properties:*

- *Persistence (parameterized by the "depth" parameter $k \in \mathbb{N}$): If in a certain round an honest party reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger (such transaction will be called "stable"), then tx will be reported by any honest party in the same position in the ledger, from this round on.*
- *Liveness (parameterized by the "wait time" $u \in \mathbb{N}$, and the "depth" parameter $k \in \mathbb{N}$): Provided that a valid transaction is given as input to all parties continuously for u consecutive rounds, then all honest parties will report this transaction more than k blocks from the end of the ledger, i. e., all report it as stable.*

In other words, persistence states that all honest parties agree on all the transactions that were written in the ledger and in which order. Liveness, on the other hand, ensures that the ledger makes progress, meaning that transactions are eventually written in the ledger (included in the chain). It has been shown that a blockchain protocol that achieves common prefix, chain growth, and chain quality maintains a robust public distributed transaction ledger [13].

2.3 Consensus Variants

In terms of distributed computing, blockchains typically solve an old well-defined problem, called *Byzantine Agreement or Consensus* [19]. The goal of a consensus algorithm is to enable multiple nodes to maintain a common state or decide on a future action. More specifically, the nodes use consensus to agree on a common value out of the values they initially propose.

In this section, we define consensus as well as a weaker variant, called consistent broadcast [20], which we will employ later in Chapter 5. We further discuss the connection between blockchains and consensus protocols. We use consensus and byzantine agreement interchangeably.

Definition 2.4 (Byzantine Agreement)**.** *A byzantine agreement protocol must generally achieve the following properties:*

- *Agreement: All correct nodes decide on the same value upon termination.*

- *Termination: All correct nodes terminate in finite time.*
- *(Weak) Validity: If all correct nodes have the same input value v, the decision value must be v.*

In consensus protocols, the validity property exists to ensure that the algorithm does not invent a decision value by itself, e. g. all nodes always decide 0, because then the problem becomes trivial. Conversely, validity in the context of blockchains is slightly different. Strictly speaking, the property of weak validity is not guaranteed in blockchains; for instance, the Bitcoin miner that wins the hash-race and produces a block might be byzantine, yet the decision value of all honest miners will be the byzantine miner's block regardless of their inputs. Furthermore, such a validity definition is meaningless for blockchains because it is highly improbable that all honest miners share the same input, since the batching process most likely produces different blocks. On the other hand, all input blocks must be valid on an application layer because the miners do not accept blocks with wrong data. One could say that blockchains should provide a different validity property, which we term *probabilistic strong validity.*

- *Probabilistic Strong Validity: The decision value must be the input value of a correct node with significant probability.*

Note that this is essentially a probabilistic version of the strong validity definition stating that the decision value must be the input value of a correct node. Intuitively, this definition guarantees the same properties as chain quality, i. e., if we repeatedly use the consensus algorithm to include data in a ledger, a significant fraction of the data will be the input of correct nodes. In other words, if a blockchain protocol repeatedly employs a consensus protocol with probabilistic strong validity, then the blockchain protocol has good chain quality.

Similarly, the common prefix property is a probabilistic version of the agreement property, and termination is captured by chain growth. So blockchains are essentially systems that repeatedly employ byzantine fault-tolerant algorithms (often providing weaker properties) to maintain a common state among distributed nodes – in other words, *state machine replication* protocols. It is easy to see that the safety and liveness properties of state machine replication protocols correspond to the persistence and liveness properties of robust public distributed transaction ledgers (Section 2.2).

Definition 2.5 (Consistent Broadcast)**.** *Consistent broadcast is a distributed protocol run by a node that wants to send a message to a set of peers reliably. Consistent broadcast must generally achieve the following properties :*

- *Validity: If a correct node s broadcasts a message m, then every correct node eventually delivers m.*

- *Integrity: Every correct node delivers at most one message. If some correct node delivers a message m with sender s and sender s is correct, then m was previously broadcast by s.*
- *Consistency: If a correct node delivers a message m and another correct node delivers message m', then $m = m'$.*

It is called consistent because the sender cannot equivocate. In other words, the protocol maintains consistency among the delivered messages with the same sender but makes no provisions that any nodes do deliver the messages. So termination is not guaranteed with a faulty sender.

If we demand that consistent broadcast also ensures agreement on the delivery of the messages in the sense that either all correct nodes deliver some message or none delivers any message (known as *totality*), then we solve *reliable broadcast*. Any reliable broadcast protocol where all nodes eventually deliver (known as *termination*) solves byzantine agreement [21].

Timing assumptions. The feasibility of consensus variants strongly depends on the timing assumptions, i. e., the network delays and the relative process speeds of nodes. There are three distinct time-related models widely used in distributed computing: the synchronous, the partially-synchronous, and the asynchronous model [22].

Definition 2.6 (Asynchronous Model)**.** *There is no upper bound on processing or network delays.*

Definition 2.7 (Synchronous Model)**.** *There are known upper bounds on processing and network delays.*

Alternatively, we say a system is synchronous if the nodes have access to local physical clocks, which have a known upper bound on the rate at which they deviate from a global real-time clock.

Definition 2.8 (Partially-Synchronous Model)**.** *There are fixed bounds for both the processing and network delays, but they are not known a priori. Equivalently [23], there is a time (global stabilization time or GST) after which the system is synchronous, but this time is unknown.*

The partially-synchronous model captures the fact that the system must not always be synchronous and there is no bound on the period during which it is asynchronous. Nevertheless, the periods of synchrony should be long enough to guarantee a meaningful execution or termination of the algorithm. This model is also called *eventually synchronous* as it operates in asynchrony but eventually reaches a long-enough state of synchrony.

Now, we briefly discuss some important bounds of consensus under specific timing assumptions.

Theorem 2.9. *A synchronous network of n nodes cannot reach byzantine agreement with $f \geq n/3$ byzantine nodes [19, 24].*

The King algorithm [25] solves byzantine agreement among $3f + 1$ nodes where f are byzantine in the synchronous model. The algorithm requires $f + 1$ communication rounds to terminate, which is optimal [26, 27]. If we assume digital signatures and PKI, byzantine agreement can be solved with $f < n/2$. Recently Abraham et al. [28] presented a protocol that achieves constant number of communication rounds and $O(n^2)$ expected communication complexity with $f < n/2$ byzantine nodes for a strongly rushing adaptive adversary[1].

Theorem 2.10 (FLP)**.** *It is impossible for a deterministic protocol to achieve consensus in the asynchronous model, even with a single faulty node [29].*

FLP holds for faulty nodes, meaning the nodes simply crash, but do not deviate arbitrarily from the protocol execution (byzantine). Of course, FLP implies that no deterministic protocol can achieve byzantine agreement as well. To circumvent this impossibility, randomized algorithms where designed to solve consensus [30,31]. However, these consensus algorithms are quite complex and inefficient. The bound of Theorem 2.9 holds for the asynchronous and the partially-synchronous model ($f < n/3$) even for probabilistic algorithms [30].

Lastly, we observe that consensus variants are permissioned, meaning that the nodes participating in each algorithm are fixed and have identities. Blockchains on the other hand are typically permissionless. However, in a synchronous model it is easy to generate a permissioned setting from a permissionless one, by simply employing any Sybil-resistant mechanism like proof-of-work to assign permissions before the execution of the algorithm.

2.4 The Scalability Challenge

Blockchains lack fundamental research so no formal bounds exist to argue for scalability. Nevertheless, blockchain protocols are called to reach probabilistic byzantine agreement in a permissionless setting as discussed in Sections 2.3 and 2.2. As a result, they inherit the challenges consensus algorithms face, most notably with respect to performance. In this section, we provide an exposition on the scalability problem of blockchain protocols which stems from the consensus mechanisms they employ.

The greatest challenge blockchains face is to balance three fundamental properties: *decentralization, security,* and *scalability.* These three properties form a trilemma, as (so far) optimizing one results in compromising another. To

[1] A strongly rushing adaptive adversary can corrupt nodes on-the-fly and decides the adversarial strategy in each round after receiving all the honest parties' messages.

understand the trade-offs and the origin of the trilemma, we thoroughly examine all known approaches for building a blockchain protocol.

Bitcoin marked the genesis of blockchains introducing the so-called Nakamoto consensus. With Nakamoto consensus we refer to the combination of proof-of-work and the longest chain selection rule that comprise the consensus mechanism of Bitcoin. One could argue that Nakamoto consensus is scalable, given that the system can handle thousands of participants in contrast to previous consensus algorithms that reach their performance limits with hundreds of nodes (e. g. PBFT [32]). Such a claim is obviously wrong, as is evident from the limited transaction throughput of Bitcoin, despite the thousands of nodes operating it. To understand why, we first need to define what scalability is for blockchain systems.

Intuitively, a blockchain is scalable when it can achieve high transaction throughput. But this is merely a performance metric, and does not encompass the crux of scalability. For instance, there exist blockchain systems with high transaction throughput, yet these systems cannot scale to thousands of nodes [33, 34] because each block must be signed by every node in the network.

So what is scalability, really?

In a nutshell, scalability measures *how much the resource requirements of the nodes of the system increase with the growth of the system itself*, i. e., with the increase of nodes and transactions. The resources each node utilizes to participate in a blockchain system are *bandwidth, computation*, and *storage*. If we express each resource as a bivariate function of the number of nodes and the number of transactions, we can determine how the system scales in each dimension.

However, the number of transactions a blockchain system can accommodate is not arbitrary. On the contrary, it is limited by the security properties of the consensus protocol, and in particular, chain growth and chain quality. Chain growth[2] combined with chain quality show how many honest blocks are included in a chain during a specific number of rounds. Since blocks typically fit a specific number of transactions and rounds express the network delay or in other words time, the combination of chain growth and chain quality measures how many transactions per time unit the consensus protocol allows for. Thereby, we can calculate the maximum transaction throughput a protocol can handle securely under byzantine adversaries[3].

One could argue that when we ask for a scalable system, we basically want a system with high (maximum) transaction throughput with reasonable scaling

[2]The round complexity (or latency) of the consensus protocol is incorporated in the chain growth property.

[3]This is not the throughput shown in experimental evaluations as experiments cannot express byzantine adversaries and are typically executed for optimistic scenarios. When we refer to the transaction throughput of a protocol we mean the theoretical bound with byzantine adversaries.

functions for each resource. Now, the resource scaling functions depend on the design of each system. Most commonly the bottleneck in resources is bandwidth where the communication complexity of consensus manifests. Note, however, that the bandwidth requirement is the *maximum bandwidth* needed by *any* node in the network. Thus, communication complexity does not express accurately the bandwidth bottleneck as it only calculates the sum of all nodes, averaging out the resource among nodes.

For instance, Hotstuff [33] has linear communication complexity so one could argue it is better than PBFT [32]. The leader in Hotstuff though exchanges information with every other node to reach consensus, hence, the bandwidth requirement of this node is the same as that of every node in PBFT. In conclusion, both protocols have the same scaling function for bandwidth, which is linear to the number of nodes. Conversely, the Bitcoin miners do not need to communicate with each other to reach consensus, therefore the bandwidth scaling function does not depend on the number of nodes but solely on the transaction workload. The limitations of Bitcoin thus stem from the security analysis that bounds the maximum transaction throughput.

On that note, we should point out that all nodes in the network must receive, store and verify all the transactions in the blockchain. As long as transactions are not handled in parallel, e. g. sharding, or off-chain, all scaling functions are at least linear to the number of transactions.

Scaling solutions. We now discuss the main approaches for scaling blockchain protocols. There are five distinct approaches found in literature:

(a) Fine-tuning the consensus algorithm for maximum throughput (e. g., block size, block generation time)

(b) Alternative ways to reach consensus in a chain (e. g., proof-of-work, proof-of-stake, BFT)

(c) Alternative data structures (e. g., DAG)

(d) Sharding the state

(e) Transferring the transaction load off-chain (e. g., payment channels, sidechains)

All (a)-(d) methods are on-chain scaling solutions, while the last one is off-chain, meaning it is independent of the consensus layer but it builds on top of it; hence it is known as layer 2, the consensus being layer 1. We can further classify all (a)-(c) methods as different techniques to scale the consensus layer of blockchains, while both sharding and off-chain solutions are complementary to the consensus.

The first and simplest approach (a) is to fine-tune the parameters of the protocol to reach the maximum throughput possible. For example, increasing the

block size in Nakamoto consensus leads to increased propagation time which in turn affects the security properties, namely common prefix [13]. Intuitively, the miners that receive the block with long delay waste computational power mining on top of the previous block. This has both security implications (many orphan blocks) and leads to centralization as it inflates all resource requirements and induces a reward disadvantage to "smaller" nodes (either in computation or in network connectivity). Hence, the block size cannot increase arbitrarily, but a fine balance is possible [5]. In addition, Eyal et al. [35] show how performance can be significantly improved when transactions are decoupled from blocks.

Second, one can try to design alternative consensus mechanisms with high throughput that scale well. There are various ways to reach consensus in the wild. Some systems employ leader election based on the Sybil-resistance mechanism such as Bitcoin [1] and Ethereum [16] (proof-of-work), Ouroboros [36] (proof-of-stake), Chia [37, 38] (proof-of-space). Others elect a committee based on the Sybil-resistance mechanism, like Byzcoin [39] (proof-of-work) and Algorand [40] (proof-of-stake). Lastly, traditional BFT consensus algorithms may be employed once the setting becomes permissioned through a Sybil-resistance mechanism, such as EOS [41] (proof-of-delegated stake), proof-of-stake Hotstuff [33], Tendermint [42]. Many other works exist in the permissioned setting attempting to optimize performance under different timing assumptions, e. g. [34, 43].

Proof-of-work protocols are typically limited by the upper bound on the transaction throughput due to security reasons [5], while BFT protocols most commonly fail to scale horizontally, i. e., the resource requirements increase as nodes increase. Proof-of-stake protocols scale well compared to others, but provide less security and decentralization. All of these works are limited by the fact that all nodes must exchange, verify, and store all on-chain data.

Third, alternative structures to chains can be investigated. Directed acyclic graphs (DAGs) allow parallel data processing and achieve better performance than simple chains [44]. Many protocols exist that use a DAG structure [45–49].

As previously discussed, scaling the consensus layer has its limitations. To circumvent these bounds, sharding was proposed, i. e., processing transactions in parallel by different nodes. Sharding can achieve horizontal scaling of blockchain protocols.

Lastly, off-chain protocols, such as payment channels, are the most promising solution for scaling blockchains. The reason is that off-chain protocols operate on top of the consensus layer hence can be employed by already deployed blockchain systems, and most importantly they enable unbounded scaling.

In the following sections, we discuss sharding and off-chain solutions in detail, as these are the focus of this thesis.

2.5 Sharding a Blockchain

Sharding is not a new idea, but it originated in the 90s to optimize the performance of traditional centralized databases. The main idea is to break a large database into smaller ones that are easily manageable, and therefore reduce the query response time. A common example of sharding a large database is to break up the customer database into geographic locations. Customers in the same geographic locations are grouped together and placed on unique servers.

In the context of blockchains, sharding is an on-chain solution that allows blockchain protocols to *scale horizontally*, i. e., protportionally to the number of nodes. The core idea behind sharding is that it enables the parallel processing of transactions by different subset of nodes and therefore reduces the resource requirements for each node in the system. To achieve this, sharding protocols employ multiple chains in parallel, the *shards*, that operate under the same consensus. Each shard handles only a fraction of the transaction load and is operated by a fraction of the nodes.

Although sharding seems like a straightforward way to scale decentralized blockchains, it faces many challenges. We list some of these challenges below to demonstrate the difficulty of building a secure and efficient decentralized sharded blockchain system.

Single shard take over attacks. When the nodes are partitioned to shards, the adversary can intentionally take over a shard by corrupting the respective nodes. In such a case, the adversary can perform a number of malicious acts; for instance, the adversary can censor transactions, or even worse commit invalid transactions. To avoid this problem, most often sharding protocols assume a static or slowly-adaptive adversary (changing the corrupted nodes is not allowed or is very slow, respectively) and the nodes are randomly assigned to shards by protocol design.

Randomness generation. Suppose the adversary can bias the randomness and for example affect the node-to-shard assignment process. In such a case, the security of the protocol is broken as explained previously. To address this problem, communication-heavy distributed randomness generation (DRG) protocols are employed but they are only executed sporadically to maintain the amortized efficiency of the protocol. There are various DRG protocols based on different cryptographic primitives, such as hash functions [1, 50], verifiable random functions [40, 51], threshold signatures [52, 53], and most commonly public verifiable secret sharing [36, 54–56].

State transition. When the system reassigns nodes to shards, the nodes must bootstrap to the new shard to verify future transactions. This is quite costly and eventually becomes a slow and inefficient process especially as the time passes by and the shard chains grow. Most shading protocols ignore this problem, because typically experimental evaluations do not reflect it. Checkpoints have been used to circumvent this issue [9].

Cross-shard communication. Sharding protocols partition the transaction state and each shard handles only a fraction of addresses. But what happens if a transaction involves two or more shards (cross-shard)? Cross-shard transactions should either commit or abort in all shards to maintain the atomic property of transactions. Thus, the shards responsible for the inputs of the transaction must verify the transaction, commit it on-chain and consequently inform the shards handling the output of the transaction. This process should be done efficiently, meaning that the overhead from the cross-shard communication must not exceed the benefits of sharding. This is arguably the greatest challenge of sharding protocols with respect to performance.

Fraud detection. Suppose a single shard take over takes place, and the adversary manages to double-spend. In a single blockchain this implies that the chain was probably reversed after the "commonly accepted" confirmation delay (e. g., 6 blocks in Bitcoin), and the system has a new shared state which is consistent. However, in sharding this is not necessarily true. The adversary can double spend the coins in different shards resulting in an inconsistent system (new money were generated). To fix this issue fraud-proofs were suggested to let an honest node prove later the inconsistency to the affected shards and revert it.

Data availability. Data availability is a weaker form of fraud detection. In this case, data is missing making for instance the verification of a committed block impossible. To ensure data availability often Coded-Merkle Trees [57] or erasure codes [58] are used.

Various sharding protocols that address most of these issues have been proposed in literature, the most prominent of which we examine thoroughly in Chapter 3, where we also delve into the sharding trade-offs and protocol design rationale.

2.6 Payments Channels (Layer 2)

2.6.1 Foundations

"One use of nLockTime is high frequency trades between a set of parties. They can keep updating a tx by unanimous agreement. The party giving money would be the

first to sign the next version. If one party stops agreeing to changes, then the last state will be recorded at nLockTime. ... Only the final outcome gets recorded by the network."

Satoshi Nakamoto [59]

In a personal email to Mike Hearn, Satoshi Nakamoto himself introduced the idea of "high frequency trades" commonly known today as *payment channels*. Shortly after, Jeremy Spilman proposed the first implementation of unidirectional payment channels [10].

Payment channels operate on top of the blockchain (Layer 2) and allow instant off-chain transactions. When capital can flow only in one direction, from a payer to a payee, and it is not possible to transfer capital back in the reverse direction, the channel is called unidirectional. Spilman channels exposed the payer to a malleability risk where the payee would be able to hold the payer's capital hostage. This malleability was addressed by employing a new Bitcoin script opcode in 2015 [17].

Despite the ingenious idea, unidirectional channels lack the flexibility of sending payments in both directions. As a result, the capital moving in one direction is bound to deplete quickly. Bidirectional payment channels were introduced to resolve this problem. The idea is that any two parties that interact (often) with each other can set up a joint account on the blockchain, i. e., a channel. Using this channel, the two parties can transact off-chain, sending capital back and forth by just sending each other signed messages. The two parties rely on the blockchain as a fail-safe mechanism in case of disputes.

Generally, a channel is set up by two parties[4] that deposit capital in a joint account on the blockchain. The channel can then be used to make arbitrarily many transactions without committing each transaction to the blockchain. When opening a channel, the parties pay a blockchain fee and place capital in the channel. The blockchain fee is the transaction fee to the miner, paid to have the transaction included in a block and thereby published on the blockchain. The deposited capital funds future channel transactions and is not available for other transactions on the blockchain during the channel's lifetime.

There are multiple proposals on how to construct payment channels [11, 12, 61–64], but all proposals share the same core idea: The two parties create a joint account on the blockchain (funding transaction). Every time the parties want to make a transaction they update the distribution of the capital between them accordingly and they both sign the new transaction as if it would be published on the blockchain (update transaction). To close the channel, a party publishes the latest update transaction either unilaterally or along with the counterparty with a closing transaction.

[4] Note that a channel can also be created between multiple parties [60].

The various proposals differ in the way they handle disputes, i. e., the case where one of the parties misbehaves and attempts to close the channel with a transaction that is not the latest update transaction. For instance, Duplex channels [11] use timelocks to guarantee that the latest update transaction will become valid before any previous update transaction. Duplex channels however do not incentivize parties to behave honestly, as parties are not penalized when they attempt to commit fraud. On the contrary, a rational party should attempt to commit fraud as their expected profit can only increase. Lightning channels [12] on the other hand, penalize the misbehaving party by awarding the funds of the channel to the counterparty in case of fraud. Therefore, Lightning channels provide stronger incentives for behaving honestly.

To open a Lightning channel, the parties publish a funding transaction where they lock their funds into a common account, i. e., both parties must sign to spend the output of the funding transaction. Every time the parties execute a transaction, they update the current state of the channel accordingly, meaning they distribute the funding transaction's output as agreed and sign the resulting "commitment" transaction. In addition, each party reveals to the counterparty a secret that allows the counterparty to sign a "revocation" transaction (or breach remedy) that spends the previous commitment transaction; the revocation transaction awards the cheating party's funds to the cheated party, effectively punishing the party that tried to cheat. The output of the party that published the commitment transaction is locked for a time period, known as the revocation (or dispute) period. The cheated party must publish the revocation transaction during the revocation period, otherwise the cheating party will be able to spend the balance of the revoked state.

To extend the concept of payment channels on an arbitrary state, *state channels* were introduced [65]. Several recent constructions exist in this direction [65–67].

Payment channels fundamentally require parties to lock up funds in advance and further do not allow cross-channel transactions. Thus, each pair of nodes that interact often must build and operate their own channel, leading to a very dense payment network and vast amounts of capital locked in a non-flexible manner. Can we somehow alleviate this requirement?

2.6.2 Payment Channel Networks (PCN)

Multiple channels form a *payment channel network*. To enable the transaction execution between two nodes that are not directly connected in the network, routing the transaction along a path of directly connected nodes is allowed. In such a case, we demand atomic execution of the transaction; either all payments in the path will be executed or none. To guarantee the atomic execution of transactions, Hashed Timelocked Contracts (HTLCs) [11, 12] can be used.

HTLCs use decreasing timelocks to guarantee the payments can be safely

reversed in case a node in the routing path fails. Furthermore, HTLCs utilize hashlocks that allow the parties to sign a conditional payment that will be executed if the hash pre-image is revealed. This is how they chain the payments in the path and as long as the first payment is executed the secret is revealed and thus every payment will go through.

For example, if Alice wants to send capital to Charlie through Bob that is an intermediate node in the PCN, Charlie provides Alice with a hash of the secret that only Charlie knows. Alice then sends the transaction to Bob with the condition that Bob can only claim the capital sent by Alice if Bob provides the secret of Charlie. Then, Bob sends the same amount to Charlie with the additional condition that Charlie can only claim the capital if Charlie provides the secret. If Charlie executes the transaction and thus reveals the secret, then Bob can claim the money from Alice. Hence, either the whole chain of transactions will be executed or none, therefore ensuring the atomic property of transactions.

Routing in a payment network alleviates the necessity for direct links, thus the need for multiple channels. However, the update transactions in a payment channel are executed in private thus it is impossible to know the current distribution of the capital on a channel. This in turn leads to complex routing schemes, because often the chosen route is depleted.

Payment hubs. A predominant network structure for payment channel networks has emerged: the star graph. PCNs that form a star graph, or so-called *payment hubs* [68–70], allow users to interact efficiently through the PCN as the star topology facilitates transactions with at most two hops. However, payment hubs are a centralized network structure and as such introduce security risks to the payment network. For this reason, many payment hub solutions occasionally publish data on-chain to assure data availability or/and fraud detection [68] (see Section 2.7).

2.6.3 Shortcomings

Payment channels promise to solve the scalability problem of blockchains. However, they suffer from major drawbacks.

First, payment channels are *capital-hungry*. To operate a payment channel participants must lock capital a priori, estimating the amount of capital they will need for future transactions either their own or for routing transactions of others. The locked capital remains unavailable for any other use at least for the dispute period which is typically quite long. The *long dispute period* is necessary to guarantee security; reducing the dispute period might result in one party cheating the temporarily offline counterparty by closing the channel in a previous more profitable state. Furthermore, a party can easily *denial-of-service* the counterparty

for a short period of time to make sure the revocation transaction does not make it on-chain. This is the second shortcoming of payment channels.

Lastly, channel safety depends on the properties and timing assumptions of the underlying blockchain, and in particular, the liveness property of the transaction ledger which of course depends on the *synchrony* of the network. An adversary that can successfully censor the revocation transaction during the dispute period can successfully close the channel in a previous more profitable state, thus steal funds from the channel. This is the third challenge channel solutions face.

In this section, we discuss in depth these drawbacks and potential solutions. There are other challenges payment channels come up against but it this thesis we focus on the aforementioned ones.

The availability problem & watchtowers. The security of channels relies on participants being responsive and actively watching the blockchain. The main problem is that one party might try to settle on an old more profitable state of the channel, in which case the other party needs to publish a proof-of-fraud (e. g., the revocation transaction in Lightning) during the dispute period t. Thus, fraud can be proven and punished, as long as both parties are often online – once every t – and monitor the blockchain. To exacerbate this issue, a party can always launch a denial-of-service attack against the counterparty to prevent the proof-of-fraud from being published in the blockchain.

A naive solution to this problem would be to open channels with a long dispute period. Then the parties need to be online only occasionally while launching a denial-of-service attack becomes costly. But this means that the capital locked in the channel will be unavailable for a long time, making channels even less lucrative for users.

As a countermeasure, the Bitcoin community proposed to outsource this job to third parties, called *watchtowers* [71]. Watchtowers act as proxies for the party that hire them, allowing the party to go offline for a long period of time. In case the counterparty attempts to close the channel in a previous state, the watchtowers act on behalf of the offline party and publish the revocation transaction.

The watchtowers in the Lightning protocol, or so-called Monitors [71], mainly focus on maintaining privacy. The current design does not provide incentives for participation for the watchtowers. In particular, the party that hires a watchtower pays the watchtower either every time a channel update occurs or when fraud happens. In the first case, the watchtower can collect the rewards without acting upon fraud, so the security of the channel is not guaranteed. In case the watchtower is paid upon fraud, the counterparty knowingly-so will not commit fraud, thus the watchtower will never be paid. Consequently, the watchtowers are not incentivized to participate.

Avarikioti et al. [72] proposed DCWC, a protocol that builds a network of

watchtowers around each participant of the payment channel network. The watchtowers are incentivized to forward channel updates to other watchtowers faithfully executing the protocol. Nevertheless, DCWC does not incentivize watchtowers' participation with similar security implications as Monitors.

Pisa [73] employs smart contracts to solve this problem. Specifically, third-parties called Custodians are hired by the channel parties to act on their behalf, and they are penalized in case they do not act upon fraud. Pisa provides rewards and punishments to incentivize the faithful protocol execution, but it allows the capital of each Custodian to be used in multiple channels. As a result, the Custodian can double-spend the capital in two channels and make profit by closing both channels in previous states in collaboration with the channel parties (one from each channel). Furthermore, Pisa does not allows parties to go offline indefinitely; in case of fraud, a misbehaving Custodian must be punished on-chain within a specific time period. Nevertheless, the punishment time period is much longer than the dispute period.

Is it possible to have incentive compatible watchtowers for Bitcoin? Many speculated the answer is no. We answer this question to the affirmative in Chapter 4.

The synchrony problem. Another problem that arises even when we manage to address the incentive-compatibility of watchtowers, is the dependency of the channels' safety on the synchrony assumptions of the blockchain. The revocation transaction or any type of proof-of-fraud, e. g. watchtower's punishment, must be published on-chain within a specific time period (using timelocks). Subsequently, if one can successfully break the liveness of the blockchain, e. g., censor the revocation transaction whether it is published by the counterparty or a watchtower, they can steal funds from the payment channel.

Here we present a concrete attack against Pisa [73]. This attack applies to any channel construction that has a time-out dispute process. The attack is simple and relies on delaying the appearance of the dispute transaction on-chain until the dispute window expires. The attack is as follows:

1. Alice and Bob establish a state-channel. As Pisa constructs, both Alice and Bob chose their favorite watchtower and provably assign to them the job of watching the chain.

2. Alice sends multiple coins to Bob through state changes $S_1, S_2, ..., S_n$.

3. Alice request the closure of the channel with state S_1 where Alice holds more coins that S_n.

4. Bob or Bob's watchtower detects the attack and sends on the network a dispute transaction for S_n.

5. Alice attacks Bob in one of the following ways:
   (a) Congests the blockchain so that the dispute transaction for S_n does not appear on-chain until after the dispute window expires.
   (b) Depending on the blockchain, Alice can censor the dispute transaction for S_n without congesting the network [74, 75].
   (c) If Alice is a classic asynchronous network adversary [30], Alice can delay the delivery of the dispute transaction for S_n until after the dispute window.
6. Since the dispute transaction for S_n appears after the dispute window expires, the safety of the channel is violated.

In Pisa this attack might cause the watchtower to be punished although the watchtower did not misbehave. In other channels [65, 76], that do not employ or do not collateralize watchtowers, this attack leads to direct value loss for Bob.

We propose a solution for this problem is Chapter 5.

Capital optimization. A major disadvantage of payment channel networks is the need for parties to lock capital in the channels in advance. The locked capital is unavailable for any other use due to safety requirements. Thus, minimizing the capital locked in the network is critical for the widespread adoption of channels.

We investigate various network structures and provide insight regarding the optimal capital assignment, either from a centralized point of view or in a decentralized manner, in Chapters 6 and 7, respectively.

2.7 Other Scaling Solutions

In this section, we briefly discuss other scaling solutions. Figure 2.1 illustrates the space of existing scaling solutions.

Sidechains. Sidechains [77, 78] are independent blockchains which employ their own consensus models and block parameters to efficiently process transactions, while maintaining compatibility with a main chain, e. g. Ethereum [16]. In some sense, sidechains lie in between channels and sharding. Similarly, to channels, sidechains transfer the transaction workload off the main chain, thus allow for arbitrary scaling. The two-way peg used to transfer money back and forth from the main chain to the sidechain is essentially the opening and closing process of a channel. On the other hand, sidechains typically enable the cross-chain transfer of data (interoperability), much like sharding enables cross-shard transactions.

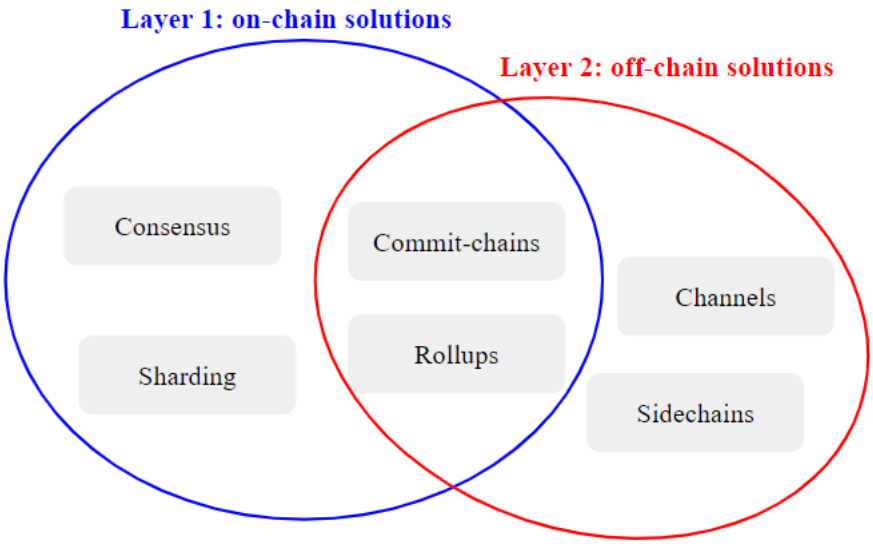


Figure 2.1: Blockchain scaling solutions.

Nevertheless, sidechains are considered a layer 2 solution as they operate independently and in a parallel capacity to the main chain, resembling channels regarding scalability. Security is responsibility of each sidechain and it is not inherited from the main chain. The nodes that operate each sidechain are responsible for verifying and confirming transactions, choosing and running the consensus protocol; the chosen consensus typically aims for faster transaction confirmation time. For instance, a payment channel is basically a sidechain where the nodes must reach unanimous agreement to update the state. One might say that sidechains are a generalization of channels. Other common consensus mechanisms employed by sidechains are proof-of-work [78], proof-of-stake [79] by xDai, proof-of-authority by POA Network, BFT protocols by Skale and Matic Network.

Public sidechains are often adopted for application-specific transactions (e. g. DAO-voting), stable transactions, and micro-transactions that do not require the same security guarantees as high-value financial transactions and can therefore be optimized for speed and cost.

Commit-chains. Commit-chains [68,80] are protocols for operating non-custodial untrusted payment channel hubs. Basically, commit-chains are maintained by a single party, the operator of the hub, that acts as an untrusted intermediary for facilitating transactions. Users can join the commit-chain by contacting the operator, and they can anytime withdraw or move their assets to the parent chain. Unlike payment channels or sidechains, the operator interacts frequently with the parent chain creating checkpoints which confirm the off-chain transactions.

The basic idea of commit-chains is that the operator periodically commits on-chain compressed information about the off-chain transactions. There are two different proposals on how to implemented this idea, ZK-rollups and optimistic rollups. Both share the same concept of bundling multiple off-chain transfers in a single on-chain transaction, hence they *rollup* transactions. ZK-rollups on the one hand are based on zero-knowledge proofs (STARKs) which are committed on-chain for each bundle of transactions enforcing consistent state transitions. They implement proactive security. Optimistic rollups on the other hand implement reactive security; the operator publishes on-chain only the Merkle root of the bundled transactions. Any user can dispute the operator during the challenge period by publishing the valid Merkle root and the corresponding Merkle proofs. If fraud occurs, the operator is slashed and the layer 2 is rolled back.

Commit-chains face a great challenge: data availability. Upon receiving a transaction from a user, the operator returns to the user a signed confirmation or some other proof of receiving the transaction. This proof is necessary to prove the misbehavior of the operator, i. e., if the operator publishes on-chain an invalid rollup. A malicious operator can simply not send the proof to the user but there is no way of showing the absence of proof (or data unavailability) is intentional because the user does not know if the operator actually received the transaction. If the required data is not available, the user is either forced to exit the commit-chain (e. g. Plasma [80]) or challenges the operator to provide the necessary data (e. g. Nocust [68]). If the operator misbehaves the users can always exit the commit-chain in the last on-chain confirmed state (checkpoint). Therefore, commit-chains offer eventual finality of transactions after an on-chain checkpoint, in contrast to payment channels that offer instant finality. The reason is that the operator does not lock any on-chain collateral to securely route the transactions. If, however, the operator locks on-chain collateral for each user, essentially implementing payment channels on top of the commit-chain, transactions are instantly final.

Famous implementations of commit-chains are Plasma Cash [80], a UTXO-based commit-chain on Ethereum using optimistic rollups, and Nocust/Nocust-ZKP [68] an account-based commit-chain employing optimistic rollups/ZK-rollups. Note that ZK-rollups and optimistic rollups are general techniques that can be applied in sidechains as well, i. e., their use is not limited to payment hubs and a single operator.

So far, interoperability of commit-chains has not been explored. Hence, enabling atomic cross-(commit-chain) transactions remains an open question.

Part III

Sharding Blockchains

Chapter 3

Formalization of Sharding Protocols

3.1 Introduction

3.1.1 Motivation

One of the most promising solutions to scaling blockchain protocols is sharding, e. g. [6–9]. However, there is no formal definition for a robust sharded ledger (similar to the definition of what a robust transaction ledger is [13]), which leads to multiple problems. First, each protocol defines its own set of goals, which tend to favor the presented protocol design. These goals are then confirmed achievable by experimental evaluations that demonstrate their improvements. Additionally, due to the lack of robust comparisons (which cannot cover all possible byzantine behaviors), sharding is often criticized as some believe that the overhead of transactions between shards cancel out the potential benefits. In order to fundamentally understand sharding, one must formally define what sharding really is, and then see whether different sharding techniques live up to their promise.

In this work, we take up the challenge of providing formal "common grounds" under which we can capture the sharding limitations and fairly compare different sharding solutions. We achieve this by defining a formal sharding framework as well as formal bounds of what a sharded transaction ledger can achieve. Then, we apply our framework to multiple sharded ledgers and identify why they do not satisfy our definition of a robust sharded transaction ledger.

To maintain compatibility with the existing models of a robust transaction ledger, we build upon the work of Garay et al. [13]. We generalize the blockchain transaction ledger properties, originally introduced in [13], namely *Persistence* and *Liveness*, to also apply on sharded ledgers. Persistence essentially expresses the agreement between honest parties on the transaction order, while liveness

encompasses that a transaction will eventually be processed and included in the transaction ledger. Further, we extend the model to capture what sharding offers to blockchain systems by defining *Consistency* and *Scalability*. Consistency is a security property that conveys the atomic property of *cross-shard transactions* (transactions that span multiple shards and should either abort or commit in all shards). Scalability, on the other hand, is a performance property that encapsulates the speedup of a sharded blockchain system compared to a non-sharded blockchain system.

Once we define the properties, we explore the boundaries of sharding protocols to maintain the security and performance properties of our framework. We observe a fine balance between consistency and scalability. To truly work, a sharding protocol needs to scale in computation, bandwidth, and storage. The computational cost of each participant in a blockchain system is dominated by verifying the consistency of the updates (either directly executing them or verifying proofs of consistency). We show that each participant of a sharding protocol must only participate in the verification process of a constant number of shards to satisfy scalability. Additionally, we show that scalable sharding is impossible if participants maintain (and verify) information on all other shards with which they execute cross-shard transactions. This happens because the storage requirements become proportional to that of a non-sharded protocol.

Finally, the high communication complexity of the underlying protocols, i. e., consensus of shards, handling cross-shard transactions, and bootstrapping of participants to shards, can make the bandwidth become the bottleneck. We identify a trade-off between the bandwidth requirements and how adaptive the adversary is, i. e., how "quickly" the adversary can change the corrupted nodes. If the adversary is not static, maintaining security requires shard reassignments and the participants need to bootstrap from the stored and ever-increasing blockchain of the shard. In this work, we prove that *in the long run* there is no sharding protocol that scales better than the blockchain substrate against an adaptive adversary. Furthermore, we show that against a somewhat adaptive adversary, sharding protocols must periodically compact the state updates (e. g., via checkpoints [9], cryptographic accumulators [81], zero-knowledge proofs [82,83] or other techniques [84–87]); otherwise, the necessary bandwidth resources of each participant increase infinitely in time, eventually exceeding the resources needed in a non-sharded blockchain.

Once we provide solid limits and bounds on the design of sharding protocols, we identify six specific components that are critical in designing a robust permissionless sharded ledger: (a) a core consensus protocol for each shard, (b) an atomic cross-shard communication protocol that enables transferring of value across shards, (c) a Sybil-resistance mechanism that forces the adversary to commit resources in order to participate, (d) a process that guarantees honest and adversarial nodes are appropriately dispersed to the shards to defend security against adversarial adaptivity, (e) a distributed randomness generation protocol

that is critical for security, (f) a process to occasionally compact the state in a verifiable manner. We then employ these components to introduce a protocol abstraction, termed the sharding crux, that achieves robust sharding in our model. We explain the design rationale and provide security proofs while we determine which components affect the efficiency of the protocol abstraction, i. e., scalability and throughput.

To demonstrate the power of our framework, we further evaluate the most original and impactful[1] permisionless sharding protocols. Specifically, we describe, abstract, and analyze Elastico [7] (inspiration of Zilliqa), OmniLedger [9] (inspiration of Harmony), Monoxide [8], and RapidChain [6]. We demonstrate that all sharding systems fail to meet the desired properties in our system model. Elastico and Monoxide do not actually (asymptotically) improve on non-sharded blockchains according to our model. OmniLedger is susceptible to a liveness attack where the adaptive adversary can simply delete a shard's state effectively preventing the system's progress. Nevertheless, a simple fix is possible such that OmniLedger satisfies all the desired properties in our model. Last, we prove RapidChain meets the desired properties but only in a weaker adversarial model. For all protocols, we provide elaborate proofs while for OmniLedger and RapidChain we further estimate how much they improve over their blockchain substrate. To that end, we define and use a throughput factor, that expresses the average number of transactions that can be processed per round. We show that both OmniLedger and RapidChain scale optimally, reaching the upper bound $O(\frac{n}{\log n})$.

3.1.2 Contribution

In summary, the contribution of this chapter is the following:

- We introduce a framework where sharded transaction ledgers are formalized and the necessary properties of sharding protocols are defined. Further, we define a throughput factor to estimate the transaction throughput improvement of sharding blockchains over non-sharded blockchains (Section 3.2).
- We provide the limitations of secure and efficient sharding protocols under our model (Section 3.3).
- We identify the critical and sufficient ingredients for designing a robust sharded ledger (Section 3.4).
- We evaluate existing protocols in our model. We show that Elastico, Monoxide, and OmniLedger fail to satisfy some of the defined properties in our model, whereas RapidChain satisfy all the necessary properties to maintain

[1] We consider all sharding protocols presented in top tier computer science conferences.

a robust sharded transaction ledger but only under a weaker adversarial model (Section 3.5).

3.2 The Sharding Framework

In this section, we introduce a formal definition of sharded transaction ledgers and define the desired properties of a secure and efficient distributed sharded ledger. Further, we employ the properties of blockchain protocols defined by Garay et al. [13], which we assume to hold in every shard. Given these properties, we later evaluate if the existing sharding protocols maintain secure and efficient sharded transaction ledgers. In addition, we define a metric to evaluate the efficiency of sharding protocols, the transaction throughput. To assist the reader, we provide a glossary of the most frequently used parameters in Table 3.1.

Table 3.1: (Glossary) The parameters in our analysis.

n	number of parties
f	number of byzantine parties
m	number of shards
v	average transaction size (number of inputs and outputs)
E	epoch, i. e., a set of consecutive rounds
T	set of transactions (input)
k	"depth" security parameter (persistence)
u	"wait" time (liveness)
Σ	scaling factor
ω_m	communication overhead factor
ω_s	space overhead factor
ω_c	computational overhead factor
σ	throughput factor
μ	chain quality parameter
τ	chain growth parameter
v	average transaction size
m'	degree of parallelism
γ	average number of a shard's interacting shards (cross-shard)

3.2.1 The Model

Network model. We analyze blockchain protocols assuming a synchronous communication network. In particular, a protocol proceeds in *rounds*, and at the end of each round the participants of the protocol are able to synchronize, and all messages are delivered. A set of R consecutive rounds $E = \{r_1, r_2, \ldots, r_R\}$ defines

an *epoch*. We consider a fixed number of participants in the system denoted by n. However, this number might not be known to the parties.

Threat model. The adversary is slowly-adaptive, meaning that the adversary can corrupt parties on the fly at the beginning of each epoch but cannot change the malicious set of participants during the epoch, i. e., the adversary is static during each epoch. In addition, in any round, the adversary decides its strategy after receiving all honest parties' messages. The adversary can change the order of the honest parties' messages but cannot modify or drop them. Furthermore, the adversary is computationally bounded and can corrupt at most f parties during each epoch. This bound f holds strictly at every round of the protocol execution. Note that depending on the specifications of each protocol, i. e., which Sybil-attack-resistant mechanism is employed, the value f represents a different manifestation of the adversary's power (e. g., computational power, stake in the system).

Transaction model. We assume UTXO-based transaction ledgers (see Section 2.1.3), similarly to Bitcoin and in contrast to a typical account-based database. Note that transactions can have multiple inputs and outputs. We define the average *size* of a transaction, i. e., the average number of inputs and outputs of a transaction in a transaction set, as a parameter v [2]. Further, we assume a transaction set T follows a distribution D_T (e. g. D_T is the uniform distribution if the sender(s) and receiver(s) of each transaction are chosen uniformly at random from all possible users). All sharding protocols considered in this chapter are UTXO-based.

3.2.2 Sharded Transaction Ledgers

In this section, we define what a robust sharded transaction ledger is. We build upon the definition of a robust public transaction ledger introduced in [13] (see Section 2.2). Then, we introduce the necessary properties a sharding blockchain protocol must satisfy in order to maintain a robust sharded transaction ledger.

A sharded transaction ledger is defined with respect to a set of valid[3] transactions T and a collection of transaction ledgers for each shard $S = \{S_1, S_2, \ldots, S_m\}$. In each shard $i \in [m] = \{1, 2, \ldots, m\}$, a transaction ledger is defined with respect to a set of valid ledgers[4] S_i and a set of valid transactions. Each set possesses an efficient membership test. A ledger $L \in S_i$, is a vector of sequences of transactions $L = \langle x_1, x_2, \ldots, x_l \rangle$, where $tx \in x_j \Rightarrow tx \in T, \forall j \in [l]$.

[2] This way v correlates to the number of shards a transaction is expected to affect. The actual size in bytes is proportional to v but unimportant for measuring scalability.

[3] Validity depends on the application using the ledger.

[4] Only one of the ledgers is actually committed as part of the shard's ledger, but before commitment there are multiple potential ledgers.

In a sharding blockchain protocol, a sequence of transactions $x_i = tx_1 \dots tx_e$ is inserted in a block which is appended to a party's local chain C in a shard. A chain C of length l contains the ledger $L_C = \langle x_1, x_2, \dots, x_l \rangle$ if the input of the j-th block in C is x_j. The *position* of transaction tx_j in the ledger of a shard L_C is the pair (i, j) where $x_i = tx_1 \dots tx_e$ (i. e., the block that contains the transaction). Essentially, a party *reports* a transaction tx_j in position i only if its local ledger of a shard includes transaction tx_j in the i-th block. We assume that a block has constant size, meaning there is a maximum constant number of transactions included in each block.

Furthermore, we define a symmetric relation on T, denoted by $M(\cdot, \cdot)$, that indicates if two transactions are conflicting, i. e., $M(tx, tx') = 1 \Leftrightarrow tx, tx'$ are conflicting. Note that valid ledgers can never contain conflicting transactions. Similarly, a valid sharded ledger cannot contain two conflicting transactions even across shards. In our model, we assume there exists a verification oracle denoted by $V(T, S)$, which instantly verifies the validity of a transaction with respect to a ledger. In essence, the oracle V takes as input a transaction $tx \in T$ and a valid ledger $L = \langle x_1, x_2, \dots, x_l \rangle \in S$ and checks whether the transaction is valid and not conflicting in this ledger; formally, $V(tx, L) = 1 \Leftrightarrow \exists tx' \in L$ s.t. $M(tx, tx') = 1$ or $L' = \langle x_1, x_2, \dots, x_l, tx \rangle$ is an invalid ledger.

Next, we introduce the security and performance properties a blockchain protocol must uphold to maintain a robust and efficient sharded transaction ledger: persistence, consistency, liveness, and scalability.

Intuitively, *persistence* expresses the agreement between honest parties on the transaction order, whereas *consistency* conveys that cross-shard transactions are either committed or aborted in all shards. On the other hand, *liveness* indicates that transactions will eventually be included in a shard, i. e., the system makes progress. Last, *scalability* encapsulates the speedup a sharded system can gain in comparison to a non-sharded blockchain system: The blockchain throughput limitation stems from the need for data propagation, maintenance, and verification from every party. Thus, to scale a blockchain protocol via sharding, each party must broadcast, maintain and verify mainly local information.

Definition 3.1 (Persistence)**.** *Parameterized by $k \in \mathbb{N}$ ("depth" parameter), if in a certain round an honest party reports a shard that contains a transaction tx in a block at least k blocks away from the end of the shard's ledger (such transaction will be called "stable"), then whenever tx is reported by any honest party it will be in the same position in the shard's ledger.*

Definition 3.2 (Consistency)**.** *Parametrized by k ("depth" parameter), $v \in \mathbb{N}$ (average size of transactions) and D_T (distribution of the input set of transactions), there is no round r in which there are two honest parties P_1, P_2 reporting transactions tx_1, tx_2 respectively as stable (at least in depth k in the respective shards), such that $M(tx_1, tx_2) = 1$.*

To evaluate the system's progress, we assume that the block size is sufficiently large, thus a transaction will never be excluded due to space limitations.

Definition 3.3 (Liveness)**.** *Parameterized by u ("wait time"), k ("depth" parameter), $v \in \mathbb{N}$ (average size of transactions) and D_T (distribution of the input set of transactions), provided that a valid transaction is given as input to all honest parties of a shard continuously for the creation of u consecutive rounds, then all honest parties will report this transaction at least k blocks from the end of the shard, i. e., all report it as stable.*

Scaling distributed ledgers depends on three vectors: *communication*, *space*, and *computation*. In particular, to allow high transaction throughput all these vectors should ideally be constant and independent of the number of participants in the system.

First, we define a *communication overhead factor* ω_m as the communication complexity of the protocol scaled over the number of participants. In essence, ω_m represents the average amount of sent or received data per participant in a protocol execution; in other words, the average bandwidth resource requirements of the system's participants. The communication overhead factor expresses both the bandwidth requirements of the protocol execution within an epoch (i. e., within and across shards communication), as well as the bandwidth resources necessary during epoch transitions. We observe that the latter becomes the bottleneck for scalability in the long run as rotating participants must bootstrap to new shards.

We notice, however, that in practise there are protocols that maintain low communication complexity – by employing efficient diffusion mechanisms of data such as gossip protocols – but fail to scale in another dimension. A notable such example is the Bitcoin protocol where the bottleneck is space and computation, while the communication overhead is minimal. For this reason, we introduce the *space overhead factor* ω_s that estimates the space complexity of a sharding protocol, i. e., how much data is stored in total by all the participants of the system in comparison to a single database that only stores the data once. The space overhead factor ranges from $O(1)$ to $O(n)$, where n is the fixed number of participants in the protocol. For instance, the Bitcoin protocol has overhead factor $O(n)$ since every party needs to store all data, while a perfectly scalable blockchain system has a constant space overhead factor.

To define the space overhead factor we need to introduce the notion of average-case analysis. Typically, sharding protocols scale well when the analysis is optimistic, in essence for transaction inputs that neither contain cross-shard nor multi-input (multi-output) transactions. However, in practice transactions are both cross-shard and multi-input/output. For this reason, we define the space overhead factor as a random variable dependent on an input set of transactions T drawn uniformly at random from a distribution D_T.

We assume T is given well in advance as input to all parties. To be specific,

we assume every transaction $tx \in T$ is given at least for u consecutive rounds to all parties of the system. Hence, from the liveness property, all transaction ledgers held by honest parties will report all transactions in T as stable. Further, we denote by $L^{\lceil k}$ the vector L where the last k positions are "pruned", while $|L^{\lceil k}|$ denotes the number of transactions contained in this "pruned" ledger. We note that a similar notation holds for a chain C where the last k positions map to the last k blocks. Each party P_j, maintains a collection of ledgers $SL_j = \{L_1, L_2, \ldots, L_s\}, 1 \leq s \leq m$. We define the space overhead factor for a sharding protocol with input T as the number of stable transactions included in every party's collection of transaction ledgers over the number of transactions given as input in the system[5],

$$\omega_s(T) = \sum_{\forall j \in [n]} \sum_{\forall L \in SL_j} |L^{\lceil k}|/|T|$$

Apart from space and communication complexity, we also consider the verification process, which can be computationally expensive. In our model, we assume the computational cost is constant per verification. Every time a party checks if a transaction is invalid or conflicting with a ledger, the running time is considered constant since this process can always speed up using efficient data structures (e. g. trees allow for logarithmic lookup time). Therefore, the computational power spent by a party for verification is defined by the number of times the party executes the verification process. For this purpose, we employ a verification oracle V. Each party calls the oracle to verify transactions, pending or included in a block. We denote by q_i the number of times party P_i calls oracle V during the protocol execution. We now define a *computational overhead factor* ω_c that reflects the total number of times all parties call the verification oracle during the protocol execution scaled over the number of input transactions,

$$\omega_c(T) = \sum_{\forall i \in [n]} q_i/|T|$$

Note that similarly to the space overhead factor, the computational overhead factor is also a random variable. Hence, the objective is to calculate the expected value of ω_c, i. e., the probability-weighted average of all possible values, where the probability is taken over the input transactions T.

In an ideally designed protocol, communication, space, and computational overhead factors express the same amount of information. However, there can be poorly designed protocols that reduce the overhead in only one dimension but fail to limit the overhead in the entire system. For this reason, we define the scaling factor of a protocol to be the total number of participants over the maximum of the three overhead factors,

$$\Sigma = \frac{n}{\max(\omega_m, \omega_s, \omega_c)}$$

[5]Without loss of generality, we assume all transactions are valid and thus are eventually included in all honest parties' ledgers.

Intuitively if all the overhead factors are constant, then every node has a constant overhead. Hence, the system can scale linearly without imposing any additional overhead to the parties. Furthermore, a sharding system can consist of multiple protocols; the scaling factor of the sharding system is therefore defined as the the number of participants divided by the maximum overhead factor of all the system's protocols. We can now define the scalability property of sharded distributed ledger protocols as follows.

Definition 3.4 (Scalability)**.** *Parameterized by n (number of participants), $v \in \mathbb{N}$ (average size of transactions), D_T (distribution of the input set of transactions), any sharding blockchain protocol that scales requires a scaling factor $\Sigma = \omega(1)$.*

In order to adhere to standard security proofs from now on we say that the protocol Π *satisfies* property Q in our model if Q holds with overwhelming probability (in a security parameter). Note that a probability p is overwhelming if $1 - p$ is negligible. A function $negl(k)$ is negligible if for every $c > 0$, there exists an $N > 0$ such that $negl(k) < 1/k^c$ for all $k >\geq N$. Furthermore, we denote by $\mathbb{E}(\cdot)$ the expected value of a random variable.

Definition 3.5 (Robust Sharded Transaction Ledger)**.** *A protocol that satisfies the properties of persistence, consistency, liveness, and scalability maintains a robust sharded transaction ledger.*

3.2.3 (Sharding) Blockchain Protocols

In this section, we adopt the definitions and properties of [13] for blockchain protocols, while we slightly change the notation to fit our model. In particular, we assume the parties of a shard of any sharding protocol maintain a chain (ledger) to achieve consensus. This means that every shard internally executes a blockchain (consensus) protocol that has three properties as explained in Chapter 2: chain growth, chain quality, and common prefix. Each consensus protocol satisfies these properties with different parameters.

In this work, we will use the properties of the shards' consensus protocol to prove that a sharding protocol maintains a robust sharded transaction ledger. In addition, we will specifically use the shard growth and shard quality parameters to estimate the transaction throughput of a sharding protocol. The following definitions follow closely Definitions 3, 4 and 5 of [13] (see Section 2.2).

Definition 3.6 (Shard Growth Property)**.** *Parametrized by $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$, for any honest party P with chain C, it holds that for any s rounds there are at least $\tau \cdot s$ blocks added to chain C of P.*

Definition 3.7 (Shard Quality Property)**.** *Parametrized by $\mu \in \mathbb{R}$ and $l \in \mathbb{N}$, for any honest party P with chain C, it holds that for any l consecutive blocks of C the ratio of honest blocks in C is at least μ.*

Definition 3.8 (Common Prefix Property)**.** *Parametrized by $k \in \mathbb{N}$, for any pair of honest parties P_1, P_2 adopting chains C_1, C_2 (in the same shard) at rounds $r_1 \leq r_2$ respectively, it holds that $C_1^{\lceil k} \preceq C_2$, where $\preceq$ denotes the prefix relation.*

Next, we define the *degree of parallelism* (DoP) of a sharding protocol, denoted m'. To evaluate the DoP of a protocol with input T, we need to determine how many shards are affected by each transaction on average; essentially, estimate how many times we run consensus for each valid transaction until it is stable. This is determined by the mechanism that handles the cross-shard transactions. To that end, we define $m_{i,j} = 1$ if the j-th transaction of set T has either an input or an output that is assigned to the i-th shard; otherwise $m_{i,j} = 0$. Then, the DoP of a protocol execution over a set of transactions T is defined as follows: $m' = \frac{T \cdot m}{\sum_{j=1}^{T} \sum_{i=1}^{m} m_{i,j}}$. The DoP of a protocol execution depends on the distribution of transactions D_T, the average size of transactions v, and the number of shards m. For instance, assuming a uniform distribution D_T, the expected DoP is $\mathbb{E}(m') = m/v$.

We can now define an efficiency metric, the *transaction throughput* of a sharding protocol. Considering constant block size, we define the transaction throughput as follows:

Definition 3.9 (Throughput)**.** *The expected transaction throughput in s rounds of a sharding protocol with m shards is $\mu \cdot \tau \cdot s \cdot m'$. We define the throughput factor of a sharding protocol $\sigma = \mu \cdot \tau \cdot m'$.*

Intuitively, the throughput factor expresses the average number of blocks that can be processed per round by a sharding protocol. Thus, the transaction throughput (per round) can be determined by the block size multiplied by the throughput factor. The block size is considered constant, however, it cannot be arbitrarily large. The limit on the block size is determined by the bandwidth of the "slowest" party within each shard. At the same time, the constant block size guarantees low latency. If the block size is very large or depends on the number of shards or the number of participants, *bandwidth or latency* becomes the performance bottleneck. Note that our goal is to estimate the efficiency of the parallelism of transactions in the protocol, thus other factors like cross-shard communication latency are omitted.

3.3 Limitations of Sharding Protocols

In this section, we explore the limits of sharding protocols. First, in Section 3.3.1, we focus on the limitations that stem from the nature of the transaction workload. In particular, sharding protocols are affected by two characteristics of the input transaction set: the number of inputs and outputs of each transaction or transaction size v, and more importantly the number of cross-shard transactions.

The number of inputs and outputs (UTXOs) of each transaction is in practice fairly small. For example, an average Bitcoin transaction has 2 inputs and 3 outputs with a small deviation (e. g. 1) [88]. Hence, we consider the number of UTXOs participating in each transaction fixed, thus the transaction size v is a small constant. Furthermore, the largest the transaction size, the more shards are affected by each transaction on expectation, hence the largest the ratio of cross-shard transactions. Thus, to meaningfully upper bound the ratio of cross-shard transactions, we consider the minimum transaction size $v = 2$.

The number of cross-shard transactions depends on the distribution of the input transactions D_T, as well as the process that partitions transactions to shards. First, we assume each ledger interacts (i. e., has a cross-shard transaction) with γ other ledgers on average, where γ is a function dependent on the number of shards m. We consider protocols that require parties to maintain information on shards other than their own and derive an upper bound for the expected value of γ such that scalability holds. Then, we prove there is no protocol that maintains a robust sharded ledger against an adaptive adversary.

Later, in Section 3.3.2, we assume shards are created with a uniformly random process; essentially the UTXO space is partitioned uniformly at random into shards. Under this assumption (which most sharding systems make), we show a constant fraction of the transactions are expected to be cross-shard, and there is no sharded ledger that satisfies scalability if participants have to maintain any information on ledgers others than their own. Note that our results hold for *any distribution* where the number of cross-shard transactions is (on expectation) *proportional* to the number of shards.

Last, in Section 3.3.3, we examine the case where parties are assigned to shards independently and uniformly at random. This assumption is met by almost all known sharding systems to guarantee security against non-static adversaries. We show that any sharding protocol can scale at most by a factor of $n/\log n$ in our model. Finally, we demonstrate the importance of periodical compaction of the valid state-updates in sharding protocols; we prove that any sharding protocol that satisfies scalability in our security model requires a state-compaction process such as checkpoints [9], cryptographic accumulators [81], zero-knowledge proofs [82], non-interactive proofs of proofs-of-work [84, 85], proof of necessary work [86], erasure codes [87].

3.3.1 General Bounds

First, we prove there is no robust sharded transaction ledger that has a constant number of shards. Then, we show that there is no protocol that maintains a robust sharded transaction ledger against an adaptive adversary.

Lemma 3.10. *In any robust sharded transaction ledger the number of shards (parametrized by n) is $m = \omega(1)$.*

Proof. Suppose there is a protocol that maintains a constant number m of sharded ledgers, denoted by $x_1, x_2, \ldots, x_m$. Let n denote the number of parties and T the number of transactions to be processed (wlog assumed to be valid). A transaction is processed only if it is stable, i. e., is included deep enough in a ledger (k blocks from the end of the ledger where k a security parameter). Each ledger will include T/m transactions on expectation. Now suppose each party participates in only one ledger (best case), thus broadcasts, verifies and stores the transactions of that ledger only. Hence, every party stores T/m transactions on expectation. The expected space overhead factor is $\omega_s = \sum_{\forall i \in [n]} \sum_{\forall x \in L_i} |x^{\lceil k}|/|T| = \sum_{\forall x \in L_i} \frac{T}{mT} = \frac{n}{m} = \Theta(n)$. Thus, $\Sigma = \Theta(1)$ and scalability is not satisfied. □

Suppose a party is participating in shard x_i. If the party maintains information (e. g. the headers of the chain for verification purposes) on the chain of shard x_j, we say that the party is a *light node* for shard x_j. In particular, a light node for shard x_j maintains information at least proportional to the length of the shard's chain x_j. Sublinear light clients [84,85] verifiably compact the shard's state, thus are not considered as light nodes but are discussed later.

Lemma 3.11. *For any robust sharded transaction ledger that requires every participant to be a light node for all the shards affected by cross-shard transactions, it holds* $\mathbb{E}(\gamma) = o(m)$.

Proof. We assumed that every ledger interacts on average with γ different ledgers, i. e., the cross-shard transactions involve γ many different shards on expectation. The block size is considered constant, meaning each block includes at most e transactions where e is constant. Thus, each party maintaining a ledger and being a light node to γ other ledgers must store on expectation $(1 + \frac{\gamma}{e})\frac{T}{m}$ information. Hence, the expected space overhead factor is

$$\mathbb{E}(\omega_s) = \sum_{\forall i \in [n]} \sum_{\forall x \in L_i} |x^{\lceil k}|/|T| = n\frac{(1 + \frac{\gamma}{e})\frac{T}{m}}{T} = \Theta\Big(\frac{\gamma n}{m}\Big)$$

where the second equation holds due to linearity of expectation. To satisfy scalability, we demand $\Sigma = \omega(1)$, hence $\mathbb{E}(\omega_s) = o(n)$, thus $\gamma = o(m)$. □

Next, we show that there is no protocol that maintains a robust transaction ledger against an adaptive adversary in our model. We highlight that our result holds because we assume *any node is corruptible* by the adversary. If we assume more restrictive corruption sets, e. g. each shard has at least one honest well-connected node, sharding against an adaptive adversary may be possible if we employ other tools, such as fraud and data availability proofs [58].

Theorem 3.12. *There is no protocol maintaining a robust sharded transaction ledger against an adaptive adversary with* $f \geq n/m$.

Proof. (Towards contradiction) Suppose there exists a protocol Π that maintains a robust sharded ledger against an adaptive adversary that corrupts $f = n/m$ parties. From the pigeonhole principle, there exists at least one shard x_i with at most n/m parties (independent of how shards are created). The adversary is adaptive, hence at any round can corrupt all parties of shard x_i. In a malicious shard, the adversary can perform arbitrary operations, thus can spend the same UTXO in multiple cross-shard transactions. However, for a cross-shard transaction to be executed it needs to be accepted by the output shard, which is honest. Now, suppose Π allows the parties of each shard to verify the ledger of another shard. For Lemma 3.11 to hold, the verification process can affect at most $o(m)$ shards. Note that even a probabilistic verification, i. e., randomly select some transactions to verify, can fail due to storage requirements and the fact that the adversary can perform arbitrarily many attacks. Therefore, for each shard, there are at least 2 different shards that do not verify the cross-shard transactions. Thus, the adversary can simply attempt to double-spend the same UTXO across every shard and will succeed in the shards that do not verify the validity of the cross-shard transaction. Hence, consistency is not satisfied. □

3.3.2 Bounds under Uniform Shard Creation

In this section, we assume that the creation of shards is UTXO-dependent; transactions are assigned to shards independently and uniformly at random. This assumption is in sync with the proposed protocols in the literature. In a non-randomized process of creating shards, the adversary can precompute and thus bias the process in a permissionless system. Hence, all sharding proposals employ a random process for shard creation. Furthermore, all shards process approximately the same amount of transactions; otherwise the efficiency of the protocol would depend on the shard that processes most transactions. For this reason, we assume the UTXO space is partitioned to shards uniformly at random. Note that we consider UTXOs to be random strings.

Under this assumption, we prove a constant fraction of transactions are cross-shard on expectation. As a result, we show there is no sharding protocol that maintains a robust sharded ledger when participants act as light clients on all shards involved in cross-shard transactions. Our observations hold for any transaction distribution D_T that results in a constant fraction of cross-shard transactions.

Lemma 3.13. *The expected number of cross-shard transactions is* $\Theta(|T|)$.

Proof. Let Y_i be the random variable that shows if a transaction is cross-shard; $Y_i = 1$ if $tx_i \in T$ is cross-shard, and 0 otherwise. Since UTXOs are assigned to shards uniformly at random, $Pr[i \in x_k] = \frac{1}{m}$, for all $i \in v$ and $k \in [m] = \{1, 2, \ldots, m\}$. The probability that all UTXOs in a transaction $tx \in T$ belong

to the same shard is $\frac{1}{m^{v-1}}$ (where v is the cardinality of UTXOs in tx). Hence, $Pr[Y_i = 1] = 1 - \frac{1}{m^{v-1}}$. Thus, the expected number of cross-shard transactions is $\mathbb{E}(\sum_{\forall tx_i \in T} Y_i) = |T|\big(1 - \frac{1}{m^{v-1}}\big)$. Since, $m(n) = \omega(1)$ (Lemma 3.10) and v constant, the expected cross-shard transactions converges to T for n sufficiently large. □

Lemma 3.14. *For any protocol that maintains a robust sharded transaction ledger, it holds* $\gamma = \Theta(m)$.

Proof. We assume each transaction has a single input and output, hence $v = 2$. This is the worst-case input for evaluating how many shards interact per transaction; if $v >> 2$ then each transaction would most probably involve more than two shards and thus each shard would interact with more different shards for same set on transactions.

For $v = 2$, we can reformulate the problem as a graph problem. Suppose we have a random graph G with m nodes, each representing a shard. Now let an edge between nodes u, w represent a transaction between shards u, w. Note that in this setting we allow self-loops, which represent the intra-shard transactions. We create the graph G with the following random process: We choose an edge independently and uniformly at random from the set of all possible edges including self-loops, denoted by E'. We repeat the process independently $|T|$ times, i. e., as many times as the cardinality of the transaction set. We note that each trial is independent and the edges chosen uniformly at random due to the corresponding assumptions concerning the transaction set and the shard creation. We now will show that the average degree of the graph is $\Theta(m)$, which immediately implies the statement of the lemma.

Let the random variable Y_i represent the existence of edge i in the graph, i. e., $Y_i = 1$ if edge i was created at any of the T trials, 0 otherwise. The set of all possible edges in the graph is E, $|E| = \binom{m}{2} = \frac{m(m-1)}{2}$. Note that this is not the same as set E' which includes self-loops and thus $|E'| = \binom{m}{2} + m = \frac{m(m+1)}{2}$. For any vertex u of G, it holds

$$\mathbb{E}[deg(u)] = \frac{2\mathbb{E}[\sum_{\forall i \in E} Y_i]}{m}$$

where $deg(u)$ denotes the degree of node u. We have,

$$Pr[Y_i = 1] = 1 - Pr[Y_i = 0] =$$

$$1 - Pr[Y_i = 0 \text{ at trial 1}]Pr[Y_i = 0 \text{ at trial 2}] \dots$$

$$Pr[Y_i = 0 \text{ at trial T}] = 1 - \Big(1 - \frac{2}{m(m+1)}\Big)^{|T|}$$

Thus,

$$\mathbb{E}[deg(u)] = \frac{2m(m-1)}{2}\Big[1 - \Big(1 - \frac{2}{m(m+1)}\Big)^{|T|}\Big]$$

$$= (m-1)\Big[1 - \Big(1 - \frac{2}{m(m+1)}\Big)^{|T|}\Big]$$

Therefore, for many transactions we have $|T| = \omega(m^2)$ and consequently $\mathbb{E}[deg(u)] = \Theta(m)$. □

Theorem 3.15. *There is no protocol that maintains a robust sharded transaction ledger when participants are light nodes on the shards involved in cross-shard transactions.*

Proof. Immediately follows from Lemmas 3.11 and 3.14. □

3.3.3 Bounds under Random Party to Shard Assignment

In this section, we assume parties are assigned to shards uniformly at random. Any other shard assignment strategy yields equivalent or worse guarantees since we have no knowledge on which parties are byzantine. Our goal is to upper bound the number of shards for a protocol that maintains a robust sharded transaction ledger in our security model. To satisfy the security properties, we demand each shard to contain at least a constant fraction of honest parties $1 - a$ ($< 1 - \frac{f}{n}$). This is due to known limitations of consensus protocols [19].

The *size* of a shard is the number of the parties assigned to the shard. We say shards are *balanced* if all shards have approximately the same size. We denote by $p = f/n$ the (constant) fraction of the byzantine parties. A shard is *a-honest* if at least a fraction of $1 - a$ parties in the shard are honest.

Lemma 3.16. *Given n parties are assigned uniformly at random to shards and the adversary corrupts at most $f = pn$ parties (p constant), all shards are a-honest (a constant) with probability $\frac{1}{n^c}$ only if the number of shards is at most $m = \frac{n}{c' \ln n}$, where $c' \geq c\frac{2 + a - p}{(a-p)^2}$.*

Proof. To prove the lemma, it is enough to show that if we only assign the $f = pn$ byzantine parties independently and uniformly at random to the m shards, then all shards will have less than $a\frac{n}{m}$ parties.

From this point on, let n denote the byzantine parties. We will show that for $m = \frac{n}{c' \ln n}$ and a carefully selected constant c', all shards will have less than $(1 + (a - p))\frac{n}{m}$ parties.

Let Y_i denote the size of shard i. Then, $\mathbb{E}[Y_i] = \frac{n}{m}$. By the Chernoff bound, we have

$$Pr\Big[Y_i \geq \big(1 + (a-p)\big)\frac{n}{m}\Big] \leq e^{\frac{(a-p)^2}{2+a-p}}\frac{n}{m}$$

The probability that all shards have less than $(1 + (a-p))\frac{n}{m}$ parties is given by the union bound on the number of shards,

$$p = 1 - \Big(Pr\Big[Y_1 \geq \big(1 + (a-p)\big)\frac{n}{m}\Big] + Pr\Big[Y_2 \geq$$

$$\big(1 + (a-p)\big)\frac{n}{m}\Big] + \cdots + Pr\Big[Y_m \geq \big(1 + (a-p)\big)\frac{n}{m}\Big]\Big)$$

$$\Rightarrow p \geq 1 - me^{\frac{(a-p)^2}{2+a-p}}\frac{n}{m} \geq 1 - \frac{1}{n^c}, c > 1$$

The last inequality holds for $m \leq \frac{n}{c' \ln n}$, where $c' \geq c\frac{2+a-p}{(a-p)^2}$, in which case all shards are a-honest with high probability $\frac{1}{n^c}$. □

Corollary 3.17. *A sharding protocol maintains a robust sharded transaction ledger against an adversary with $f = pn$, only if $m = \frac{n}{c' \ln n}$, where $c' \geq c\frac{5/2-p}{(1/2-p)^2}$.*

Proof. To maintain the security properties, all shards must be a-honest, where a depends on the underlying consensus protocol. To the best of our knowledge, all byzantine-fault tolerant consensus protocols require at least honest majority, hence $a \leq 1/2$. From Lemma 3.16, $m = \frac{n}{c' \ln n}$, where $c' \geq c\frac{5/2-p}{(1/2-p)^2}$. □

Next, we prove that any sharding protocol may scale at most by an $n/\log n$ factor. This bound refers to independent nodes. If, for instance, we shard per authority, with all authorities represented in each shard, the bound of the theorem does not hold and the actual system cannot be considered sharded since every authority holds all the data.

Theorem 3.18. *Any protocol that maintains a robust sharded transaction ledger in our security model has scaling factor at most $\Sigma = O(\frac{n}{\log n})$.*

Proof. In our security model, the adversary can corrupt $f = pn$ parties, p constant. Hence, from Corollary 3.17, $m = O(\frac{n}{\log n})$. Each party stores at least T/m transactions on average and thus the expected space overhead factor is $\omega_s \geq n\frac{T/m}{T} = \frac{n}{m}$. Therefore, the scaling factor is at most $m = O(\frac{n}{\log n})$. □

Next, we show that any sharding protocol that satisfies scalability requires some process of *verifiable compaction of state*, such as checkpoints [9], cryptographic accumulators [81], zero-knowledge proofs [82], non-interactive proofs of proofs-of-work [84,85], proof of necessary work [86], erasure codes [87]. Such a process allows the state of the distributed ledger (e. g., stable transactions) to be compressed

significantly while users can verify the correctness of the state. Intuitively, in any sharding protocol secure against a slowly adaptive adversary parties must periodically shuffle in shards. To verify new transactions the parties must receive a verifiably correct UTXO pool for the new shard without downloading the full shard history; otherwise the communication overhead of the bootstrapping process eventually exceeds that of a non-sharded blockchain. Although existing evaluations typically ignore this aspect with respect to bandwidth, we stress its importance in the long-term operation: *the bootstrap cost will eventually become the bottleneck due to the need for nodes to regularly shuffle.*

Theorem 3.19. *Any protocol that maintains a robust sharded transaction ledger in our security model employs verifiable compaction of the state.*

Proof. (Towards contradiction) Suppose there is a protocol that maintains a robust sharded ledger without employing any process that verifiably compacts the blockchain. To guarantee security against a slowly-adaptive adversary, the parties change shards at the end of each epoch. At the beginning of each epoch, the parties must process a new set of transactions. To check the validity of this new set of transactions, each (honest) shard member downloads and maintains the corresponding ledger. Note that even if the party only maintains the hash-chain of a ledger, the cost is equivalent to maintaining the list of transactions given that the block size is constant. We will show that the communication overhead factor increases with time, eventually exceeding that of a non-sharded blockchain; thus scalability is not satisfied from that point on.

In each epoch transition, a party changes shards with probability $1 - 1/m$, where m is the number of shards. As a result, a party changing a shard in epoch k must download the shard's ledger of size $\frac{k \cdot T}{m}$. Therefore, the expected communication overhead factor of bootstrapping during the k-th epoch transition is $\frac{k \cdot T}{m} \cdot (1 - \frac{1}{m})$. We observe the communication overhead grows with the number of epochs k, hence it will eventually become the scaling bottleneck. For instance, for $k > m \cdot n$, the communication factor is greater than linear to the number of parties in the system n, thus the protocol does not satisfy scalability. □

Theorem 3.19 holds even if parties are not assigned to shards uniformly at random but follow some other shuffling strategy like in [89]. *As long as a significant fraction of honest parties change shards from epoch to epoch, verifiable compaction of state is necessary* to restrict the bandwidth requirements during bootstrapping in order to satisfy scalability.

3.4 The Sharding Crux

In this section, we first discuss our design rationale for robust sharding. Using the bounds provided in Section 3.3, we deduce some sufficient components for robust sharding in our model. We then introduce a *protocol abstraction* for robust sharding, termed *the sharding crux*, employing the introduced sharding components. We prove our protocol abstraction to be secure in our model (assuming the components are secure) and evaluate its efficiency with respect to the choices of the individual components.

3.4.1 Sharding Components

We explain our design rationale and introduce the ingredients of a protocol that maintains a robust sharded ledger.

(a) **Consensus protocol of shards or `Consensus`:** A sharding protocol either runs consensus in every shard separately (multi-consensus) or provides a single total ordering for all the blocks generated in each shard (uni-consensus). Since uni-consensus takes polynomial cost per block, such a protocol can only scale if the block size is also polynomial (e.g., includes $\Omega(n)$ transactions). However, in such a case, the resources of each node generating an $\Omega(n)$-sized block must also grow with n, and therefore scalability would not be satisfied. Hence, in our protocol abstraction we chose a multi-consensus approach.

The consensus protocol run per shard must satisfy the properties defined by Garay et al. [13], namely common prefix, chain quality and chain growth. These properties are necessary (but not sufficient) to ensure the persistence, liveness, and consistency of the sharding system.

(b) **Cross-shard mechanism or `CrossShard`:** The cross-shard transacting mechanism is the protocol that handles the transactions that span across multiple shards. The cross-shard mechanism is critical to the security of the sharding system as it guarantees consistency. Furthermore, it is also crucial for scalability as a naively designed cross-shard mechanism may induce high storage or communication overhead on the nodes when handling several cross-shard transactions. To that end, the limitations from Section 3.3 apply for any protocol designed to maintain a robust sharded transaction ledger. The cross-shard transaction mechanism should provide the ACID properties (just like in database transactions). Durability and Isolation are provided directly by the blockchains of the shards, hence, the cross-shard transaction protocol should provide Consistency (every transaction that commits produces a semantically valid state) and Atomicity (transactions are committed and aborted atomically - all or nothing). Typically the cross-shard protocol runs on top of the consensus of the shards, meaning that it invokes the consensus protocol to guarantee Consistency.

(c) **Sybil-resistance mechanism or `Sybil`:** Necessary to the global consensus on the new set of Sybil resistant identities in a permissionless setting, which guarantees the security bounds of the consensus protocol (e. g., $f < 1/3$ for BFT). Given the slowly adaptive adversary, the Sybil-resistance mechanism needs to depend on the unknown randomness of the previous epoch. The exact protocol (PoW, PoS, etc.) is irrelevant.

(d) **Division of nodes to shards or `Divide2Shards`:** From Theorem 3.12 we know that no system can shard the state securely if the adversary is fully adaptive. Therefore, we design our protocol abstraction in a slowly-adaptive adversarial model – static adversaries are an easier subcase. To guarantee the security properties against a slowly adaptive adversary, nodes must be divided to shards appropriately.

To satisfy transaction finality (i. e., liveness and persistence), either the consensus security bounds must hold for each shard, or the protocol must guarantee with high probability that if the adversary compromises a shard then the security violation will be restored within a specific (small) number of rounds. To be specific, if an adversary completely or partially compromises a shard, effectively violating the consensus bounds, then the adversary can double spend within the shard (violates persistence), as well as across shards (because nodes cannot verify cross-shard transactions from Lemma 3.11). Therefore, the transactions included in these blocks can only be executed when honest participants have verified them. Partial solutions towards this direction have been proposed such as fraud proofs that allow an honest party to later prove a misbehavior. Another problem with this approach is the need to guarantee data availability.

Due to the complexity of such solutions and their implications on the transactions' finality, we design our protocol assuming the security bounds of consensus are maintained when nodes are divided into shards. Specifically, we assume that at the beginning of each epoch, the nodes are shuffled among the shards to guarantee the consensus bounds. A secure shuffling process typically requires a source of randomness (i. e., see the DRG protocol below). When assigned to a shard, nodes need to update their local state with the state of the new shard they are asked to secure.

(e) **Randomness generation protocol or `DRG`:** Necessary for any setting and should provide unbiasable randomness [50, 54–56, 90–92] such that both the Sybil-resistance mechanism and the protocol that divides nodes to shards result in shards that maintain the security bounds for the consensus protocol. Due to the slowly-adaptive adversary assumption, the DRG protocol must be executed (at least) once per epoch, while its high communication complexity can be amortized over the rounds of an epoch such that the system remains scalable.

(f) **Verifiable compaction of state or `CompactState`:** At the end of each epoch, the history of the shards' ledgers should be summarized such that new

parties can bootstrap with minimal effort (Theorem 3.19). To this end, a process for verifiable compaction of the state-updates must be employed. Then, the compacted state must be broadcast to all the nodes in the network via reliable broadcast [30] to ensure data integrity and data availability to the node that will be assigned to each shard in the next epoch. Note that any protocol that ensures data binding and data availability can be used. This step must take place before the new epoch begins as the slowly adaptive adversary is allowed to compromise the nodes of a shard in the previous epoch, violating the liveness of the system.

For the purpose of our analysis, we consider that the protocol that partitions the state (e. g., transaction space) into shards yields the worst possible outcome. In other words, we consider all transactions to be cross-shard, because any secure protocol that can perform well in the most pessimistic case, can obviously also perform well where transactions are intra-shard. We further note that designing a secure sharding system when most transactions are intra-shard is trivial, hence not the study of this work. Last, we assume that all protocols satisfy liveness in our security model.

3.4.2 Protocol Abstraction

In this section, we introduce a protocol abstraction that achieves robust sharding in our model. In the protocol design we employ the components of sharding as black boxes, and later prove security and analyze the performance of the protocol depending on the choice of the components.

3.4.3 Analysis

We show that the sharding crux abstraction is secure in our model (i. e., satisfies persistence, consistency, and liveness), while its efficiency (i. e., scalability and throughput factor) depends on the chosen subprotocols.

Theorem 1. The sharding crux satisfies persistence in our system model assuming at most f byzantine nodes.

Proof. Assuming `Divide2Shard` respects the security bounds of `Consensus`, the common prefix property is satisfied in each shard, so persistence is satisfied. □

Theorem 2. The sharding crux satisfies consistency in our system model assuming at most f byzantine nodes.

Proof. Transactions can either be intra-shard (all UTXOs within a single shard) or cross-shard. Consistency is satisfied for intra-shard transactions as long as `Divide2Shard` respects the security bounds of `Consensus`, hence the common prefix property is satisfied. Furthermore, consistency is satisfied for cross-shard

Protocol Abstraction 1: The Sharding Crux

Data: N_0 nodes are participating in the system at round 0 (genesis block). $m(N_E)$ denotes the function that determines the number of shards in epoch E. The transactions of epoch E are T_E. i denotes the block round (its relation to the communication rounds depends on the employed components).

Result: Shard state $T = \{T_0, T_1, \dots\}$.

```
/* Initialization                                                    */
i ← 1
E ← 0
/* Beginning of epoch: retrieve identities from Sybil resistant protocol,
   execute the DRG protocol to create the new epoch randomness, and
   assign nodes to shards                                           */
if i mod R = 1 :
    E ← E + 1
    if i ≠ 1 :
        N_E ← Sybil(r_{E-1})
    r_E ← DRG(N_E)
    Call Divide2Shards(N_E, m(N_E), r_E)
/* End of epoch: compact the state of the shard                     */
elif i mod R = 0 :
    Call CompactState(i)
/* During epoch: run the consensus protocol for intra-shard and
   cross-shard transactions                                         */
else:
    if If transaction t ∈ T_E is cross-shard :
        Call CrossShard(t) ;      // Invokes Consensus in multiple shards
    else:
        Call Consensus(t)
i ← i + 1
Go to step 3
```

transactions from the `CrossShard` protocol as long as it correctly provides atomicity (provided by assumption). □

Theorem 3. The sharding crux satisfies liveness in our system model assuming at most f byzantine nodes.

Proof. Follows from the assumption that all subprotocols satisfy liveness , as well as the `CompactState` protocol that ensures data availability between epochs. □

Scalability. The scaling factor of sharding crux depends on the maximum overhead factor of all the components it employs. The maximum overhead factor

for `DRG, Divide2Shards, Sybil`, and `CompactState` can be amortized over the rounds of an epoch because these protocols are executed once each epoch. Thus, the size of an epoch is critical for scalability. Intuitively, this implies that *if the size of the epoch is small, hence the adversary highly-adaptive, sharding is not that beneficial as the protocols that are executed on the epoch transaction are as resource demanding as the consensus in a non-sharded system.*

Throughput factor. Similarly to scalability, the throughput factor also depends on the employed subprotocols, and in particular, `Consensus` and `CrossShards`. To be specific, the throughput factor depends on the shard growth and shard quality parameters which are determined by `Consensus`. In addition, given a transaction input, the degree of parallelism, which is the last component of the throughput factor, is determined by the maximum number of shards possible and the way cross-shard transactions are handled. The maximum number of shards depends on `Consensus` and `Divide2Shards`, while `CrossShard` determines how many shards are affected by a single transaction. For instance, if the transactions are divided in shards uniformly at random, the sharding crux can scale at most by $n/\log n$ as stated in Corollary 3.17. We further note that the minimum number of affected shards for a specific transaction is the number of UTXOs that map to different shards; otherwise security cannot be guaranteed.

We demonstrate how to calculate the scaling factor and the throughput factor for OmniLedger and RapidChain in the following section.

3.5 Evaluation of Existing Protocols

In this section, we evaluate existing sharding protocols with respect to the desired properties defined in Section 3.2.2. A summary of our evaluation can be found in Table 3.2 in Section 3.5.6.

The analysis is conducted in the synchronous model and thus any details regarding performance on periods of asynchrony are discarded. The same holds for other practical refinements that do not asymptotically improve the protocols' performance.

3.5.1 Elastico

Overview

Elastico is the first distributed blockchain sharding protocol introduced by Luu at al. [7]. The protocol lies in the intersection of traditional BFT protocols and the Nakamoto consensus. The protocol is synchronous and proceeds in epochs. The setting is permissionless, and during each epoch the participants create

valid identities by producing proof-of-work (PoW) solutions. The adversary is slowly-adaptive (as described in Section 3.2) and controls at most $f < \frac{n}{4}$ (n valid identities in total) or 25% of the computational power of the system.

At the beginning of each epoch, participants are partitioned into small shards (committees) of constant size c. The number of shards is $m = 2^s$, where s is a small constant such that $n = c \cdot 2^s$. A shard member contacts its directory committee to identify the other members of the same shard. For each party, the directory committee consists of the first c identities created in the epoch in the party's local view. Transactions are randomly partitioned in disjoint sets based on the hash of the transaction input (in the UTXO model); hence each shard only processes a fraction of the total transactions in the system. The shard members execute a BFT protocol to validate the shard's transactions and then send the validated transactions to the final committee. The final committee consists of all members with a fixed s-bit shards identity, and is in charge of two operations: (i) computing and broadcasting the final block, which is a digital signature on the union of all valid received transactions[6] (via executing a BFT protocol), and (ii) generating and broadcasting a bounded exponential biased random string which is used as a public source of randomness in the next epoch (e. g. for the PoW).

`Consensus:` Elastico does not specify the consensus protocol, but instead can employ any standard BFT protocol, like PBFT [93].

`CrossShard:` Each transaction is assigned to a shard according to the hash of the transaction's inputs. Every party maintains the entire blockchain, thus each shard can validate the assigned transaction independently, i. e., there are no cross-shard transactions. Note that Elastico assumes that transactions have a single input and output, which is not the case in cryptocurrencies as discussed in Section 3.3. To generalize Elastico's transaction assignment method to multiple inputs, we assume each transaction is assigned to the shard corresponding to the hash of all its inputs. Otherwise, if each input is assigned to a different shard according to its hash value, an additional protocol is required to guarantee the atomicity of transactions and hence the security (consistency) of Elastico.

`Sybil:` Participants create valid identities by producing PoW solutions using the randomness of the previous epoch.

`Divide2Shards & CompactState:` The protocol assigns each identity to a random shard in 2^s, identified by an s-bit shard identity. At the end of each epoch,

[6] The final committee in Elastico broadcasts only the Merkle root for each block. However, this is asymptotically equivalent to including all transactions since the block size is constant. Furthermore, the final committee does not check if the received transactions are conflicting, but merely verifies the presence of signatures.

the final committee broadcasts the final block that contains the Merkle hash root of every block of all shards' block. The final block is stored by all parties in the system. Hence, when the parties are re-assigned to new shards they already have the hash-chain to confirm the shard ledger and future transactions. Essentially, an epoch in Elastico is equivalent to a block generation round.

`DRG:` In each epoch, the final committee (of size c) generates a set of random strings R via a commit-and-XOR protocol. First, all committee members generate an r-bit random string r_i and send the hash $h(r_i)$ to all other committee members. Then, the committee runs an interactive consistency protocol to agree on a single set of hash values S, which they include on the final block. Later, each (honest) committee member broadcasts its random string r_i to all parties in the network. Each party chooses and XORs $c/2+1$ random strings for which the corresponding hash exists in S. The output string is the party's randomness for the epoch. Note that $r > 2\lambda + c - \log(c)/2$, where λ is a security parameter.

Analysis

Elastico's threat model allows for adversaries that can drop or modify messages, and send different messages to honest parties, which is not allowed in our model. However, we show that even under a more restrictive adversarial model, Elastico fails to meet the desired sharding properties. Specifically, we prove Elastico does not satisfy *scalability* and *consistency*. From the security analysis of [7], it follows that Elastico satisfies persistence and liveness in our system model.

Theorem 4. Elastico does not satisfy consistency in our system model.

Proof. Suppose a party submits two valid transactions, one spending input x and another spending input x and input y. Note that the second is a single transaction with two inputs. In this case, the probability that both hashes (transactions), $H(x, y)$ and $H(x)$, land in the same shard is $1/m$. Hence, the probability of a successful double-spending in a set of T transactions is almost $1 - (1/m)^T$, which converges to 0 as T grows, for any value $m > 1$. However, $m > 1$ is necessary to satisfy scalability (Lemma 3.10). Therefore, there will be almost surely a round in which two parties report two conflicting transactions. Since the final committee does not verify the validity of transactions but only checks the appropriate signatures are present, consistency is not satisfied. □

Lemma 5. *The space overhead factor of Elastico is* $\omega_s = \Theta(n)$.

Proof. At the end of each epoch, the final committee broadcasts the final block to the entire network. All parties store the final block, hence all parties maintain the entire input set of transactions. Since the block size is considered constant,

maintaining the hash-chain for each shard is equivalent to maintaining all the shards' ledgers. It follows that $\omega_s = \Theta(n)$, regardless of the input set T. □

Theorem 6. Elastico does not satisfy scalability in our system model.

Proof. Immediately follows from Definition 3.4 and Lemma 5. □

3.5.2 Monoxide

Overview

Monoxide [8] is an asynchronous proof-of-work protocol, where the adversary controls at most 50% of the computational power of the system. The protocol uniformly partitions the space of user addresses into shards (zones) according to the first k bits. Every party is permanently assigned to a shard uniformly at random. Each shard employs the GHOST [45] consensus protocol.

Participants are either full-nodes that verify and maintain the transaction ledgers, or miners investing computational power to solve PoW puzzles for profit in addition to being full-nodes. Monoxide introduces a new mining algorithm, called Chu-ko-nu, that enables miners to mine in parallel for all shards. The Chu-ko-nu algorithm aims to distribute the hashing power to protect individual shards from an adversarial takeover. Successful miners include transactions in blocks. A block in Monoxide is divided into two parts: the chaining block that includes all metadata (Merkle root, nonce for PoW, etc.) creating the hash-chain, and the transaction-block that includes the list of transactions. All parties maintain the hash-chain of every shard in the system.

Furthermore, all parties maintain a distributed hash table for peer discovery and identifying parties in a specific shard. This way the parties of the same shard can identify each other and cross-shard transactions are sent directly to the destination shard. Cross-shard transactions are validated in the shard of the payer and verified from the shard of the payee via a relay transaction and the hash-chain of the payer's shard.

`Consensus:` The consensus protocol of each shard is GHOST [45]. GHOST is a DAG-based consensus protocol similar to Nakamoto consensus [1], but the consensus selection rule is the heaviest subtree instead of the longest chain.

`CrossShard:` An input shard is a shard that corresponds to the address of a sender of a transaction (payer) while an output shard one that corresponds to the address of a receiver of a transaction (payee). Each cross-shard transaction is processed in the input shard, where an additional relay transaction is created and included in a block. The relay transaction consists of all metadata needed to

verify the validity of the original transaction by only maintaining the hash-chain of a shard (i. e. for light nodes). The miner of the output shard verifies that the relay transaction is stable and then includes it in a block in the output shard. Note that in case of forks in the input shard, Monoxide invalidates the relay transactions and rewrites the affected transaction ledger to maintain consistency.

`Sybil:` In a typical PoW election scheme, the adversary can create many identities and target its computational power to specific shards to gain control over more than half of the shard's participants. In such a case, the security of the protocol fails (both persistence and consistency properties do not hold). To address this issue, Monoxide introduces a new mining algorithm, Chu-ko-nu, that allows parallel mining on all shards. Specifically, a miner can batch valid transactions from all shards and use the root of the Merkle tree of the list of chaining headers in the batch as input to the hash, alongside with the nonce (and some configuration data). Thus, when a miner successfully computes a hash lower than the target, the miner adds a block to every shard.

`Divide2Shards:` Parties are permanently assigned to shards uniformly at random according the first k bits of their address.

`DRG:` The protocol uses deterministic randomness (e. g. hash function) and does not require any random source.

`CompactState:` No compaction of state is used in Monoxide.

Analysis

We prove that Monoxide satisfies persistence, liveness, and consistency, but does not satisfy scalability. Note that Theorem 3.15 also implies that Monoxide does not satisfy scalability since Monoxide demands each party to verify cross-shard transactions by acting as a light node to all shards.

Theorem 7. Monoxide satisfies persistence and liveness in our system model for $f < n/4$.

Proof. From the analysis of Monoxide, it holds that if all honest miners follow the Chu-ko-nu mining algorithm, then honest majority within each shard holds with high probability for any adversary with $f < n/4$ (Section 5.3 [8]).

Assuming honest majority within shards, persistence depends on two factors: the probability a stable transaction becomes invalid in a shard's ledger, and the probability a cross-shard transaction is reverted after being confirmed. Both these factors solely depend on the common prefix property of the shards' consensus

mechanism. Monoxide employs GHOST as the consensus mechanism of each shard, hence the common prefix property is satisfied if we assume that invalidating the relay transaction does not affect other shards [94]. Suppose common prefix is satisfied with probability $1 - p$ (which is overwhelming on the "depth" security parameter k). Then, the probability none of the outputs of a transaction are invalidated is $(1 - p)^{(v-1)}$ (worst case where $v - 1$ outputs – relay transactions – link to one input). Thus, a transaction is valid in a shard's ledger after k blocks with probability $(1-p)^v$, which is overwhelming in k since v is considered constant. Therefore, persistence is satisfied.

Similarly, liveness is satisfied within each shard. Furthermore, this implies liveness is satisfied for cross-shard transactions. In particular, both the initiative and relay transactions will be eventually included in the shards' transaction ledgers, as long as chain quality and chain growth are guaranteed within each shard [94]. □

Theorem 8. Monoxide satisfies consistency in our system model for $f < n/4$.

Proof. The common prefix property is satisfied with high probability in GHOST [36]. Thus, intra-shard transactions satisfy consistency with high probability (on the "depth" security parameter). Furthermore, if a cross-shard transaction output is invalidated after its confirmation, Monoxide allows rewriting the affected transaction ledgers. Hence, consistency is restored in case of cross-transaction failure. Thus, overall, consistency is satisfied in Monoxide. □

Note that allowing to rewrite the transaction ledgers in case a relay transaction is invalidated strengthens the consistency property but weakens the persistence and liveness properties.

Intuitively, to satisfy persistence in a sharded PoW system, the adversarial power needs to be distributed across shards. To that end, Monoxide employs a new mining algorithm, Chu-ko-nu, that incentivizes honest parties to mine in parallel on all shards. However, this implies that a miner needs to verify transactions on all shards and maintain a transaction ledger for all shards. Hence, the verification and space overhead factors are proportional to the number of (honest) participants and the protocol does not satisfy scalability.

Theorem 9. Monoxide does not satisfy scalability in our system model.

Proof. Let m denote the number of shards (zones), m_p the fraction of mining power running the Chu-ko-nu mining algorithm and m_d the rest of the mining power ($m_p + m_d = 1$). Additionally, suppose m_s denotes the mining power of one shard. The Chu-ko-nu algorithm enforces the parties to verify transactions that belong to all shards, hence the parties store all sharded ledgers. To satisfy scalability, the space overhead factor of Monoxide can be at most $o(n)$. Thus, at most $o(n)$ parties can run the Chu-ko-nu mining algorithm, hence $nm_p = o(n)$.

We note that the adversary will not participate in the Chu-ko-nu mining algorithm as distributing the hashing power is to the adversary's disadvantage.

To satisfy persistence, every shard running the GHOST protocol [45] must satisfy the common prefix property. Thus, the adversary cannot control more than $m_a < m_s/2$ hash power, where $m_s = \frac{m_d}{m} + m_p$. Thus, we have $m_a < \frac{m_s}{2(m_d+m_p)} = \frac{1}{2} - \frac{m_d(m-1)}{2m(m_d+m_p)}$. For n sufficiently large, m_p converges to 0; hence $m_a < \frac{1}{2} - \frac{(m-1)}{2m} = \frac{1}{2m}$. From Lemma 3.10, $m = \omega(1)$, thus $m_a < 0$ for sufficiently large n. Therefore, Monoxide does not satisfy scalability in our model. □

3.5.3 OmniLedger

Overview

OmniLedger [9] proceeds in epochs, assumes a partially synchronous model within each epoch (to be responsive), synchronous communication channels between honest parties (with a large maximum delay), and a slowly-adaptive computationally-bounded adversary that can corrupt up to $f < n/4$ parties.

The protocol bootstraps using techniques from ByzCoin [39]. The core idea is that there is a global identity blockchain that is extended once per epoch with Sybil resistant proofs (proof-of-work, proof-of-stake, or proof-of-personhood [95]) coupled with public keys. At the beginning of each epoch a sliding window mechanism is employed to define the eligible validators as the ones with identities in the last W blocks, where W depends on the adaptivity of the adversary. For our definition of slowly adaptive, we set $W = 1$. The UTXO space is partitioned uniformly at random into m shards, each shard maintaining its own ledger.

At the beginning of each epoch, a new common random value is created via a distributed randomness generation (DRG) protocol. The DRG protocol employs verifiable random functions (VRF) to elect a leader who runs RandHound [54] to create the random value. The random value is used as a challenge for the next epoch's identity registration and as a seed to assigning identities of the current epoch into shards.

Once the participants for this epoch are assigned to shards and bootstrap their internal states, they start validating transactions and updating the shards' transaction ledgers by operating ByzCoinX, a modification of ByzCoin [39]. When a transaction is cross-shard, a protocol that ensures the atomic operation of transactions across shards called *Atomix* is employed. Atomix is a client-driven atomic commit protocol secure against byzantine adversaries.

`Consensus:` OmniLedger suggests the use of a strongly consistent consensus in order to support Atomix. This modular approach means that any consensus protocol [39, 40, 93, 96, 97] works with OmniLedger as long as the deployment

setting of OmniLedger respects the limitations of the consensus protocol. In its experimental deployment, OmniLedger uses a variant of ByzCoin [39] called ByzCoinX [97] in order to maintain the scalability of ByzCoin and be robust as well. We omit the details of ByzCoinX as it is not relevant to our analysis.

`CrossShard (Atomix):` Atomix is a client-based adaptation of two-phase atomic commit protocol running with the assumption that the underlying shards are correct and never crash. This assumption is satisfied because of the random assignment of parties to shards, as well as the byzantine fault-tolerant consensus of each shard.

In particular, Atomix works in two steps: First, the client that wants the transaction to go through requests a proof-of-acceptance or proof-of-rejection from the shards managing the inputs, who log the transactions in their internal blockchain. Afterwards, the client either collects proof-of-acceptance from all the shards or at least one proof-of-rejection. In the first case, the client communicates the proofs to the output shards, who verify the proofs and finish the transaction by generating the necessary UTXOs. In the second case, the client communicates the proofs to the input shards who revert their state and abort the transaction. Atomix, has a subtle replay attack, hence we analyze OmniLedger with the proposed fix [98].

`Sybil:` A global identity blockchain with Sybil resistant proofs coupled with public keys is extended once per epoch.

`Divide2Shards:` Once the parties generate the epoch randomness, the parties can independently compute the shard they are assigned to for this epoch by permuting (*mod* n) the list of validators (available in the identity chain).

`DRG:` The DRG protocol consists of two steps to produce unbiasable randomness. On the first step, all parties evaluate a VRF using their private key and the randomness of the previous round to generate a "lottery ticket". Then the parties broadcast their ticket and wait for Δ to be sure that they receive the ticket with the lowest value whose generator is elected as the leader of RandHound.

This second step is a partially-synchronous randomness generation protocol, meaning that even in the presence of asynchrony safety is not violated. If the leader is honest, then eventually the parties will output an unbiasable random value, whereas if the leader is dishonest there are no liveness guarantees. To recover from this type of fault the parties can view-change the leader and go back to the first step in order to elect a new leader.

This composition of randomness generation protocols (leader election and multiparty generation) guarantees that all parties agree on the final randomness

(due to the view-change) and the protocol remains safe in asynchrony. Furthermore, if the assumed synchrony bound (which can be increasing like PBFT [93]) is correct, an honest leader will be elected in a constant number of rounds.

Note, however, that the DRG protocol is modular, thus any other scalable distributed randomness generation protocol with similar guarantees, such as Hydrand [56] or Scrape [55], can be used.

`CompactState:` A key component that enables OmniLedger to scale is the epoch transition. At the end of every epoch, the parties run consensus on the state changes and append the new state (e. g. UTXO pool) in a state-block that points directly to the previous epoch's state-block. This is a classic technique [93] during reconfiguration events of state machine replication algorithms called checkpointing. New validators do not replay the actual shard's ledger but instead, look only at the checkpoints which help them bootstrap faster.

In order to guarantee the continuous operation of the system, after the parties finish the state commitment process, the shards are reconfigured in small batches (at most 1/3 of the parties in each shard at a time). If there are any blocks committed after the state-block, the validators replay the state-transitions directly.

Analysis

In this section, we prove OmniLedger satisfies persistence, consistency, and scalability (on expectation) but fails to satisfy liveness. Nevertheless, we estimate the efficiency of OmniLedger by providing an upper bound on its throughput factor.

Lemma 10. *At the beginning of each epoch, OmniLedger provides an unbiased, unpredictable, common to all parties random value (with overwhelming probability in t within t rounds).*

Proof. If the elected leader that orchestrates the distributed randomness generation protocol (RandHound or equivalent) is honest the statement holds. On the other hand, if the leader is byzantine, the leader cannot affect the security of the protocol, meaning the leader cannot bias the random value. However, a byzantine leader can delay the process by being unresponsive. We show that there will be an honest leader, hence the protocol will output a random value, with overwhelming probability in the number of rounds t.

The adversary cannot pre-mine PoW puzzles, because the randomness of each epoch is used in the PoW calculation of the next epoch. Hence, the expected number of identities the adversary will control (number of byzantine parties) in the next epoch is $f < n/4$. Hence, the adversary will have the smallest ticket – output of the VRF – and thus will be the leader that orchestrates the

distributed randomness generation protocol (RandHound) with probability 1/2. Then, the probability there will be an honest leader in t rounds is $1 - \frac{1}{2^t}$, which is overwhelming in t.

The unpredictability is inherited by the properties of the employed distributed randomness generation protocol. □

Lemma 11. *The distributed randomness generation protocol has $O(\frac{n \log^2 n}{R})$ amortized communication complexity, where R is the number of rounds in an epoch.*

Proof. The DRG protocol inherits the communication complexity of RandHound, which is $O(c^2 n)$ [56]. In [54], the authors claim that c is constant. However, the protocol requires a constant fraction of honest parties (e. g. $n/3$) in each of the n/c partitions of size c against an adversary that can corrupt a constant fraction of the total number of parties (e. g. $n/4$). Hence, from Lemma 3.16, we have $c = \Omega(\log n)$, which leads to communication complexity $O(n \log^2 n)$ for each epoch. Assuming each epoch consist of R rounds, the amortized per round communication complexity is $O(\frac{n \log^2 n}{R})$. □

Corollary 12. *In each epoch, the expected size of each shard is n/m.*

Proof. Due to Lemma 10, the n parties are assigned independently and uniformly at random to m shards. Hence, the expected number of parties in a shard is n/m. □

Lemma 13. *In each epoch, all shards are $\frac{1}{3}$-honest for $m \leq \frac{n}{300c \ln n}$, where c is a security parameter.*

Proof. Due to Lemma 10, the n parties are assigned independently and uniformly at random to m shards. Since $a = 1/3 > p = 1/4$, both a, p constant, the statement holds from Lemma 3.16 for $m = \frac{n}{c' \ln n}$, where $c' > 300c$. □

Note that the bound is theoretical and holds for a large number of parties since the probability tends to 1 as the number of parties grows. For practical bounds, we refer to OmniLedger's analysis [9].

Theorem 14. OmniLedger satisfies persistence in our system model for $f < n/4$.

Proof. From Lemma 13, each shard has an honest supermajority $\frac{2}{3}\frac{n}{m}$ of participants. Hence, persistence holds by the common prefix property of the consensus protocol of each shard. Specifically, for ByzCoinX, persistence holds for depth parameter $k = 1$ because ByzCoinX guarantees finality. □

Theorem 15. OmniLedger does not satisfy liveness in our system model for $f < n/4$.

Proof. To estimate the liveness of the protocol, we need to examine all the subprotocols: (i) `Consensus`, (ii) `CrossShard` or Atomix, (iii) `DRG`, (iv) `CompactState`, and (v) `Divide2Shards`.

`Consensus:` From Lemma 13, each shard has an honest supermajority $\frac{2}{3}\frac{n}{m}$ of participants. Hence, in this stage liveness holds by chain growth and chain quality properties of the underlying blockchain protocol (an elaborate proof can be found in [13]). The same holds for `CompactState` as it is executed similarly to `Consensus`.

`CrossShard:` Atomix guarantees liveness since the protocol's efficiency depends on the consensus of each shard involved in the cross-shard transaction. Note that liveness does not depend on the client's behavior; if the appropriate information or some part of the transaction is not provided in multiple rounds to the parties of the protocol then the liveness property does not guarantee the inclusion of the transaction in the ledger. Furthermore, if some other party wants to continue the process it can collect all necessary information from the ledgers of the shards.

`DRG:` During the epoch transition, the DRG protocol provides a common random value with overwhelming probability within t rounds (Lemma 10). Hence, liveness is satisfied in this subprotocol as well.

`Divide2Shrds:` Liveness is not satisfied in this protocol. The reason is that a slowly-adaptive adversary can select who to corrupt during epoch transition, and thus can corrupt a shard from the previous epoch. Since the compact state has not been disseminated in the network, the adversary can simply delete the shard's state. Thereafter, the data unavailability prevents the progress of the system. □

Theorem 16. OmniLedger satisfies consistency in our system model for $f < n/4$.

Proof. Each shard is $\frac{1}{3}$-honest (Lemma 13). Hence, consistency holds within each shard, and the adversary cannot successfully double-spend. Nevertheless, we need to guarantee consistency even when transactions are cross-shard. OmniLedger employs Atomix, a protocol which guarantees cross-shard transactions are atomic. Thus, the adversary cannot validate two conflicting transactions across different shards.

Moreover, the adversary cannot revert the chain of a shard and double-spend an input of a cross-shard transaction after the transaction is accepted in all relevant shards because persistence holds (Theorem 14). Suppose persistence holds with probability p. Then, the probability the adversary breaks consistency in a cross-shard transaction is the probability of successfully double-spending in one of the relevant to the transaction shards, $1 - p^v$, where v is the average size of transactions. Since v is constant, consistency holds with high probability, given that persistence holds with high probability. □

To prove OmniLedger satisfies scalability (on expectation) we need to evaluate the scaling factors in the following subprotocols of the system: (i) `Consensus`, (ii) `CrossShard`, (iii) `DRG`, and (iv) `Divide2Shards`. Note that `CompactState` is merely an executon of `Consensus`.

Lemma 17. *The maximum overhead factor of Consensus is $O(n/m)$.*

Proof. From Corollary 12, the expected number of parties in a shard is n/m. ByzCoin has quadratic to the number of parties worst-case communication complexity, hence the communication overhead factor of the protocol is $O(n/m)$. The verification complexity collapses to the communication complexity. The space overhead factor is $O(n/m)$, as each party maintains the ledger of the assigned shard for the epoch. □

Lemma 18. *The maximum overhead factor of Atomix (CrossShard) is $O(v\frac{n}{m})$, where v is the average size of transactions.*

Proof. In a cross-shard transaction, Atomix allows the participants of the output shards to verify the validity of the transaction's inputs without maintaining any information on the input shards' ledgers. This holds due to persistence (Theorem 14).

Furthermore, the verification process requires each input shard to verify the validity of the transaction's inputs and produce a proof-of-acceptance or proof-of-rejection. This corresponds to one query to the verification oracle for each input. In addition, each party of an output shard must verify that all proofs-of-acceptance are present and no shard rejected an input of the cross-shard transaction. The proof-of-acceptance (or rejection) consists of the signature of the shard which is linear to the number of parties in the shard. The relevant parties have to receive all the information related to the transaction from the client (or leader), hence the communication overhead factor is $O(v\frac{n}{m})$.

So far, we considered the communication complexity of Atomix. However, each input must be verified within the corresponding input shard. From Lemma 17, we get that the communication overhead factor at this step is $O(v\frac{n}{m})$. □

Lemma 19. *The maximum overhead factor of Divide2Shards is $O(\frac{n}{mR})$, where R is the number of rounds in an epoch.*

Proof. During the epoch transition each party is assigned to a shard uniformly at random and thus most probably needs to bootstrap to a new shard, meaning the party must store the new shard's ledger. At this point, within each shard OmniLedger introduces checkpoints, the state blocks that summarize the state of the ledger (CompactState). Therefore, when a party syncs with a shard's ledger,

it does not download and store the entire ledger but only the active UTXO pool corresponding to the previous epoch's state block.

For security reasons, each party that is reassigned in a new shard must receive the state block of the new shard by $O(n/m)$ parties. Thus, the communication complexity of the protocol is $O(\frac{n}{mR})$ amortized per round, where R is the number of rounds in an epoch.

The space complexity is constant as the state block has constant size, and there is no verification process at this stage.

□

Theorem 20. OmniLedger satisfies scalability in our system model for $f < n/4$ with scaling factor $\Sigma = O(\frac{n}{v \log n})$, where v constant.

Proof. To evaluate the scalability of OmniLedger, we need to estimate the maximum overhead factor of all the subprotocols of the system: (i) `Consensus`, (ii) `CrossShard`, (iii) `DRG`, and (iv) `Divide2Shards`.

`Consensus` has maximum overhead factor $O(n/m)$ (Lemma 17) while Atomix (`CrossShard`) has expected maximum overhead factor $O(v\frac{n}{m})$ (Lemma 18). As discussed, in Section 3.3, we assume the average size of transactions to be constant. In this case, the Atomix protocol has expected overhead factor $O(n/m)$.

The epoch transition consists of the `DRG`, `CompactState`, and `Divide2Shards` protocols. `CompactState` has the same overhead with Consensus hence it is not critical. For $R = \Omega(\log n)$ number of rounds per epoch, `DRG` has an expected amortized overhead factor $O(\log n)$ (Lemma 11), while `Divide2Shards` has an expected amortized overhead factor of $O(1)$ (Lemma 19).

Overall, $\Sigma = \Theta(m/v) = O(\frac{n}{v \log n})$, where the last equation holds from Lemma 3.10 and Lemma 13. □

Theorem 21. In OmniLedger, the throughput factor is $\sigma = \mu \cdot \tau \cdot \frac{m}{v} < \mu \cdot \tau \cdot \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p} \cdot \frac{1}{v}$.

Proof. In Atomix, at most v shards are affected per transaction, thus $m' < m/v$ [7]. From Lemma 3.16, $m < \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p}$. Therefore, $\sigma < \mu \cdot \tau \cdot \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p} \cdot \frac{1}{v}$ □

The parameter v depends on the input transaction set. The parameters μ, τ, a, p depend on the choice of the consensus protocol. Specifically, μ represents

[7] Note that if v is constant, a more elaborate analysis could yield a lower upper bound on m' better than m/v (depending on D_T). However, if v is not constant but approximates the number of shards m, then m' is also bounded by the scalability of the Atomix protocol (Lemma 18), and thus the throughout factor can be significantly lower.

the ratio of honest blocks in the chain of a shard. On the other hand, τ depends on the latency of the consensus protocol, i. e., what is the ratio between the propagation time and the block generation time. Last, a expresses the resilience of the consensus protocol (e. g., 1/3 for PBFT), while p the fraction of corrupted parties in the system ($f = pn$).

In OmniLedger, the consensus protocol is modular, so we chose to maintain the parameters for a fairer comparison to other protocols.

3.5.4 RapidChain

Overview

RapidChain [6] is a synchronous protocol and proceeds in epochs. The adversary is slowly-adaptive, computationally-bounded and corrupts less than 1/3 of the participants ($f < n/3$).

The protocol bootstraps via a committee election protocol that selects $O(\sqrt{n})$ parties – the root group. The root group generates and distributes a sequence of random bits used to establish the reference committee. The reference committee consists of $O(\log n)$ parties, is re-elected at the end of each epoch, and is responsible for: (i) generating the randomness of the next epoch, (ii) validating the identities of participants for the next epoch from the PoW puzzle, and (iii) reconfiguring the shards from one epoch to the next (to protect against single shard takeover attacks).

The parties are divided into shards of size $O(\log n)$ (committees). Each shard handles a fraction of the transactions, assigned based on the prefix of the transaction ID. Transactions are sent by external users to an arbitrary number of active (for this epoch) parties. The parties then use an inter-shard routing scheme (based on Kademlia [99]) to send the transactions to the input and output shards, i. e., the shards handling the inputs and outputs of a transaction, resp.

To process cross-shard transactions, the leader of the output shard creates an additional transaction for every different input shard. Then the leader sends (via the inter-shard routing scheme) these transactions to the corresponding input shards for validation. To validate transactions (i. e., a block), each shard runs a variant of the synchronous consensus of Ren et al. [96] and thus tolerates 1/2 byzantine parties.

At the end of each epoch, the shards are reconfigured according to the participants registered in the new reference block. Specifically, RapidChain uses a bounded version of Cuckoo rule [100]; the reconfiguration protocol adds a new party to a shard uniformly at random, and also moves a constant number of parties from each shard and assigns them to other shards uniformly at random.

`Consensus:` In each round, each shard randomly picks a leader. The leader creates a block, gossips the block header H (containing the round and the Merkle root) to the members of the shard, and initiates the consensus protocol on H. The consensus protocol consists of four rounds: (1) The leader gossips $(H, propose)$, (2) All parties gossip the received header $(H, echo)$, (3) The honest parties that received at least two echoes containing a different header gossip $(H', pending)$, where H' contains the null Merkle root and the round, (4) Upon receiving $\frac{nf}{m} + 1$ echos of the same and only header, an honest party gossips $(H, accept)$ along with the received echoes. To increase the transaction throughput, RapidChain allows new leaders to propose new blocks even if the previous block is not yet accepted by all honest parties.

`CrossShard:` For each cross-shard transaction, the leader of the output shard creates one "dummy" transaction for each input UTXO in order to move the transactions' inputs to the output shard, and execute the transaction within the shard. To be specific, assume we have a transaction with two inputs I_1, I_2 and one output O. The leader of the output shard creates three new transactions: tx_1 with input I_1 and output I'_1, where I'_1 holds the same amount of money with I_1 and belongs to the output shard. tx_2 is created similarly. tx_3 with inputs I'_1 and I'_2 and output O. Then the leader sends tx_1, tx_2 to the input shards respectively. In principle, the output shard is claiming to be a trusted channel [101] (which is guaranteed from the assignment), hence the input shards should transfer their assets there and then execute the transaction atomically inside the output shard (or abort by returning their assets back to the input shards).

`Sybil:` A party can only participate in an epoch if it solves a PoW puzzle with the previous epoch's randomness, submit the solution to the reference committee, and consequently be included in the next reference block. The reference block contains the active parties' identities for the next epoch, their shard assignment, and the next epoch's randomness, and is broadcast by the reference committee at the end of each epoch.

`Divide2Shards:` During bootstrapping, the parties are partitioned independently and uniformly at random in groups of size $O(\sqrt{n})$ with a deterministic random process. Then, each group runs the DRG protocol and creates a (local) random seed. Every node in the group computes the hash of the random seed and its public key. The e (small constant) smallest tickets are elected from each group and gossiped to the other groups, along with at least half the signatures of the group. These elected parties are the root group. The root group then selects the reference committee of size $O(\log n)$, which in turn partitions the parties randomly into shards as follows: each party is mapped to a random position in $[0, 1)$ using a hash function. Then, the range $[0, 1)$ is partitioned into k regions,

where k is constant. A shard is the group of parties assigned to $O(\log n)$ regions.

During epoch transition, a constant number of parties can join (or leave) the system. This process is handled by the reference committee which determines the next epoch's shard assignment, given the set of active parties for the epoch. The reference committee divides the shards into two groups based on each shard's number of active parties in the previous epoch: group A contains the $m/2$ larger in size shards, while the rest comprise group I. Every new node is assigned uniformly at random to a shard in A. Then, a constant number of parties is evicted from each shard and assigned uniformly at random in a shard in I.

`DRG:` RapidChain uses Feldman's verifiable secret sharing [90] to distributively generate unbiased randomness. At the end of each epoch, the reference committee executes a distributed randomness generation (DRG) protocol to provide the random seed of the next epoch. The same DRG protocol is also executed during bootstrapping to create the root group.

`CompactState:` No protocol for compaction of the state is used.

Analysis

RapidChain does not maintain a robust sharded transaction ledger under our security model since it assumes a weaker adversary. To fairly evaluate the protocol, we weaken our security model. First, assume the adversary cannot change more than a constant number of byzantine parties during an epoch transition, which we term *constant-adaptive adversary*. In general, we assume *bounded epoch transitions*, i. e., at most a constant number of leave/join requests during each transition. Furthermore, the number of epochs is asymptotically less than polynomial to the number of parties. In this weaker security model, we prove RapidChain maintains a robust sharded transaction ledger, and provide an upper bound on the throughput factor of the protocol.

Note that in cross-shard transactions, the "dummy" transactions that are committed in the shards' ledgers as valid, spend UTXOs that are not signed by the corresponding users. Instead, the original transaction, signed by the users, is provided to the shards to verify the validity of the "dummy" transactions. Hence, the transaction validation rules change. Furthermore, the protocol that handles cross-shard transactions has no proof of security against byzantine leaders. For analysis purposes, we assume the following holds:

Assumption 3.20. CrossShard satisfies safety even under a byzantine leader (of the output shard).

Lemma 22. *The maximum overhead factor of DRG is* $O(n/m)$.

Proof. The DRG protocol is executed by the final committee once each epoch. The size of the final committee is $O(n/m) = O(\log n)$. The communication complexity of the DRG protocol is quadratic to the number of parties [90]. Thus, the communication overhead factor is $O(n/m)$. □

Lemma 23. *In each epoch, all shards are $\frac{1}{2}$-honest for $m \leq \frac{n}{78c \ln n}$, where c is a security parameter.*

Proof. During the bootstrapping process of RapidChain (first epoch), the n parties are partitioned independently and uniformly at random into m shards [90]. For $p = 1/3$, the shards are $\frac{1}{2}$-honest only if $m \leq \frac{n}{78c \ln n}$, where c is a security parameter (Lemma 3.16). At any time during the protocol, all shards remain $\frac{1}{2}$-honest ([6], Theorem 5). Hence, the statement holds after each epoch transition, as long as the number of epochs is $o(n)$. □

Lemma 24. *In each epoch, the expected size of each shard is $O(n/m)$.*

Proof. During the bootstrapping process of RapidChain (first epoch), the n parties are partitioned independently and uniformly at random into m shards [90]. The expected shard size in the first epoch is n/m. Furthermore, during epoch transition the shards remain "balanced" (Theorem 5 [6]), i. e., the size of each shard is $O(n/m)$. □

Theorem 25. RapidChain satisfies persistence in our system model for constant-adaptive adversaries with $f < n/3$ and bounded epoch transitions.

Proof. The consensus protocol in RapidChain achieves safety if the shard has no more than $t < 1/2$ fraction of byzantine parties ([6], Theorem 2). Hence, the statement follows from Lemma 23. □

Theorem 26. RapidChain satisfies liveness in our system model for constant-adaptive adversaries with $f < n/3$ and bounded epoch transitions.

Proof. To estimate the liveness of RapidChain, we need to examine the following subprotocols: (i) `Consensus`, (ii) `CrossShard`, (iii) `DRG`, and (iv) `Divide2Shards`.

The consensus protocol in RapidChain achieves liveness if the shard has less than $\frac{n}{2m}$ byzantine parties (Theorem 3 [6]). Thus, liveness is guaranteed during `Consensus` (Lemma 23).

Furthermore, the final committee is $\frac{1}{2}$-honest with high probability. Hence, the final committee will route each transaction to the corresponding output shard. We assume transactions will reach all relevant honest parties via a gossip protocol. RapidChain employs IDA-gossip protocol, which guarantees message delivery to all honest parties (Lemma 1 and Lemma 2 [6]). From Assumption 3.20,

the protocol that handles cross-shard transactions satisfies safety even under a byzantine leader. Hence, all "dummy" transactions will be created and eventually delivered. Since the consensus protocol within each shard satisfies liveness, the "dummy" transactions of the input shards will become stable. Consequently, the "dummy" transaction of the output shard will become valid and eventually stable (consensus liveness). Thus, `CrossShard` satisfies liveness.

During epoch transition, `DRG` satisfies liveness [90]. Moreover, `Divide2Shards` allows only for a constant number of leave/join/move operations and thus terminates in a constant number of rounds. □

Theorem 27. RapidChain satisfies consistency in our system model for constant-adaptive adversaries with $f < n/3$ and bounded epoch transitions.

Proof. In every epoch, each shard is $\frac{1}{2}$-honest; hence, the adversary cannot double-spend and consistency is satisfied.

Nevertheless, to prove consistency is satisfied across shards, we need to prove that cross-shard transactions are atomic. `CrossShard` in RapidChain ensures that the "dummy" transaction of the output shard becomes valid only if all "dummy" transactions are stable in the input shards. If a "dummy" transaction of an input shard is rejected, the "dummy" transaction of the output shard will not be executed, and all the accepted "dummy" transactions will just transfer the value of the input UTXOs to other UTXOs that belong to the output shard. This holds because the protocol satisfies safety even under a byzantine leader (Assumption 3.20).

Lastly, the adversary cannot revert the chain of a shard and double-spend an input of the cross-shard transaction after the transaction is accepted in all relevant shards because consistency with each shard and persistence (Theorem 14) hold. Suppose persistence holds with probability p. Then, the probability the adversary breaks consistency in a cross-shard transaction is the probability of successfully double-spending in one of the relevant to the transaction shards, hence $1 - p^v$ where v is the average size of transactions. Since v is constant, consistency holds with high probability, given persistence holds with high probability. □

Similarly to OmniLedger, to calculate the scaling factor of RapidChain, we need to evaluate the following protocols of the system: (i) `Consensus`, (ii) `CrossShard`, (iii) `DRG`, and (iv) `Divide2Shards`.

Lemma 28. *The maximum overhead factor of Consensus is* $O(\frac{n}{m})$.

Proof. From Lemma 24, the expected number of parties in a shard is $O(n/m)$. The consensus protocol of RapidChain has quadratic to the number of parties communication complexity. Hence, the communication overhead factor `Consensus` is $O(\frac{n}{m})$. The verification complexity collapses to the communication complexity.

The space overhead factor is $O(\frac{n}{m})$, as each party maintains the ledger of the assigned shard for the epoch. □

Lemma 29. *The maximum overhead factor of CrossShard is $O(v\frac{n}{m})$, where v is the average size of transactions.*

Proof. During the execution of the protocol, the interaction between the input and output shards is limited to the leader, who creates and routes the "dummy" transactions. Hence, the communication complexity of the protocol is dominated by the consensus within the shards. For an average size of transactions v, the communication overhead factor is $O(vn/m + v) = O(vn/m)$ (Lemma 24). Note that this bound holds for the worst case, where transactions have $v - 1$ inputs and a single output while all UTXOs belong to different shards.

For each cross-shard transaction, each party of the input and output shards queries the verification oracle once. Hence, the verification overhead factor is $O(vn/m)$. Last, the protocol does not require any verification across shards and thus the only storage requirement for each party is to maintain the ledger of the shard it is assigned to. □

Lemma 30. *The maximum overhead factor of Divide2Shards is $O(\frac{R \cdot n}{m^2})$.*

Proof. The number of join/leave and move operations is constant per epoch, denoted by k. Further, each shard is $\frac{1}{2}$-honest (Lemma 23) and has size $O(\frac{n}{m})$ (Lemma 24); these guarantees hold as long as the number of epochs is $o(n)$.

Each party changing shards receives the new shard's ledger of size T/m by $O(n/m)$ parties in the new shard. Thus the total communication complexity at this stage is $O(\frac{T}{m} \cdot \frac{n}{m})$, hence the communication overhead factor is $O(\frac{T}{m^2}) = O(\frac{R \cdot e}{m^2})$, where R is the number of rounds in each epoch and e the number of epochs since genesis. Since $e = o(n)$, the communication overhead factor is $O(\frac{R \cdot n}{m^2})$.

□

Theorem 31. RapidChain satisfies scalability in our system model for constant-adaptive adversaries with $f < n/3$ and bounded epoch transitions, with expected scaling factor $\Sigma = \Theta(m) = O(\frac{n}{\log n})$, for v constant and epoch size $R = O(m)$.

Proof. `Consensus` has expected maximum overhead factor $O(\frac{n}{\log n})$ (Lemma 28), while `CrossShard` has expected expected maximum overhead factor $O(v\frac{n}{\log n})$. (Lemma 29). As discussed, in Section 3.3, we assume constant average size of transactions, thus the expected maximum overhead factor $O(\frac{n}{\log n})$.

During epoch transitions, `DRG` has expected maximum overhead factor $O(\frac{n}{m})$ (Lemma 22) while `Divide2Shards` has expected overhead factor $O(\frac{n \cdot R}{m^2})$ (Lemma

30). Thus for $R = O(m)$, the maximum overhead factor during epoch transitions is $O(n/m)$.

Overall, RapidChain's expected scaling factor is $\Theta(m) = O(\frac{n}{\log n})$, where the equation holds for $m = \frac{n}{78c \ln n}$ (Lemma 23). □

Theorem 32. In RapidChain, the throughput factor is $\sigma = \mu \cdot \tau \cdot \frac{m}{v} < \mu \cdot \tau \cdot \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p} \cdot \frac{1}{v}$.

Proof. In RapidChain, at most v shards are affected per transaction – when each transaction has $v-1$ inputs and one output, and all belong to different shards. Therefore, $m' < m/v$. From Lemma 3.16, $m < \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p}$. Therefore, $\sigma < \mu \cdot \tau \cdot \frac{n}{\ln n} \cdot \frac{(a-p)^2}{2+a-p} \cdot \frac{1}{v}$. □

In RapidChain, the consensus protocol is synchronous and thus not practical. We estimate the throughput factor irrespective of the chosen consensus, to provide a fair comparison to other protocols. We notice that both RapidChain and OmniLedger have the same throughout factor when v is constant.

We provide an example of the throughput factor in case the employed consensus is the one suggested in RapidChain. In this case, we have $a = 1/2$, $p = 1/3$, $\mu = 1/2$ (Theorem 1 [6]), and $\tau = 1/8$ (4 rounds are needed to reach consensus for an honest leader, and the leader will be honest every two rounds on expectation [33].). Note that τ can be improved by allowing the next leader to propose a block even if the previous block is not yet accepted by all honest parties; however, we do not consider this improvement. Hence, for $v = 5$, we have throughput factor

$$\sigma < \frac{1}{2} \cdot \frac{1}{8} \cdot \frac{n}{\ln n} \cdot \frac{(\frac{1}{2} - \frac{1}{3})^2}{2 + \frac{1}{2} - \frac{1}{3}} \cdot \frac{1}{5} = \frac{n}{6240 \ln n}$$

3.5.5 Chainspace

Chainspace is a sharding protocol introduced by Al-Bassam et al. [102] that operates in the permissioned setting. The main innovation of Chainspace is on the application layer. Specifically, Chainspace presents a sharded, UTXO-based distributed ledger that supports smart contracts. Furthermore, limited privacy is enabled by offloading computation to the clients, who need to only publicly provide zero-knowledge proofs that their computation is correct. Chainspace focuses on specific aspects of sharding; epoch transition or reconfiguration of the protocol is not addressed. Nevertheless, the cross-shard communication protocol, namely S-BAC, is of interest as a building block to secure sharding.

S-BAC protocol. S-BAC is a shard-led cross-shard atomic commit protocol used in Chainspace. In S-BAC, the client submits a transaction to the input shards. Each shard internally runs a BFT protocol to tentatively decide whether to accept or abort the transaction locally and broadcasts its local decision to other shards that take part in the transaction. If the transaction fails locally (e. g., is a double-spend), then the shard generates pre-abort(T), whereas if the transaction succeeds locally the shard generates pre-accept(T) and changes the state of the input to 'locked'. After a shard decides to pre-commit(T), it waits to collect responses from other participating shards, and commits the transaction if all shards respond with pre-accept(T), or aborts the transaction if at least one shard announces pre-abort(T). Once the shards decide, they send their decision (accept(T) or abort(T)) to the client and the output shards. If the decision is accept(T), the output shards generate new 'active' objects and the input shards change the input objects to 'inactive'. If an input shard's decision is abort(T), all input shards unlock the input objects by changing their state to 'active'.

S-BAC, just like Atomix, is susceptible to replay attacks [98]. To address this problem, sequence numbers are added to the transactions, and output shards generate dummy objects during the first phase (pre-commit, pre-abort). More details and security proofs can be found on [98], as well as a hybrid of Atomix and S-BAC called Byzcuit.

3.5.6 Comparison of Sharding Protocols

Table 3.2 illustrates the comparison between the sharding protocols, evaluated with respect to the desired properties defined in Section 3.2.2.

Table 3.2: Summarizing sharding protocol properties under our model

Protocol	Persistence	Consistency	Liveness	Scalability	Permissionless	Slowly-adaptive
Elastico	✓	✗	✓	✗	✓	✓
Monoxide	✓	✓	✓	✗	✓	✓
OmniLedger	✓	✓	✗	✓	✓	✓
RapidChain	✓	✓	✓	✓	✓	∼
Chainspace	✓	✓	✓	✓	✗	✗

We first showed that Elastico does not satisfy consistency due to its `CrossShard` protocol (as long as the workload does have cross-shard transactions). Additionally, Elastico does not satisfy scalability as all epoch-transition protocols (e. g., `Divide2Shards`,`CompactState`) are executed for every block (epoch in Elastico). We highlight that Elastico does not satisfy scalability by design regardless the transaction distribution; that is, even with a few cross-shard transactions, Elastico maintains a global hash chain which is broadcast to all the system's participants, and thus the scaling factor is constant at best (determined by the block size).

We then showed that Monoxide does not satisfy scalability because the state

is not partitioned when security properties are satisfied. We observe that the lack of scalability stems from the protocol's mechanism to defend against adversaries in the PoW setting; hence it cannot scale even in an optimistic transaction distribution with no cross-shard transactions.

Third, we proved that OmniLedger maintains all properties but liveness. Specifically, OmniLedger uses checkpoints of the UTXO pool at the end of each epoch, but the state is not broadcast to the network. Therefore, OmniLedger is vulnerable to slowly adaptive adversaries that can corrupt a shard from the previous epoch before the new nodes of the shard bootstrap to the state during epoch transition. This attack violates the liveness property of the system. Nevertheless, simply adding a reliable broadcast step at the end of each epoch restores the liveness of OmniLedger, as all other components satisfy liveness. The overhead of this step can be amortized over the rounds of the epoch hence scalability is also maintained.

Fourth, we proved RapidChain maintains a robust sharded ledger but only under a weaker model than the one defined in Section 3.2. Specifically, the protocol only allows a constant number of parties to join or leave and the adversary can at most corrupt a constant number of additional parties with each epoch transition. Another shortcoming of RapidChain is the synchronous consensus mechanism it employs. In case of temporary loss of synchrony in the network, the consensus of cross-shard transactions is vulnerable, hence consistency might break [6]. However, most of these drawbacks can be addressed with simple solutions, such as changing the consensus protocol (trade-off performance with security), replacing the epoch transition process with one similar to (fixed) OmniLedger, etc. Although OmniLedger (with the proposed fix) maintains a robust sharded ledger in a stronger model (as defined in Section 3.2), RapidChain introduces practical speedups on specific components of the system. These improvements are not asymptotically important – and thus not captured by our framework – but might be significant for the performance of deployed sharding protocols.

Finally, we include Chainspace in the comparison of sharding protocols, which maintains a robust sharded transaction ledger but only in the permissioned setting against a static adversary. Chainspace could be secure in our model in the permissioned setting if it adopts OmniLedger's epoch transition protocols and the proposed fix for data availability in the verifiable compaction of state. We omit the security proofs for Chainspace since they are either included in [102] or are similar to OmniLedger. We include in the evaluation the "permissionless" and "slowly-adaptive" property, in addition to the security and efficiency properties, namely persistence, consistency, liveness and scalability.

Lastly, we note that the cross-shard communication protocols of some of these sharding systems suffer from replay attacks [98]. In our analysis, we consider the fixes proposed in [98].

3.6 Conclusion, Limitations, and Extensions

Summary. In this chapter, we defined and analyzed the properties a sharded distributed ledger should fulfill. More specifically, we showed that a sharded blockchain with n participants cannot be scalable under a fully adaptive adversary, but it can scale up to $O(n/\log n)$ under an epoch-adaptive adversary. This is possible only if the distributed ledger creates succinct proofs of the valid state updates at the end of each epoch. Our model builds upon and extends the Bitcoin backbone protocol by defining *consistency* and *scalability*. Consistency encompasses the need for atomic execution of cross-shard transactions to preserve safety, whereas scalability encapsulates the speedup a sharded system can gain in comparison to a non-sharded system. We further introduced a protocol abstraction and highlighted the sufficient components for secure and efficient sharding in our model. In order to show the power of our framework, we analyzed the most prominent sharded blockchains (Elastico, Monoxide, OmniLedger, RapidChain) and showed that their lack of formal treatment leads to all of them being subpar (or clearly broken) to the sharding crux.

Application. Although we restrict our evaluation to the most impactful (so far) sharding proposals, we stress that the power of our framework and the bounds we provide are not limited to these works. For instance, we observe that Chainweb [103], a recently deployed sharding proposal, is not scalable – as it claims – because it violates Theorem 3.15. We believe out framework is general enough to cover most sharding approaches, and we aspire it will be established as a tool for proving security of sharding protocols in the future.

Multi-consensus vs uni-consensus. In our study, we assumed all shards run consensus to ensure the transaction validity. A different approach that was recently proposed [89, 104], the so-called uni-consensus[8], suggests a system design where the transaction order, and thus the transaction validity, is established through a main hash-chain that contains the compressed information for all shards. To achieve almost linear scaling, breaking the impossibility result of Theorem 3.12, [89] propose dynamically adjustable block size, proportional to the number of nodes in the network.

We consider this approach outside the realm of our study, since *the growth of such a system inevitably leads to increased resource requirements* (e. g., bandwidth of each node). This holds because the information compressed in the hash-chain must be at least linear to the number of nodes in the network for such a system to scale. But this means that the block size is at least linear to the number of nodes in the network, which in turn implies that the bandwidth of each node will grow

[8] As opposed to the classic approach studied in this work, which is termed multi-consensus.

proportionally to the growth of the system. This contradicts the definition of sharding and we therefore believe such systems fall under the study of performance optimizations on the consensus layer of blockchains, e. g., [34, 35, 105, 106].

Part IV

Payment Channels

Chapter 4

Cerberus: Incentive-Compatible Channels for Bitcoin

4.1 Introduction

4.1.1 Motivation

Since its inception, Bitcoin [1] is the leading cryptocurrency in terms of market capitalization. Unfortunately, Bitcoin suffers from limited transaction throughput due to its underlying consensus mechanism, as explained in Section 2.4. Payment channels are the foremost solution for scaling already-deployed decentralized blockchain systems such as Bitcoin.

Although Lightning channels offer a simple and efficient solution to the limited transaction throughput of blockchain systems, they also suffer from the availability problem described in Section 2.6.3. In a nutshell, the correct operation of a payment channel depends on all parties of the channel being active and in sync with the blockchain. Otherwise, a party can close the channel in an old state which will be finalized, unless the counterparty revokes it within the dispute period.

A natural solution to relieve the parties from this necessity is outsourcing the dispute process to third-parties called watchtowers. However, all current proposals and implementations of watchtowers on Bitcoin Lightning network [71, 72] do not provide incentives for watchtower participation. In particular, the party that hires the watchtower pays it only if fraud happens. However, the watchtower knows that rational parties never commit fraud, thus there is little incentive to become a watchtower in the first place. Additionally, a rational watchtower can benefit from unintentional broadcasting of revoked updates and thus may lobby for buggy or misleading channel software. A naive alternative would be for the hiring party to pay the watchtower a small fee regularly, e. g., each time a channel transaction is executed. In this case, however, a rational watchtower would avoid the cost of

storing the hiring party's data and monitoring the blockchain and would thus fail to act upon fraud.

We introduce Cerberus channels, where watchtowers are (i) incentivized to participate in the system, and (ii) penalized in case they do not act upon fraud. In particular, each party has the option to employ a watchtower as a service provider. The watchtower is paid for every transaction executed on the channel and locks collateral on-chain as guarantee for its honest behavior. In case the watchtower misbehaves and does not dispute an outdated state, the cheated party can claim the watchtower's collateral. Hence, rational watchtowers are incentivized both to participate and act upon fraud. In our construction, the parties can go offline for an extended period of time and need only be online to penalize the watchtower. This way we weaken the availability requirement for the parties of a payment channel. More importantly, Cerberus channels build upon and extend Lightning channels and only require timelocks and additional transactions.

4.1.2 Contribution

To summarize, the contribution of this chapter is the following:

- We introduce Cerberus payment channels that enable participants on Bitcoin to employ watchtowers and thus go securely offline for an extended period of time.
- We define the desired properties for payment channel solutions and prove Cerberus channels are secure under our security model. Specifically, we show watchtowers are incentivized to both participate and act upon fraud. Thus, Cerberus channels are secure against collusion and bribing.

4.2 Protocol Overview

4.2.1 System Model

Cryptographic assumptions. We make the typical cryptographic assumptions, i. e., there are secure communication channels between participants, and cryptographically secure hash functions, signatures, and encryption schemes. Additionally, all parties of the protocol (watchtowers, channel parties, external adversaries) are computationally bounded.

Blockchain assumptions. We assume a perfect blockchain, in the sense that both persistence and liveness hold [13]. In particular, we assume that if a valid

transaction is propagated in the blockchain network it cannot be censored and will be included in the "permanent" part of the blockchain immediately[1].

Network model. We assume that any participant of a channel can go offline (intentionally or due to a Denial-of-Service (DoS) attack) for a (long) period of time up to T. Furthermore, we consider watchtowers that are resilient against DoS attacks. We argue this assumption is realistic since watchtowers are also required to lock high collateral to participate in the system and thus operators will invest in anti-DoS protection.

Threat model. We assume that the watchtowers as well as the channel participants are rational players. Thus, they will only deviate from the honest protocol execution if they can gain more profit. Moreover, channel participants can collude with the watchtower(s).

Transaction model. We assume a UTXO-based transaction model and follow the notation introduced in Section 2.1.3.

4.2.2 Cerberus Overview

Cerberus channels aim to alleviate the need for the channel parties to be frequently online watching the blockchain. We propose simple modifications on the Lightning protocol that allow the parties to employ watchtowers while incentives for active participation and thus security of the channels are guaranteed.

In particular, in Cerberus, the watchtower is rewarded for every update on the channel but also locks collateral as guarantee for the case the watchtower does not act upon fraud. The party employing the watchtower can claim the collateral within the penalty period, if the latter misbehaves. The penalty period is however much larger than the revocation period, hence the hiring party can go offline for an extended period of time. On the other hand, on a normal operation of the channel the watchtower can reclaim the collateral when its service is terminated. The watchtower has the option to terminate its service to the party at any point during the protocol execution subject to the penalty period. In such a case, the party can either update the channel and employ a new watchtower, close the channel, or be online more frequently to avoid fraud.

[1]Note that Cerberus channels can be made secure for any confirmation time k, but we choose $k = 1$ to simplify the protocol and security analysis.

4.2.3 Payment Channel Properties

We define the desired properties of a payment channel construction below, summarizing the work of [65, 73, 107, 108].

1. **Security:** Any party of the channel can enforce the last agreed state (balance) on-chain at any time.
2. **Privacy:** No third-party, external to the payment channel, can gain information about the state (distribution of funds) of the channel.
3. **Scale-out:** The number of transactions on-chain is constant.

The first two, namely safety and privacy, are security properties, while the third is the performance property that is required in a channel to achieve its main purpose – higher transaction throughput.

We note that these properties are also met by Lightning, but under a different network model. Specifically, Lighting requires participants to be frequently online watching the blockchain to guarantee security. In this work, we aim to alleviate this requirement, thus providing security in the model specified in Section 4.2.1.

4.3 Cerberus Design

In this section, we describe in detail the architecture of Cerberus payment channels. We base our design on Lightning channels, and introduce the necessary modifications and extensions to guarantee the desired properties under the predefined model for our design and the applicability to Bitcoin.

First, we divide the channel lifetime in four phases: *Open, Update, Abort,* and *Close*. Then, we describe the necessary transactions and present the protocol design for each phase. To simplify the description, we assume, wlog, that party B has hired watchtower W and the potentially cheating party is A.

We note that Cerberus can accommodate the usecase where only one party wishes to hire a watchtower, as well as that in which both parties choose to do so. Different watchtowers can be used by the two parties. In the case neither party employs a watchtower, the protocol reverts to Lightning.

4.3.1 Phase: Open

Similarly to the Lightning protocol, phase *Open* includes a funding transaction and a commitment transaction. The funding transaction creates a common account between the parties and is eventually published on-chain to notarize the committed funds, hence the opening of the channel. The funding transaction of

a Cerberus channel spends the funds of the channel parties and creates a new 2-of-2 multisig output spendable only if the parties collaborate.

The commitment transaction (first of many to follow) distributes the funds between the parties and is signed and held in private by both parties. The commitment transaction, if published, does not allow the publishing party to spend its funds immediately, to ensure there is enough time for punishment in case of fraud (i. e., the commitment transaction is not the last agreed state by the channel parties). This is known as the revocation period, denoted by t. The first commitment transaction should be signed by both parties before signing and publishing the funding transaction to avoid a hostage situation.

Furthermore, we introduce two new transactions in phase *Open*, the *collateral transaction* and *reclaim transaction*, that involve the watchtower service. The collateral transaction is funded by the watchtower, while its output is a joint account between the watchtower and the hiring party. The value of the collateral is slightly higher than the channel funds. The reclaim transaction, on the other hand, allows the watchtower to reclaim the collateral, effectively terminating its service. The output of the reclaim transaction can be spent as follows: either a long "penalty" period T has elapsed since the watchtower signed and published on-chain the reclaim transaction, or both the signatures of the watchtower and the party are present. Intuitively, the penalty period T allows the cheated party to penalize an inactive/malicious watchtower, during phase *Close*. Further, timelock T allows the party employing the watchtower to be notified either in case of fraud or in case the watchtower simply wants to withdraw its service. In the latter case, the party can either be online more frequently (revocation period), close the channel, or collaboratively update the channel to cancel out the previous commitment transactions and employ a new watchtower. Note that the reclaim transaction is signed by both the watchtower and the hiring party before the collateral transaction is put on-chain to avoid a hostage situation.

Further, $T >> t$, and security holds as long as any participant in a Cerberus channel employing a watchtower is offline for at most T time.

Next, we present in detail the transactions involved in the first phase, *Open*, which is described in Protocol 2.

- **Funding transaction:** Opens a channel between two parties, A and B. The inputs are the parties' funds and the output is a 2-of-2 multisig of A and B.

- **Commitment transaction:** Updates the state of the channel, i. e., the distribution of the funds between the parties. The input of the commitment transaction spends the output of the funding transaction. The commitment transaction has two outputs, one for each channel party, and distributes the funds of the channel to the two parties as agreed. Each party has its own version of the commitment transaction, signed by the counterparty. Each

version has two outputs, one awarded to the party holding the commitment transaction, wlog party A, and one awarded to the counterparty B. Both outputs are timelocked for the revocation period t. Further, each output can be spent before t time elapses in collaboration with the watchtower W, i. e., if both the watchtower and the corresponding party sign a transaction.

- **Collateral transaction:** Commits the watchtower's collateral on-chain. Note that the value of the collateral should be slightly higher than the total funds of the channel. The input of the collateral transaction is funded by the watchtower, while the output is a 2-of-2 multisig of B and W.

- **Reclaim transaction:** Allows the watchtower to reclaim the collateral. The input of the reclaim transaction spends the output of the collateral transaction. The output, on the other hand, requires the signature of the watchtower relatively timelocked by T or both the signatures of W and B.

Protocol 2: Open a CERBERUS channel

Data: A, B and W own on-chain a, b and c coins respectively. It holds that $c > a + b$ and W is employed by B.

Result: Opening of the channel between A and B with total value $a + b$ coins, and employment of W from B with c coins as collateral.

/* A and B prepare the necessary transactions */

1. A and B create the funding transaction:
$TX_f = \left[(a \mid \#_{\sigma_A}), (b \mid \#_{\sigma_B})\right] \mapsto (a + b \mid \sigma_{A,B})$

2. A and B create the first commitment transaction (version held by A):
$TX_{C1A} = o_f \mapsto \left[(a \mid (\sigma_A \wedge \Delta t) \vee \sigma_{A,W}), (b \mid (\sigma_B \wedge \Delta t) \vee \sigma_{B,W})\right]$

/* A and B open the channel */

3. B sends the signature of TX_{C1A} to A. Symmetrically, A sends the signature of TX_{C1A} to B.

4. Both A and B sign TX_f and publish it on-chain, as in Lightning.

/* W locks its collateral for the channel on-chain */

5. W creates the collateral and reclaim transactions and sends both to B:
$TX_{Coll} = (c|\#_{\sigma_W}) \mapsto (c \mid \sigma_{W,B})$
$TX_{RC} = o_{Coll} \mapsto (c \mid (\sigma_W \wedge \Delta T) \vee (\sigma_{B,W}))$

6. B verifies, signs and sends to W the signature for the input of TX_{RC}.

7. W signs and publishes on-chain the collateral transaction TX_{Coll}.

4.3.2 Phase: Update

The second phase, *Update*, materializes the main functionality of a channel. In this phase, the parties update the current state of the channel (distribution of funds) and consequently transactions are executed off-chain. The *Update* phase in the Lightning protocol consists of a (new) commitment transaction and a revocation transaction. The commitment transaction represents the last, agreed by both parties, state of the channel. On the other hand, the revocation transaction allows a party to claim all the funds of the channel in case the other party publishes the previous commitment transaction (attempts to cheat).

The *Update* phase in Cerberus produces the following transactions: a commitment transaction, a revocation transaction, and two *penalty transactions*. The revocation transaction spends both outputs of the previous valid commitment transaction and awards them to the cheated party B. The revocation transaction is signed by the potentially cheating party A and is sent to B. Then, both party B and watchtower W sign, exchange, and store the revocation transaction. Note that B will not sign the new commitment transaction unless both A and W sign the revocation transaction. Further, A acts as in Lightning.

The penalty transactions allow B to penalize watchtower W during the penalty period in case fraud occurred and W did not publish the revocation transaction in time. Specifically, the penalty transactions both depend on the previous commitment transaction and on the collateral and reclaim transactions, respectively. Hence, the penalty transaction is valid only if fraud occurs and is not revoked. Thus, the revocation transaction has a double functionality; it awards the money of cheater A to the cheated party B and additionally acts as insurance for W, since it invalidates the penalty transactions by spending the outputs of the commitment transaction. Note that B will only sign the new commitment transaction after receiving the signed penalty transactions from W.

Further, we assume that a watchtower is paid regularly on every channel update by the hiring party via a unidirectional payment channel[2].

To sum up, during the *Update* phase the following transactions are created:

- **Revocation transaction:** In case of fraud, i. e., if party A publishes a revoked commitment transaction, the revocation transaction returns all channel funds to the cheated party B. The inputs of the revocation transaction spend the outputs of the commitment transaction. The output of the revocation transaction is awarded to the cheated party B.

- **Penalty transaction 1:** Allows a party to claim the watchtower's collateral during the penalty period T, in case fraud occurred and the watchtower did not act during the revocation period t. The inputs of penalty transaction 1

[2]Ideally, this payment should be integrated with Cerberus for efficiency (see Section 4.5).

are (a) the output that reflects B's channel balance on the corresponding commitment transaction, and (b) the output of the collateral transaction. The output of penalty transaction 1 awards all funds to B.

- **Penalty transaction 2:** Allows a party to claim the watchtower's collateral during the penalty period T, in case fraud occurred and the watchtower did not revoke it during time t, but tried to reclaim the collateral. The inputs of penalty transaction 2 are (a) the output that reflects B's channel balance on the corresponding commitment transaction, and (b) the output of the reclaim transaction. The output awards all funds to B.

All described transactions and their dependencies are illustrated in Protocol 3.

Protocol 3: Update a CERBERUS channel

Data: A and B own a coins and b coins respectively in a channel. W has a locked collateral of c coins. Note that $a' + b' = a + b$.
Result: A, B own a', b' coins, resp. W has c coins locked as collateral.

/* A,B, and W create the revocation transaction */
1. B creates the next commitment transaction: // but does not sign it
$TX_{C,i+1,A} = o_f \mapsto \left[(a' \mid (\sigma_A \wedge \Delta t) \vee \sigma_{A,W}), (b' \mid (\sigma_B \wedge \Delta t) \vee \sigma_{B,W})\right]$.

2. B creates and sends to A the revocation transaction for the previous commitment transaction: $TX_{RiA} = \left[o^1_{CiA}, o^2_{CiA}\right] \mapsto (a + b \mid \sigma_B)$.

3. A sends to B its signature for the revocation transaction TX_{RiA}.

4. B sends to W both parties' signatures for the revocation transaction TX_{RiA}, along with the commitment transaction TX_{CiA}.

5. W sends to B its signature for the revocation transaction TX_{RiA}.

/* A,B, and W create the penalty transactions */
6. W creates penalty transactions 1 and 2:
$TX_{P1iA} = \left[o_{Coll}, o^2_{CiA}\right] \mapsto (b + c \mid \sigma_B)$,
$TX_{P2iA} = \left[o_{RC}, o^2_{CiA}\right] \mapsto (b + c \mid \sigma_B)$.

7. W sends to B its signatures for o_{Coll} and o_{RC}.

/* The channel is updated */
8. B sends to A its signature for $TX_{C,i+1,A}$.

/* We omit the process for party B due to space limitations */

4.3.3 Phase: Close

The last phase, *Close*, handles the closing of a Cerberus channel. Similarly to the Lightning protocol, in this phase either the parties close the channel in collaboration or a commitment transaction is published on-chain unilaterally by one of the channel parties. As soon as the commitment transaction is included on-chain the revocation period t begins, allowing the counterparty or the watchtower to supervene to a potential dispute resolution.

Collaborative closure. The normal closure of the channel (no cheating occurs) is described in Protocol 4. Note that in this case, both parties sign the agreed distribution of the channel's funds and the funds are immediately awarded to the parties as soon as the transaction is included in the blockchain, i. e., no timelocks are required.

Non-collaborative non-cheating closure. This happens if one party wants to close the channel and the other party is unresponsive. Then, the responsive party publishes on-chain the last commitment transaction (already signed by both parties on last update). After time t, the funds of the channel are distributed to the parties according to the published state. As soon as the last commitment is published on-chain, the watchtower can safely put on-chain the reclaim transaction, since no penalty transaction corresponding to the last commitment transaction exists. Consequently, the watchtower can spend the collateral after time T, or immediately if the hiring party agrees to collaborate and sign a transaction that spends the collateral and awards the funds to the watchtower.

Protocol 4: Non-cheating Cerberus channel closure

Data: A owns a coins, B owns b coins, W has a locked collateral of c coins.

Result: The channel is closed, A owns a' coins, B owns b' coins and the collateral is returned to the watchtower W.

1. A and B sign and broadcast $o_f \mapsto \big[(a \mid \sigma_A), (b \mid \sigma_B)\big]$.

2. W publishes on-chain the reclaim transaction TX_{RC} (can spend it after T).

Cheating closure & responsive watchtower. In case A cheats and publishes an old commitment transaction, the watchtower publishes the corresponding revocation transaction during the dispute period, awarding all the funds of the channel to the cheated party B. Then, the watchtower can publish the reclaim

transaction and spend its output safely since the corresponding penalty transaction has been invalidated. As described in Protocol 5, publishing the revocation and reclaim transaction can be done simultaneously, since the timelock t on the hiring party's output of the published commitment transaction guarantees that the hiring party cannot claim the watchtower's collateral during the revocation period. On the other hand, in case B cheats, A acts exactly as in Lightning.

Protocol 5: CERBERUS close – cheating party & responsive watchtower

Data: A owns a coins, B owns b coins, W has a locked collateral of c coins. The last commitment transaction is denoted TX_{CnA}.
Result: The channel is closed, the channel funds are awarded to the cheated party B, the collateral returned to the watchtower W.

1. Party A publishes on-chain an old commitment transaction $TX_{CiA},\ i < n$.

2. During the revocation period t, W publishes the corresponding revocation transaction TX_{RiA}.

3. W publishes on-chain the reclaim transaction TX_{RC} (can spend it after T).

Cheating closure & unresponsive watchtower. If the watchtower does not publish the revocation transaction in time when fraud occurs, the cheated party can publish a penalty transaction and claim the watchtower's collateral. Specifically, if the reclaim transaction is not published, the cheated party publishes penalty transaction 1. Otherwise, if the watchtower has published the reclaim transaction but not the revocation transaction, the cheated party publishes penalty transaction 2. Both penalty transactions are valid, as long as the cheated party does not spend its output of the commitment transaction. Further, penalty transaction 2 is valid only if less than time T has elapsed since the reclaim transaction was put on-chain. Hence, the cheated party must go online within at most T time to claim the watchtower's collateral. Protocol 6 describes this case.

4.3.4 Phase: Abort

In this phase, W withdraws the collateral and thus terminates its employment by party B. *Abort* is an intermediate phase, that can occur at any time between phases *Open* and *Close*[3]. To that end, the watchtower publishes on-chain the

[3]We assume that a rational watchtower will publish the reclaim transaction at the latest when the channel is closed.

Protocol 6: Cerberus close – cheating party & malicious watchtower

Data: A owns a coins, B owns b coins, W has a locked collateral of c coins. The last commitment transaction is denoted TX_{CnA}.

Result: The channel is closed, the channel funds are awarded to the parties according to the published commitment transaction TX_{CiA}, and the collateral (c coins) is awarded to party B.

1. Party A publishes on-chain an old commitment transaction $TX_{CiA},\ i < n$ waits for the timelock to expire and spends the $(a \mid \sigma_A \wedge \Delta t)$ output.

2. B checks the chain periodically every time T and notices TX_{CiA} is on-chain and spent. If W has published the reclaim transaction TX_{RC}, B publishes Penalty transaction 2 TX_{P2iA}. Else, B publishes Penalty transaction 1 TX_{P1iA}.

reclaim transaction. Consequently, timelock T comes in effect, locking the watchtower's collateral for the penalty period. During this period, an honest hiring party will check the blockchain once. The party can then close the channel, hire a new watchtower or sync with the blockchain once every t time.

4.4 Security Analysis

4.4.1 Security

We show that Cerberus channels are secure within our system model under any collusion/bribing scheme involving the channel parties and the watchtower.

Lemma 4.1. *Phase Abort does not affect the security of a* Cerberus *channel, i. e., no honest party or watchtower can be cheated out of its funds.*

Proof. To prove the lemma, we distinguish three cases:

(a) *Abort* terminates before *Close* initiates.

(b) *Abort* terminates during *Close*.

(c) *Abort* includes *Close* entirely, i. e., *Close* initiates at most $T - t$ after the reclaim transaction is put on-chain.

In the first case, the watchtower withdraws the collateral, and is not liable anymore for the channel operation. Specifically, W publishes on-chain the reclaim transaction. Since no commitment transaction is published on-chain, all penalty

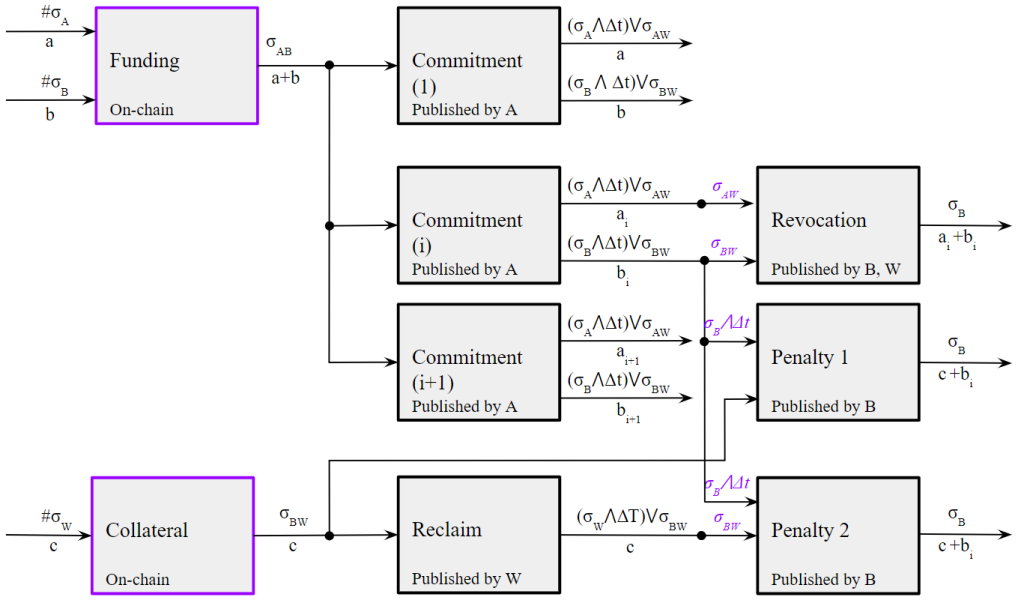


Figure 4.1: CERBERUS channel transactions. When many spending conditions are available in the output, the one used is explicitly shown emphasized in the input.

transactions that can interfere with the ownership of the collateral are invalid. Hence, the watchtower claims the collateral after time T elapses – before *Close* initiates. From that point on, the security of the channel – the funds of the hiring party – is the same as in a Lightning channel, unless the channel is updated with a new watchtower service.

For the second case, suppose *Abort* initiates at time $t = 0$, and there is a time t', such that $T - t < t' < T$, in which *Close* initiates. If the closing of the channel is collaborative between the parties or the last commitment transaction was published by one of the parties, there are no valid penalty transactions that can interfere with the ownership of the collateral. Hence, the collateral will be owned by the watchtower at time T. Further, the funds of the channel parties will be distributed as last agreed. However, the channel can close with one of the parties publishing an old commitment transaction. In this case, both penalty transactions become valid at time $t' + t > T$. But the watchtower can spend the collateral at time T, before any penalty transaction becomes valid. Therefore, when Abort terminates during *Close*, the watchtower can safely reclaim the collateral, no matter how a CERBERUS channel closes. Note, however, that the watchtower is not liable for the channel's correct operation when Abort terminates during *Close*. Thus, the watchtower is not obligated to publish the revocation transaction in time. Nevertheless, the hiring party comes online once during the *Abort* phase. From then on, it comes online every t time, since it realizes that the watchtower stops offering its service. Furthermore, it holds $t' + t > T$. Therefore, the hiring

party will notice the fraud in time and will publish the revocation transaction. Hence, no honest hiring party or watchtower can be cheated out of its funds when Abort terminates during *Close*.

In contrast to the first two cases, when *Abort* includes *Close* entirely, the watchtower is still responsible for the correct operation of the channel. In such a case, the security of a Cerberus channel is the same with or without phase *Abort*. Thus, overall, *Abort* does not affect the security of a Cerberus channel.

□

Next, we show that any honest party will maintain at least its funds in a Cerberus channel, even against malicious parties.

Lemma 4.2. *Any honest party involved in a* Cerberus *channel cannot be cheated out of its funds.*

Proof. Watchtowers are incentivized to participate in Cerberus channels due to the occasional rewards on every update. Thus, we need only show that under any collusion scheme Cerberus channels remain secure with respect to the system model of Section 4.2.1. This implies the scheme is also secure if no collusion occurs, e. g., normal channel operation or the watchtower is offline. Note that whenever we assume collusion, it can also be the case that the same person handles both colluding identities. There can be the following collusion schemes:

(i) Both parties of the channel A and B collude. According to the cryptographic assumptions, the signature of the watchtower cannot be forged, hence the parties cannot create a penalty transaction without the collaboration of the watchtower. Moreover, the channel parties only hold the watchtower's signature for revocation and penalty transactions of previous commitment transactions. The reclaim transaction is only held by W and phase *Abort* does not affect the security of the protocol (Lemma 4.1). Therefore, the only available action for the colluding parties is to publish a previous commitment transaction on purpose. In such a case, an honest hence online watchtower publishes the corresponding revocation transaction. As soon as the revocation transaction is on-chain, one of the inputs of both penalty transactions – which is the same as the one in the revocation transaction – become invalid thus both penalty transactions become invalid. Therefore, the malicious parties cannot claim an honest watchtower's collateral.

(ii) Watchtower W colludes with party A. In this case, the malicious parties try to cheat B's balance out of the channel (equivalent to at most $a + b$ coins), while B is offline. We assume *Abort* has not initiated, since it does not affect the security of the channel[4] (Lemma 4.1). The colluding parties cannot

[4]To be precise, *Abort* could have initiated but less than $T - t$ time has elapsed between the reclaim transaction was published on-chain and the time *Close* initiated.

forge B's signature to close the channel in a new state. Thus, the only available action is to publish a previous and more favorable to A commitment transaction. However, for each previous commitment transaction, party B holds two corresponding penalty transactions that award the collateral of the watchtower to B. Therefore, as soon as B goes online (within the time window T), B will publish the suitable penalty transaction on-chain (type 2 if the reclaim transaction is on-chain, type 1 otherwise) and claim the watchtower's collateral, $c > a + b$ coins. Note that this argument holds because the watchtower's collateral is locked on-chain involving the hiring party and thus cannot be used in parallel for other channels/parties.

(iii) In the last case, B colludes with watchtower W. This is the simplest case, since A is either online or employs its own watchtower. If A is online the security holds similarly to the Lightning protocol, i. e., A publishes the revocation transaction on-chain and receives all the funds of the channel. If A employs a watchtower, then the previous analysis holds. □

Lemma 4.3. *A* Cerberus *channel will not close in a state that is not the last state agreed by all the participants of the channel.*

Proof. Any party of a Cerberus channel cannot gain more profit by deviating from the honest protocol execution, because all parties involved in a Cerberus channel always maintain (at least) their funds as shown in Lemma 4.2. Hence, a rational party that aims to maximize its profit will honestly follow the Cerberus protocol in every phase, and ultimately close the Cerberus channel in a non-cheating state, as described in Section 4.3. □

Lemma 4.4. *Any party of a* Cerberus *channel can close it at any time.*

Proof. Both parties of the Cerberus channel hold at least one valid commitment transaction (the one created at the last execution of the Update protocol, or if no update has taken place, the unique commitment transaction created during the Open protocol) which allows them to initiate phase *Close*, unilaterally, as described in Section 4.3.3. Hence, a Cerberus channel will be closed at the latest within time t from publishing a commitment transaction on-chain (given a perfect underlying blockchain protocol and network synchrony). □

Theorem 4.5. Cerberus *channels achieve security in our system model.*

Proof. Lemma 4.4 states that any party can close a Cerberus channel at any time. From Lemma 4.3, it holds that a Cerberus channel can only close on the last agreed by all parties state. Hence, any party of a Cerberus channel can enforce the last agreed state on-chain at any time. □

4.4.2 Performance

Next, we show that Cerberus channels scale well, meaning that the number of on-chain transactions is constant and independent of the number of transactions executed in a channel, similarly to the Lightning protocol. The analysis below considers a single watchtower for each party of the channel. We discuss the scalability of the protocol if we enable multiple watchtowers in Section 4.5.

Theorem 4.6. Cerberus *channels achieve scale-out in our system model.*

Proof. In phase *Open*, a Cerberus channel requires 3 transactions, one funding transaction to open the channel and two collateral transactions for each watchtower to lock the collateral to the corresponding party of the channel.

Phase *Update* is executed off-chain, hence no transactions are published on-chain.

During phase *Close*, the number of on-chain transactions can vary, however in the worst case 4 transactions are published. Specifically, in case of a non-cheating closure 3 transactions are published: (i) either a commitment transaction published unilaterally by a party of the channel, or a collaborative closing transaction published by both parties, (ii)-(iii) one reclaim transaction for each of the parties' watchtowers, published by the corresponding watchtowers respectively.

On the contrary, in case of fraud and a responsive watchtower, the following 4 transactions are necessary: (i) an old commitment transaction published by the cheating party (step 1, Protocol 5), (ii) the reclaim transaction of the cheating party's watchtower, published by the watchtower, (iii) the revocation transaction from the cheated party's watchtower, published by the watchtower (step 2, Protocol 5), and (iv) the reclaim transaction of the cheated party's watchtower, published by the watchtower (step 3, Protocol 5).

Furthermore, in case of fraud and an unresponsive watchtower, up to 4 transactions are published on-chain, as illustrated in Protocol 6: (i) an old commitment transaction published by the cheating party, (ii) the reclaim transaction of the cheating party's watchtower (published by the watchtower), (iii) the reclaim transaction of the cheated party's watchtower (published by the watchtower), and (iv) the corresponding penalty transaction published by the cheated party.

Phase *Abort* is an optional intermediate phase, that allows the watchtower to withdraw its service to the channel party. This phase includes one on-chain transaction – the reclaim transaction – but not additively to phase *Close*. After phase *Abort* the protocol for the party is similar to the Bitcoin Lightning protocol, which requires at most 3 on-chain transactions in total for a channel operation. Thus, the worst case performance analysis does not include phase *Abort*.

Overall, a Cerberus channel requires at most 7 on-chain transactions. □

4.5 Conclusion, Limitations and Extensions

Are incentivized watchtowers at all possible in a system like Bitcoin?

Summary. In this chapter, we answered this question affirmatively, by introducing Cerberus channels, an extension of Lightning channels. Cerberus channels reward watchtowers while remaining secure against bribing and collusion; thus participants can safely go offline for an extended period of time. We defined the desired properties for payment channel solutions and proved Cerberus channels are correct under our security model.

Privacy. Cerberus channels maintain the privacy property, as defined in Section 4.2.3, assuming that the watchtowers are considered internal to the channel parties. This means that the transactions executed off-chain during the phase *Update* are known only to the parties of the channel and the hired watchtowers. Any other third-party, external to the channel, does not have any knowledge on the state of the channel. Nevertheless, Cerberus channels do not guarantee privacy from the watchtowers. Although Lightning watchtowers preserve the privacy of transactions from watchtowers, they also suffer from inadequate incentives for participation in the system. On the other hand, Cerberus channels provide the necessary incentive mechanisms to guarantee security, but watchtowers are aware of all transactions executed in the channel. Introducing stronger privacy mechanisms while maintaining the appropriate incentives is left for future work.

Extension to multiple watchtowers. To enhance security against possible crash failures or withdrawal of service of a watchtower, parties can employ multiple watchtowers. In such a case, the number of on-chain transactions grows linearly with the number of hired watchtowers. Every watchtower needs to lock its collateral on-chain, thus one collateral transaction per watchtower is needed. However, the sum of all watchtowers' collateral remains the same, to guarantee security; that is at least greater than the total funds locked in the channel by both parties. This way the counterparty of the channel cannot bribe all watchtowers since the sum of the required bribes will exceed the party's potential gain. Therefore, there will be at least one watchtower that will publish the revocation transaction in case of fraud.

Rewards and collateral. Currently, in a Cerberus channel, rewards are awarded to the watchtowers on every update of the channel by the hired party via a unidirectional channel. Ideally, these rewards should be returned to the hired party if the watchtower misbehaves. To this end, we can modify the collateral transaction to build a bidirectional payment channel in which both the watchtower

locks collateral and the hiring party locks future rewards. Then, during an update of the Cerberus channel and upon receiving the signed penalty transaction, the hiring party can update the distribution of funds in this channel, rewarding the watchtower for its active participation. Note that this process does not require a fair exchange protocol since the watchtower can simply withdraw its service in case the hiring party does not pay the reward, similarly to the current protocol. However, this modification implies that the watchtowers' rewards will be locked during the lifetime of the channel.

Assumptions. In every channel construction that the counterparty is allowed to publish on-chain a valid outdated state, timelocks are necessary to secure the construction. Assuming distrusting parties (including the watchtower), this implies that every party must be online once in a while to verify the correct operation of the construction. Hence, at best we can alleviate the availability assumption, but not abolish it completely. In turn, due to timelocks, the security of Cerberus channels depends on synchrony assumptions and a perfect blockchain that cannot be censored. Although we enable shorter revocation periods and parties can go offline for an extended period of time, we cannot decouple the dependency of the security of payment channels on Bitcoin from the liveness and synchrony of the underlying blockchain.

Chapter 5

Brick: Asynchronous Payment Channels

5.1 Introduction

5.1.1 Motivation

Payment channels, although fast and efficient, are only secure as long as all parties of the channel are frequently – at least once in t time – online and monitoring the blockchain. This is problematic in real networks [74, 75], as one party may simply execute a denial-of-service (DoS) attack on the other party. To add insult to injury, the dispute period t is public; the attacking party hence knows the exact duration of the denial-of-service attack.

The issue is well-known in the community, and there were solution attempts using semi-trusted third parties called *watchtowers* [71–73, 76, 109], as explained in Section 2.6.3. The idea is that worrisome channel parties can hire watchtowers that watch the blockchain on their behalf in case they were being attacked. So instead of DoSing a single machine of the channel partner, the attacker might need to DoS the channel partner as well as its watchtower(s). This certainly needs more effort as the adversary must detect a watchtower reacting and then block the dispute from appearing on-chain. However, if large amounts of money are in a channel, it will easily be worth the investment.

While DoS attacks are also possible in blockchains such as the Bitcoin blockchain, DoS attacks on channels have a substantially different threat level. A DoS attack on a blockchain is merely a liveness attack: One may prevent a transaction from entering the blockchain at the time of the attack. However, the parties involved with the transaction will notice this, and can simply re-issue their transaction later. A DoS attack on a channel, on the other hand, will steal all the funds that were in the channel. Once the fraudulent transaction is in the blockchain, uncontested for t time, the attack succeeds, and nobody but the cheated party (and its watchtowers) will know any better.

Channels need a more fundamental solution. Not unlike blockchains, introducing timing parameters is acceptable for liveness. Security on the other hand should be guaranteed even if the network behaves completely asynchronously.

To that end we introduce Brick, a novel incentive-compatible payment channel construction that does not rely on timing assumptions for the delivery of messages to be secure. Brick provides proactive security, detecting and preventing fraud before it appears on-chain. As a result, Brick can guarantee the channels' security even under censorship [74, 75] or any liveness attack.

To achieve these properties, Brick needs to address three key challenges. The first challenge is how to achieve this proactive check without using a single trusted third party that approves every transaction. The core idea of Brick is to provide proactive security to the channel instead of reactive dispute resolution. To this end, Brick employs a group of *wardens*. If there is a dispute, the wardens make sure the correct state is the only one available for submission on-chain, regardless of the amount of time it takes to make this final state visible.

The second challenge for Brick is cost. To simulate this trusted third party, it would need the wardens to run costly asynchronous consensus [91] for every update. Instead, in Brick we show that a light-weight consistent broadcast protocol is enough to preserve both safety and liveness.

A final challenge of Brick that we address is incentives. While the wardens may be partially byzantine, we additionally want honest behavior to be their dominant strategy. Unfortunately, existing watchtower solutions do not align the expected and rational behavior of the watchtower, hence a watchtower is reduced to a trusted third party. Specifically, Monitors [71], Watchtowers [76], and DCWC [72] pay the watchtower upon fraud. Given that the use of a watchtower is public knowledge, any rational channel party will not commit fraud and hence the watchtower will never be paid. Therefore, there is no actual incentive for a third party to offer a watchtower service. On the other hand, Pisa [73] pays the watchtower regularly every time a transaction is executed on the channel. The watchtower also locks collateral on the blockchain in case it misbehaves. However, Pisa's collateral is not linked to the channel or the party that employed the watchtower. Hence, a watchtower that is contracted by more than one channel can double-assign the collateral, making Pisa vulnerable to bribing attacks. Even if the incentives of Pisa get fixed, punishing a misbehaving watchtower in Pisa is still a synchronous protocol (though for a longer period). In Brick, we employ both rewards and punishment to design the appropriate incentives such that honest and rational behavior of wardens align, while no synchrony assumptions are required, i. e., the punishment of misbehaving wardens is not conditional on timing assumptions.

We also present Brick+, a channel construction suitable for regulated environments such as supporting a fiat currency [110]. Brick+ needs to additionally provide auditability of transactions without forfeiting privacy [108] and meanwhile

preserve the accountability of the auditor. To resolve this, we change Brick in two ways. First, the state updates are no longer just private but also interconnected in a tamper-evident hash-chain. Essentially, the wardens maintain a single hash (constant-size storage cost) which is the head of the hash-chain of the state history. Second, to provide accountability for the auditor, we require that the auditor posts the access request on-chain [111]. Only then will the wardens provide the auditor the necessary metadata to verify the state history received from the parties of the channel.

5.1.2 Contribution

In summary, this chapter makes the following contributions:

- We introduce Brick, the first incentive-compatible off-chain construction that operates securely with offline channel participants under full asynchrony with sub-second latency.
- We introduce Brick+, a modification that enables external auditors to lawfully request access to the channels history while maintaining privacy.
- We define the desired channel properties and show they hold for Brick under a hybrid model of rational and byzantine participants (channel parties and wardens). Specifically, we present elaborate incentive mechanisms (rewards and punishments) for the wardens to maintain the channel properties under collusion or bribing.

5.2 Protocol Overview

5.2.1 System Model

Cryptographic assumptions. We make the usual cryptographic assumptions: the participants are computationally bounded and cryptographically-secure communication channels, hash functions, signatures, and encryption schemes exist.

Blockchain assumptions and network model. We assume that any message sent by an honest party will be delivered to any other honest party within a polynomial number of rounds. We do not make any additional assumptions about the network (e.g., known bounds for message delivery). Furthermore, we do not require a "perfect" blockchain system since Brick can tolerate temporary liveness attacks. Specifically, if an adversary temporarily violates the liveness property of the underlying blockchain, this may result in violating the liveness property of channels but will not affect the safety. Nevertheless, we assume the underlying

blockchain satisfies persistence [13]. In Section 5.6, we discuss a modification of Brick that is safe even when persistence is temporarily violated.

Threat model. We initially assume that at least one party in the channel is honest to simplify the security analysis. However, later, we show that the security analysis holds as long as the "richest" party of the channel is rational and intentionally deviates from the protocol only if it can increase its profit (utility function). Regarding the committee, we assume that there are at most f out of $n = 3f + 1$ byzantine wardens, and we define a threshold $t = 2f + 1$ to achieve the liveness and safety properties. The non-byzantine part of the committee is assumed rational; we first prove the protocol goals for t honest wardens, and subsequently align the rational behavior to this through incentives.

5.2.2 Brick Overview

Both parties of a channel agree on a committee of wardens before opening the channel. The wardens commit their identities on the blockchain during the funding transaction of the channel (opening of the channel). After opening the channel on the blockchain, the channel can only be closed either by a transaction published on the blockchain and signed by both parties or by a transaction signed by one of the parties and a threshold (t) of honest wardens. Thus, the committee acts as power of attorney for the parties of the channel. Furthermore, Brick employs correct incentives for the t rational wardens to follow the protocol, hence it can withstand $t = 2f + 1$ rational and f byzantine wardens, while the richest channel party is assumed rational and the other byzantine.

A naive solution would then instruct the committee to run asynchronous consensus on every new update, which would cost $O(n^4)$ [91] per transaction, a rather big overhead for the critical path of our protocol[1]. Instead in Brick, consensus is not necessary for update transactions, as we only provide guarantees to rational parties (if both parties misbehave one of them might lose its funds). As a result, every time a new update state occurs in the channel (i.e., a transaction), the parties run a consistent broadcast protocol (cost of $O(n)$) with the committee. Specifically, a party announces to each warden that a state update has occurred. This announcement is a monotonically increasing sequence number to guarantee that the new state is the freshest state, signed by both parties of the channel to signal that they are in agreement. If the consistent broadcast protocol succeeds (t wardens acknowledge reception) then this can serve as proof for both parties that the state update is safe. After this procedure terminates correctly, both parties proceed to the execution of the off-chain state.

At the end of the life-cycle of a channel, a dispute might occur, leading to

[1]For this step, we would require a fully asynchronous consensus protocol without a trusted setup. To the best of our knowledge, [91] has the lowest communication complexity of $O(n^4)$.

the unilateral closing of the channel. Even in this case, we can still guarantee the security and liveness of the closing with consistent broadcast. The crux of the idea is that if $2f + 1$ wardens accepted the last sequence number before receiving the closing request (hence the counterparty has committed), then at least one honest warden will not accept the closing at the old sequence number. Instead, the warden will reply to the party that it can only close at the state represented by the last sequence number. As a result we define a successful closing to be at the maximum of all proposed states, which guarantees safety. *Although counter-intuitive, this closing process is safe because the transactions are already totally ordered and agreed to by the parties of the channel;* thus, the committee simply acts as shared memory of the last sequence number.

5.2.3 Brick+ Overview

Brick+ is designed to enable payment channels in a permissioned, regulated setting, for example, a centrally-banked cryptocurrency. In such a setting, there will be an auditor (e. g. the IRS) that can check all the transactions inside a channel as these transactions might be taxable or illegal. This is a realistic case as the scalability in payment channels comes from a persistent relationship that models well B2B and B2C relationships that are usually taxed. In this setting, we assume that the auditor can punish the parties and the committee externally of the system, hence our goal is to enhance transparency even if misbehavior is detected retroactively.

In order to convert Brick into Brick+ we must ensure that (a) nothing happens without the committee's approval and (b) a sufficient audit trail is left on-chain to stop regulators from misbehaving. We resolve the first issue by disabling the parties' ability to close the channel without the participation of the committee. For the second challenge, the parties generate a hashchain of (blinded) states and sequence numbers while the wardens always remember the head of the hashchain together with the last sequence number. To prevent the auditor from misbehaving, we force the auditor to post a "lawful access request" [112] on-chain to convince the channel parties to initiate the closing of the channel for audit purposes and send the state history to the auditor.

5.2.4 Reward Allocation & Collateral

To avoid bribing attacks, we enforce the wardens to lock collateral in the channel. The total amount of collateral is proportional to the value of the channel meaning that if the committee size is large, then the collateral per warden is small. More details on the necessary amount of collateral are thoroughly discussed in Sections 5.3.2 and 5.6. Additionally, the committee is incentivized to actively participate in the channel with a small reward that each warden gets when they acknowledge a state update of the channel. This reward is given with a unidirectional

channel [108], which does not suffer from the problems BRICK solves. Moreover, the wardens that participate in the closing state of the channel get an additional reward, hence the wardens are incentivized to assist a party when closing in collaboration with the committee is necessary.

5.2.5 Protocol Goals

To define the goals of BRICK, we first need to define the necessary properties of a channel construction. Intuitively, a channel should ensure similar properties with a blockchain system, i. e., a party cannot cheat another party out of its funds, and any party has the prerogative to eventually spend its funds at any point in time. The first property, when applied to channels, means that no party can cheat the channel funds of the counterparty, and is encapsulated by *Safety*. The second property for a channel solution is captured by *Liveness*; it translates to any party having the right to eventually close the channel at any point in time. We say that a channel is *closed* when the locked funds of the channel are spent on-chain, while a channel *update* refers to the off-chain change of the channel's state.

In addition, we define *Privacy* which is not guaranteed in many popular blockchains, such as Bitcoin [1] or Ethereum [16], but constitutes an important practical concern for any functional monetary (cryptocurrency) system. Furthermore, we define another optional property, *Auditability*, which allows authorized external parties to audit the states of the channel, thus making the channel construction suitable for use on a regulated currency. The first three properties are met by BRICK, while the latter is only available in BRICK+.

First, we define some characterizations on the state of the channel, namely, validity and commitment. Then, we define the properties for the channel construction. Each state of the channel has a discrete sequence number that reflects the order of the state. We assume the initial state of the channel has sequence number 1 and every new state has a sequence number one higher than the previous state agreed by both parties. We denote by s_i the state with sequence number i.

Definition 5.1. *A state of the channel, s_i, is **valid** if the following hold:*

- *Both parties of the channel have signed the state s_i.*
- *The state s_i is the **freshest** state, i. e., no subsequent state s_{i+1} is valid.*
- *The committee has not invalidated the state. The committee can invalidate the state s_i if the channel closes in the state s_{i-1}.*

Definition 5.2. *A state of the channel is **committed** if it was signed by at least $2f + 1$ wardens or is valid and part of a block in the persistent[2] part of the blockchain.*

[2]The part of the chain where the probability of fork is negligible hence there is transaction finality, e. g., 6 blocks in Bitcoin.

Definition 5.3 (**Safety**). *A* BRICK *channel will only close in the freshest committed state.*

Definition 5.4 (**Liveness**). *Any valid operation (update, close) on the state of the channel will eventually*[3] *be committed (or invalidated).*

Definition 5.5 (**Privacy**). *No unauthorized*[4] *external (to the channel) party learns about the state of the channel (e. g., the current distribution of funds between the parties of a payment channel) unless at least one of the parties initiate the closing of the channel.*

Definition 5.6 (**Auditability**). *All committed states of the channel are verifiable by an authorized third party.*

5.3 BRICK Design

In this section, we first present the BRICK architecture assuming t honest wardens, and then introduce the incentive mechanisms aligning honest and rational behavior.

5.3.1 Architecture

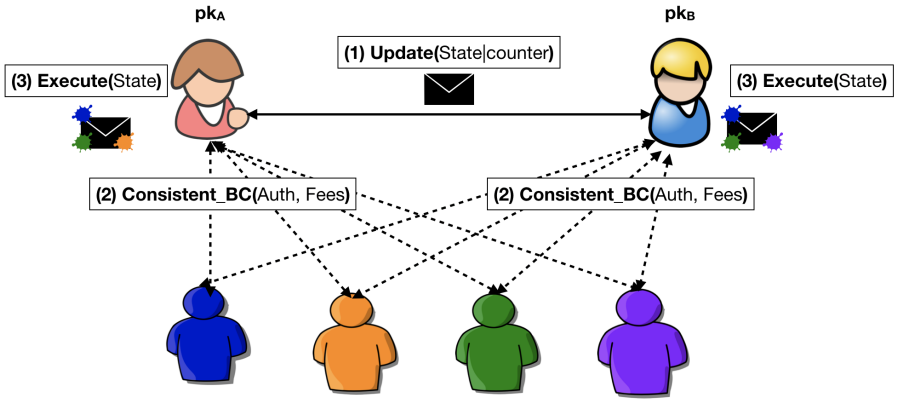


Figure 5.1: Typical workflow of BRICK for a state update. (1) Alice and Bob agree on a new state update. (2) They individually broadcast that they reached agreement on a new state to the committee (with an associated fee). (3) When a threshold of the committee replies that the state is committed, each sender (Alice or Bob) executes this update as persistent.

BRICK consists of three phases: *Open*, *Update*, and *Close*. We assume the existence of a smart contract that has two functions, Open and Close, which

[3] Depending on the message delivery.
[4] Authorized parties are potential auditors of the channel, as described in BRICK+.

receive the inputs of the protocols and verify that they adhere to the abstractly defined protocols specified below.

Protocol 7 describes the first phase, *Open*, which is the opening of a channel between two parties. In this phase, the parties create the initial funding transaction, similarly to other known payment channels such as [11, 12]. However, in Brick we also define two additional parameters in the funding transaction: the hashes of the public keys of the wardens of the channel, denoted by $W_1, W_2, \ldots, W_n$, and the threshold t.

Protocol 7: Brick Open

Data: Parties A, B, wardens $W_1, \ldots, W_n$, initial state s_1.
Result: Open a Brick payment channel.

/* The parties agree on the first update before opening the channel */
1. Register to $\{M, \sigma(M)\}$ the announcement of Protocol 8 on input (A, B, s_1).

/* The parties broadcast the first sequence number to the wardens */
2. Execute Protocol 9 on input $(M, \sigma(M), A, B, W_1, W_2, \ldots, W_n)$.
// without an update fee

/* The parties open the Brick channel */
3. Both parties A, B sign and publish on-chain $open(H(W_1), H(W_2), \ldots, H(W_n), t, s_1)$.
/* A closing fee F is included in the funding transaction, as well as the collateral C of each warden along with their signature. */

The second phase, *Update*, consists of two protocols, Protocol 8 (Update), and Protocol 9 (Consistent Broadcast). Both algorithms are executed consecutively every time an update occurs, i. e., when the state of the channel changes. In Protocol 8, the parties of the channel agree on a new state and create an announcement, which they subsequently broadcast to the committee with Protocol 9. To agree on a new state, both parties sign the hash of the new state[5]. This way both parties commit to the new state of the channel, while none of the parties can unilaterally close the channel without the collaboration of either the counterparty or the committee. The announcement, on the other hand, is the new sequence number signed by both parties of the channel[6]. The signed sequence number allows the wardens to verify agreement has been reached between the channel parties on the new state, while the state of the channel remains private. Upon receiving a valid announcement from a party, wardens reply with their signature

[5] Blinding the commitment to the state is not necessary for Brick, but we do it for compatibility with Brick+.

[6] We abuse the notation of signature σ to refer to the multisig of both A and B.

on the announcement. A party executes the new state update when it receives t signatures from the wardens.

Protocol 8: Brick Update

Data: Parties A, B, current state s.
Result: Create announcement $M, \sigma(M)$ (sequence number of new state signed by both parties).

1. Both parties A, B sign, exchange, and store: $\{H(s_i, r_i), i\}$, where r_i is a random number and s_i the current state. `// The parties store only the current and previous hash`

2. Upon receiving the signature of the counterparty on $\{H(s_i, r_i), i\}$, a party replies with its signature on the sequence number $\sigma(i)$.
`// creating the announcement` $\{M, \sigma(M)\}(M = i)$

Protocol 9: Brick Consistent Broadcast

Data: Parties A, B, wardens $W_1, \ldots, W_n$, announcement $\{M, \sigma(M)\}$.
Result: Inform the committee of the new update state and verify the validity of the new state.

1. Each party broadcasts to all the wardens $W_1, W_2, \ldots, W_n$ the announcement $\{M, \sigma(M)\}$. `// alongside a fee` r

2. Each warden W_j, upon receiving $\{M, \sigma(M)\}$, verifies that both parties' signatures are present, and the sequence number is exactly one higher than the previously stored sequence number. If the warden has published a closing state, it ignores the state update. Otherwise, W_j stores the announcement $\{M, \sigma(M)\}$ (replacing the previous announcement), signs M, and sends the signature $\sigma_{W_j}(M)$ to the parties. `// only to the parties that payed the fee`

3. Each party, upon receiving at least t signatures on the announcement M, considers the state committed and proceeds to the state transition.

The last phase of the protocol, *Close*, can be implemented in two different ways: the first is similar to the traditional approach for closing a channel (Protocol 10: Optimistic Close) where both parties collectively sign the freshest state (closing transaction) and publish it on-chain. However, in case a channel party is not responding to new state updates or closing requests, the counterparty can unilaterally close the channel in collaboration with the committee of the channel (Protocol 11: Pessimistic Close).

In Protocol 11, a party requests from each warden its signature on the last committed sequence number. A warden, upon receiving the closing request, publishes on-chain a closing announcement, i. e., the stored sequence number signed along with a flag close. When t closing announcements are on the persistent part of the chain, the party recovers the state that corresponds to the maximum sequence number from the closing announcements s_i. Then, the party publishes state s_i and the random number r_i along with the signatures of both parties on the corresponding hash and sequence number $\sigma(H(s_i, r_i), i)$ on-chain. As soon as these data are included in a (permanent) block, the Brick smart contract performs the following operations: (a) recovers from the submitted state s_i and salt r_i the hash $H(s_i, r_i)$ and the maximum sequence number i, (b) verifies that the signatures of both parties are on the message $\{H(s_i, r_i), i\}$, and (c) there are t submitted announcements that correspond to warden identities committed on-chain in Protocol 7. If all verifications check the smart contract closes the channel in the submitted state s_i.

Protocol 10: Brick Optimistic Close

Data: Parties A, B, state s.
Result: Close a channel on state s, assuming both parties are responsive and in agreement.

1. A party $p \in \{A, B\}$ broadcasts the request $close(s)$.

2. Both parties A, B sign the state s (if they agree) and exchange their signatures.

3. The party p (or any other channel party) publishes the signed by both parties state, $\sigma_{A,B}(s)$ on-chain.

`/* The collateral` C `is returned to each warden */`
`/* The closing fee` F `is returned to the parties */`

5.3.2 Incentivizing Honest Behavior

Brick actually works without the fees, if we assume one honest party and t honest wardens. However, our goal is to have no honest assumptions and instead align rational behavior to honest through incentives. There are three incentive mechanisms in Brick:

Update Fee (r). The parties establish a unidirectional channel [108] with each warden and send a small payment every time they want a signature for a state update. Note that the update fee is awarded to the wardens at step 1 of Protocol 9.

Protocol 11: BRICK Pessimistic Close

Data: Party $p \in \{A, B\}$, wardens $W_1, \ldots, W_n$, state s_i, random nonce r_i.
Result: Close a channel on state s_i with the assist of the committee.

1. Party p broadcasts to the wardens $W_1, W_2, \ldots, W_n$ the request $close()$.

2. Each warden W_j publishes on-chain a signature on the (last) stored announcement $\sigma_{W_j}(M, close)$ and stops signing new state updates.

3. Party p, upon verifying t on-chain signed announcements by the wardens, recovers the $max(i)$ that is included in the announcements. Then, party p publishes on-chain the state s_i, the random number r_i, and the signature of both parties on $\{H(s_i, r_i), i\}$.

4. After the state is included in a (permanent) block, the smart contract recovers $\{H(s_i, r_i), i\}$, verifies both parties' signatures and the wardens identities, and then closes the channel in state s_i.

Closing Fee (F). During phase *Open* (Protocol 7), the parties lock a closing fee F in the channel. If a party closes in a collaboration with the wardens, the closing fee is split only among the first t wardens that publish an announcement on-chain (see Protocol 12). If the channel closes optimistically (Protocol 10), the closing fee returns to the parties.

Collateral (C). During phase *Open*, each warden locks collateral C at least equal to the amount locked in the channel v divided by f. If a warden misbehaves, the closing party can claim the warden's collateral by submitting a proof-of-fraud in the BRICK smart contract during phase *Close*; otherwise, the collateral is returned to the warden when the channel closes (Protocol 12). A proof-of-fraud consists of two conflicting messages signed by the same warden: (a) a signature on an announcement on a state update of the channel, and (b) a signature on an announcement for closing on a previous state of the channel.

In case, a party submits $x \leq f$ proofs-of-fraud, the closing process is extended until $x + t$ wardens have published an announcement on-chain. Then, the channel closes in the state with the maximum sequence number from the announcements submitted by the t non-cheating wardens. On the other hand, if a party submits at least $f + 1$ proofs-of-fraud, the party that submitted the proofs-of-fraud claims only the collateral from the cheating wardens, while the entire channel balance is awarded to the counterparty. If no proofs-of-fraud are submitted the channel closes as described in Protocol 11, as it is a subcase of Protocol 12 for $x = 0$.

We further demand that the size of the committee is at least $n > 7$, hence

Protocol 12: BRICK Pessimistic Close with Incentives

Data: Party $p \in \{A, B\}$, wardens $W_1, \ldots, W_n$, state s_i, random nonce r_i.
Result: Close a channel on state s_i with the assist of the committee.

`/* Similarly to Protocol 11 */`

1. Party p broadcasts to the wardens $W_1, W_2, \ldots, W_n$ the request $close()$.

2. Each warden W_j publishes on-chain a signature on the (last) stored announcement $\sigma_{W_j}(M, close)$ and stops signing new state updates.

`/* Closing party submits also proofs-of-fraud */`

3. Party p, upon verifying t on-chain signed announcements by the wardens, recovers the $max(i)$ that is included in the announcements. Then, party p publishes on-chain the state s_i, the random number r_i, the signature of both parties on $\{H(s_i, r_i), i\}$, and any proofs-of-fraud.

`/* Closing the channel with punishments */`

4. After the state is included in a (permanent) block, the smart contract recovers $\{H(s_i, r_i), i\}$, and verifies both parties' signatures, the wardens identities, and the proofs-of-fraud.

   (a) If the valid proofs-of-fraud $x \leq f$, the smart contract closes the channel as soon as $t + x$ wardens have published an announcement on-chain. The channel closes in the state with the maximum sequence number included in the announcements, s_i.
   `// Protocol 11 with` $t + x$ `wardens`

   (b) If the valid proofs-of-fraud $x \geq f + 1$, the smart contract closes the channel, and awards the entire channel balance to the counterparty.

The smart contract awards the collateral of cheating wardens to party p, and returns the collateral of all non-cheating wardens. The first t non-cheating wardens whose signature are published on-chain get an equal fraction of the closing fee F/t.

$f > 2$. As a result, we guarantee there is at least one channel party with locked funds greater than each individual warden's collateral, $\frac{v}{2} > \frac{v}{f}$. This restriction along with the aforementioned incentive mechanisms ensure resistance to collusion and bribing of the committee, meaning that following the protocol is the dominant strategy for the rational wardens.

We note that in a network with multiple channels, each channel needs to maintain a unique id which will be included in the announcement to avoid replay attacks. Otherwise, if there exist two channels with the same parties and

watchtowers, the parties can unjustly claim the watchtowers' collateral by using signed sequence numbers from the other channel, effectively violating safety.

5.4 Brick Analysis

We first prove Brick satisfies *safety* and *liveness* assuming at least one honest channel party and at least t honest wardens. Furthermore, we note that Brick achieves *privacy* even if all wardens are byzantine while the channel parties are rational. Then, we show that rational players (parties and wardens) that want to maximize their profit will follow the protocol specification, for the incentive mechanisms presented in Section 5.3.2. Essentially, we show that Brick enriched with the proposed incentive mechanisms is dominant-strategy incentive-compatible.

5.4.1 Security Analysis under one Honest Participant and t Honest wardens

Theorem 5.7. Brick *achieves safety in asynchrony assuming one byzantine party and f byzantine wardens.*

Proof. In Brick there are two ways to close a channel (phase *Close*), either by invoking Protocol 10 or by invoking Protocol 11. In the first case (Optimistic Close), both parties agree on closing the channel in a specific state (which is always the freshest valid state[7]). As long as this valid state is published in a block in the persistent part of the blockchain, it is considered to be committed. Thus, safety is guaranteed.

In the second case, when Protocol 11 (Pessimistic Close) is invoked, a party has decided to close the channel unilaterally in collaboration with the committee. The Brick smart contract verifies that the state is valid, i. e., the signatures of both parties are present and the sequence number of that state is the maximum from the submitted announcements. Given the validity of the closing state, it is enough to show that the channel cannot close in a state with a sequence number smaller than the one in the freshest committed state. This holds because even if the channel closes in a valid but not yet committed state with sequence number larger than the freshest committed state, this state will eventually become the freshest committed state (similarly to Protocol 10).

Let us denote by s_i the closing state of the channel. Suppose that there is a committed state s_k such that $k > i$, thus s_i is not the freshest state agreed by both parties. We will prove safety by contradiction. For the channel to close at s_i at least $t = 2f + 1$ wardens have provided a signed closing announcement,

[7] We assume that if the parties want to close the channel in a previous state, they will still create a new state similar to the previous one but with an updated sequence number.

and the maximum sequence number of these announcements is i. Otherwise the BRICK smart contract would not have accepted the closing state as valid. According to the threat model, at most $n - t = f$ wardens are byzantine, thus at least $f + 1$ honest wardens have submitted a closing announcement. Hence, none of the $f + 1$ honest wardens have received and signed the announcement of any state with sequence number greater than i. However, in phase *Update*, an update state is considered to be committed, according to Protocol 9 (step 3), if and only if it has been signed by at least $t = 2f + 1$ wardens. Since at most $n - (f + 1) = 3f + 1 - f - 1 = 2f < 2f + 1$ wardens have seen (and hence signed) the state s_k, the state s_k is not committed. Contradiction.

We note that safety is guaranteed even if both parties crash. This holds because a state update requires unanimous agreement between the parties of the channel, i. e., both parties sign the hash of the new state. □

Theorem 5.8. BRICK *achieves liveness in asynchrony assuming one byzantine party and f byzantine wardens.*

Proof. We will show that every possible valid operation is either committed or invalidated. There are two distinct operations: *close* and *update*. We say that operation close applies in a state s if this state was published on-chain either in collaboration of both parties (Protocol 10) or unilaterally by a channel party as the closing state (step 3, Protocol 11).

If the operation is close and not committed, either the parties did not agree on this operation (Optimistic Close), or a verification of the smart contract failed (Pessimistic Close). In both cases, the operation is not valid.

Suppose now the operation is close and never invalidated. Then, if it is an optimistic close, all the parties of the channel have signed the closing state since it is valid. Since at least one party is honest the transaction will be broadcast to the blockchain. As soon as the blockchain is live, the state will be included in a block in the persistent part of the blockchain, and thus, the state will be eventually committed. On the other hand, if it is a pessimistic close and not invalidated, the smart contact verifications were successful therefore the state was committed.

Suppose the operation is a valid update and it was never committed. Since the operation is valid and at least one party of the channel is honest, the wardens eventually received the state update (Consistent Broadcast). However, the new state was never committed, therefore at least $f + 1$ wardens did not sign the update state. We assumed at most f byzantine wardens, hence at least one honest warden did not sign the valid update state. According to Protocol 9 (line 2), an honest warden does specific verifications and if the verifications hold the warden signs the new state. Thus, for the honest warden that did not sign, one of the verifications failed. If the first verification fails, then a signature from the parties of the channel is missing thus the state is not valid. Contradiction. The second

verification concerns the sequence number and cannot fail, assuming at least one honest channel party. Thus, the warden has published previously a closing announcement on-chain and ignores the state update. In this case, either (a) the closing state of the channel is the new state - submitted by another warden that received the update before the closing request - or (b) the closing state had a smaller sequence number from the new state. In the first case (a), the new state is committed eventually (on-chain), while in the second case (b) the new state is invalidated as the channel closed in a previous state.

For the last case, suppose the operation is a valid update and it was never invalidated. We will show the state update was eventually committed. Suppose the negation of the argument towards contradiction. We want to prove that an update state that is not committed is either not valid or invalidated. The reasoning of the proof is similar to the previous case. □

Lastly, Brick achieves *privacy even against byzantine wardens* since they only receive the sequence number of each state update in a channel. Therefore, as long as parties do not intentionally reveal any information to anyone external, privacy is maintained.

5.4.2 Incentivizing Rational Players

In this section, we show that rational players, parties and wardens, that want to maximize their profit follow the protocols, i. e., deviating from the honest protocol executions can only result in decreasing a player's expected payoff. Therefore, security and liveness hold from Theorems 5.7 and 5.8. Note that in our system model $2f + 1$ wardens and the richest party are rational, while the rest can be byzantine. We consider each protocol separately, and evaluate the players' payoff for each possible action.

Open. Naturally, if a party is not incentivized to open a channel then that channel will never be opened. We assume the parties have some business interest to use the blockchain and since transacting on channels is faster and cheaper they will prefer it. Deviating from the protocol at this phase is meaningless. Furthermore, we assume the wardens follow Protocol 7 and commit the requested collateral on-chain (Brick smart contract). Otherwise, the parties simply replace the unresponsive/misbehaving wardens.

Update. In phase *Update*, we analyze Protocol 8 for rational parties as the wardens do not participate in this protocol. Then, we analyze Protocol 9 for both parties and wardens.

Lemma 5.9. *Rational parties faithfully follow Protocol 8.*

Proof. During the execution of Protocol 8, any party can deviate from the protocol by not signing the hash of the new state. In this case, the new state will not be valid and thus cannot be committed and will not be executed. No party can increase its profit from such behavior directly (attacking the safety of the channel). The same argument holds in case the party signs the hash but not the sequence number. Moreover, attempting to attack the liveness of the channel is not profitable since the counterparty can always request to close in collaboration with the committee by invoking Protocol 12.

Lastly, channel parties can collude and stop updating the channel (liveness attack) in order to enforce a hostage situation on the wardens' collateral. However, the committee size is at least $n > 7$. Thereby, from the pigeonhole principle there is at least one party in the channel that has locked funds at least equal to $\frac{v}{2} > \frac{v}{f}$, i. e., the richest party of the channel. Thus, the richest channel party locks an amount larger than each warden's collateral which means that this party's cost is larger than a wardens. Since we assume the richest party is rational, at any time this party is incentivized to close the channel in collaboration with the committee. Thus, the richest party (if rational) will not deviate from the honest execution of Protocol 8. □

Lemma 5.10. *Rational parties faithfully follow Protocol 9.*

Proof. During the execution of Protocol 9, a party can deviate as follows:

(a) First, a party can choose not to broadcast the announcement to the committee or part of the committee. In this case, the party has signed the new state, which is now a valid state. This state will be considered committed for the counterparty after the execution of Protocol 9. We show a rational party cannot increase its payoff by not broadcasting the announcement to all wardens. To demonstrate this, we consider two cases; either the new state is beneficial to the party or not. If the new state is beneficial to party A, then this state is not beneficial for the counterparty (e. g., B payed A for a service). Thus, if the committee has not received the freshest state, party B can "rightfully" close the channel in the previous state. Hence, the expected payoff of party A decreases. On the other hand, suppose the new state is not beneficial to party A and it chooses not to send the announcement to the committee. Then, either the counterparty will have the state committed or the state will not be executed. From the safety property of the channels, party A cannot successfully close the channel in a previous state if the state was committed. Hence, party A does not increase its payoff. On the other hand, if party A does not request the signature of a warden that will later commit fraud, then the party cannot construct a proof-of-fraud to claim the warden's collateral and therefore the party's payoff decreases.

(b) Second, a party can broadcast different messages to the committee or parts of the committee. During the execution of Protocol 9, the wardens verify

the parties' signatures, thus an invalid message will not be acknowledged from an honest warden. If the messages are valid (both parties' signatures are present), the parties have misbehaved in collaboration. This can lead to a permanent partition of the view of the committee regarding the state history, but at most one of the states can be committed (get the $2f + 1$ signatures). Thus, this strategy has the same caveats as the previous one, where the party can only lose from following it.

(c) Lastly, the party can choose not to proceed to the state transition. This is outside the scope of this work and a problem of a different nature (a fair exchange problem). □

Lemma 5.11. *Rational wardens faithfully follow Protocol 9.*

Proof. A warden only acts in step 2 of Protocol 9, and can deviate as follows:

(a) The warden does not perform the necessary verifications (signatures and sequence number). Then, the warden might unintentionally commit fraud by signing an invalid state, hence allow the party to claim the collateral.

(b) The warden replies to the party although it has published a closing announcement on-chain. Then, the party can penalize the warden by claiming its collateral.

(c) The warden sends its signature on the new state but does not store the new announcement. In this case, if a party requests to close unilaterally the warden cannot participate and thence collect the closing fee. Consequently, the warden's expected payoff decreases.

(d) The warden ignores the party's request to update the state (does not reply to the party), since the update fee is already collected. At first sight, this game looks like a fair exchange game, which is impossible to solve without a trusted third party [113]. Furthermore, we cannot use a blockchain to solve it [114] as the whole point of channels is to reduce the number of transactions that go on-chain. Fortunately, the state update game is a repeated game where wardens want to increase their expected rewards in the long term. The wardens know that if they receive an update fee from a party and do not respond, then the party will stop using them (there is f fault tolerance in Brick), thus their expected payoff for the repeated game will decrease. □

Close. A channel can either close optimistically by both parties with Protocol 10, or unilaterally by one of the parties in collaboration with the wardens with Protocol 12. We first show that rational parties do not deviate from Protocol 10 (Lemma 5.12), and later we consider the most complicated case of Protocol 12.

Intuitively, a rational party invoking Protocol 12 always prefers to close the channel in the correct state and claim the wardens' collateral, when $C \geq v/f$, as they sum up to larger profits (Lemma 5.13). The rational wardens, therefore, will not collaborate with a cheating party but instead follow Protocol 11. As a result, Brick channels achieve safety and liveness with rational players (Theorem 5.16).

Lemma 5.12. *Rational parties faithfully follow Protocol 10.*

Proof. A party can deviate from Protocol 10 in the following ways:

(a) It is the party requesting the closing of the channel in a cheating state. The counterparty will not sign the state since it is being cheated, else it would not be a cheating state. Thus, safety is guaranteed and the party cannot profit from this strategy.

(b) It is the party requesting the closing of the channel and never publishes the signed closing state. In line 2 of Protocol 10, the signatures on the state are exchanged between the parties, hence the counterparty will eventually publish the closing state. Note that we assume that the closing party sends its signature first with the closing request.

(c) It is the party that got the closing request and does not sign the state. In this case, the party requesting to close the channel can invoke Protocol 12 and close the channel in collaboration with the committee in the freshest committed state. □

Lemma 5.13. *Rational parties invoking Protocol 12 maximize their profit when closing the channel in the freshest committed state.*

Proof. Let us denote by p_A the payoff function of party A (the cheating party wlog). The payoff function depends on the channel balance of party A when requesting to close the channel, denoted by c_A ($0 \geq c_A \geq v$, where v is the total channel funds), the collateral the party claims through proofs-of-fraud, and the total amount spent for bribing rational wardens. Formally,

$$p_A = c_A + x\frac{v}{f} - b\big(\frac{v}{f} + \epsilon\big)$$

where x is the number of proofs-of-fraud submitted by the party, b the number of bribed rational wardens, and ϵ the marginal gain over the collateral the wardens require to be bribed. Note that each warden has locked $\frac{v}{f}$ as collateral. Further, note that the byzantine wardens do not require a bribe but act arbitrarily malicious, meaning that they will provide a proof-of-fraud to party A without any compensation.

Next, we analyze all potential strategies for party A and demonstrate that the payoff function maximizes when the party does not bribe any rational warden, but closes the channel in the freshest committed state. There are four different outcomes in the strategy space of party A:

(a) The channel closes in the freshest committed state and no proofs-of-fraud are submitted. Then, $p_A = c_A$.

(b) Party A submits the proofs-of-fraud only from the byzantine wardens and the channel closes in the freshest committed state (by the remaining t rational warden). Then, $p_A = c_A + f\frac{v}{f} = c_A + v$. Note that the payoff function in this case maximizes for $x = f$.

(c) Party A bribes $b > 0$ rational wardens and the channel closes in the freshest committed state. Then, $p_A \leq c_A + (f+b)\frac{v}{f} - b(\frac{v}{f} + \epsilon) = c_A + v - b\epsilon \leq c_A + v - \epsilon$. The first inequality holds because the bribed wardens might be part of the second set of t closing announcements, in which case they do not contribute to the claimed collateral.

(d) Party A bribes $b > 0$ rational wardens and the channel closes in a state other than the freshest committed state. Let us denote by y the number of rational wardens that provide a proof-of-fraud to the party, and m the number of submitted proofs-of-fraud that belong to byzantine wardens. Then, the possible actions in this strategy are depicted in Table 5.1. Note

Table 5.1: Potential actions to close the channel in a "fraudulent" state.

Action	**Proof-of-fraud**	**Close**	**Total**
Byzantine	m	$f - m$	f
Bribed (rational)	y	$f + 1 - (f - m) = m + 1$	$y + m + 1$
Total	$m + y$	$f + 1$	-

that at least $f + 1$ misbehaving wardens are required to close the state in a previous state since we assume there can be f slow rational wardens that have not yet received the update states. In this case, the payoff function is

$$p_A \leq v + \big(m + y\big)\frac{v}{f} - \big(y + m + 1\big)\big(\frac{v}{f} + \epsilon\big)$$
$$= v - \frac{v}{f} - \epsilon\big(y + m + 1\big) \leq v - \frac{v}{f} - \epsilon$$

where the first inequality holds since $0 \geq c_A \geq v$. Therefore, the payoff function maximizes in case the party follows the second strategy, i. e., when the channel closes in the freshest committed state and no rational wardens are bribed. □

Using Lemma 5.13, we show that rational parties essentially follow Protocol 11 with respect to the rational wardens. That is, apart from the proofs-of-frauds submitted by the party for the byzantine wardens that "gift" their collateral to the closing party, in every other way, the selfish behavior of the closing party executing Protocol 12 aligns with the honest behavior of a party executing Protocol 11.

Lemma 5.14. *Rational parties are incentivized to faithfully follow Protocol 11 with respect to the rational wardens.*

Proof. The party that requested to close, invoking Protocol 11, can deviate from the protocol's specification in two ways: either (a) the party publishes an invalid closing state (e. g., random state, previously valid state, a committed state that is not the freshest), or (b) the party is not responsive, meaning that the party does not publish the closing state.

In case (a), publishing an invalid state can only decrease a party's profit, as shown in Lemma 5.13.

In case (b), either the party is the richest of the channel or not. If the party is the richest of the two then the party's cost of not responding is higher than that of the counterparty and any other warden's cost. Therefore, the party is not the richest of the channel. In this case, the counterparty which wants to close the channel, can use the on-chain closing announcements of the wardens and publish the closing state. In both cases, the party that requested close cannot increase its payoff by not revealing the closing state, but only lose from locking its channel funds for a longer period of time that necessary (since no updates are possible). □

Next, we show that assuming the smart contract executes Protocol 12 when closing the channel, the rational wardens will honestly follow Protocol 11; implying that the wardens will not commit fraud in collaboration with the closing party.

Lemma 5.15. *Rational wardens faithfully follow Protocol 11.*

Proof. A warden can deviate from the protocol as follows:

(a) The warden does not publish a closing announcement on-chain. In this case, the warden can attempt to enforce a hostage situation on the funds of the channel in collaboration with other wardens in order to blackmail the channel parties. However, to enforce a hostage situation on the channel's funds, at least $f + 1$ wardens must collude, hence at least one rational warden must participate. However, a rational warden cannot be certain that the other wardens will indeed maintain the hostage situation or participate in the consensus thus claim the closing fee (only the first t wardens get paid); therefore, we reduce our problem to the prisoner's dilemma problem. As a result, the only strong Nash equilibrium for a rational warden is to immediately publish a closing announcement on-chain in order to claim later the closing fee.

(b) The warden signs later a new state update. Then, the warden allows the party receiving the signed announcement with higher sequence number than

the closing announcement to create a proof-of-fraud and claim the warden's collateral.

(c) The warden publishes on-chain a closing announcement that is not the stored one[8]. However, the warden has already sent a signature on an announcement with a higher sequence number, therefore the party can create a proof-of-fraud and claim the warden's collateral. Hence, any rational warden will request as a bribe an amount at least marginally higher than the collateral to perform the fraud. In Lemma 5.13, we show that no rational party will provide such a bribe to any rational warden. Thus, all rational wardens will honestly follow the protocol and submit the stored announcement for closing the channel. □

Theorem 5.16. BRICK *channels achieve safety and liveness in asynchrony assuming the richest party and* $2f + 1$ *wardens are rational, while the rest are byzantine (wardens and other party).*

Proof. We showed that any rational player, party or warden, honestly follows Protocols 7, 8 (Lemma 5.9), 9 (Lemmas 5.10 and 5.11), 10 (Lemma 5.12), and 11 (Lemmas 5.14 and 5.15). Therefore, both safety and liveness are achieved in asynchrony in our system model from Theorems 5.7 and 5.8, respectively. □

5.5 BRICK+ Design

5.5.1 Architecture

BRICK+ consists, similarly to BRICK, of three phases, *Open*, *Update*, and *Close*. However, BRICK+ has one additional functionality, *Audit*, that allows an authorized third party to audit the states of a channel. Invoking this functionality though enforces the closing of the channel. The *Audit* functionality is illustrated in Figure 5.2 and described in detail in Protocol 13.

To enable the *Audit* functionality and therefore achieve *Auditability*, we disable Protocol 10 (Optimistic Close), and enforce the parties to close in collaboration with the committee. This way we guarantee that all states of the channel are available to the committee and hence to the potential auditor for verification. Moreover, we modify Protocol 8 and Protocol 9 (phase *Update*) such that the wardens store a hash-chain of the state history instead of the sequence number of the freshest valid state they received. Thereby, we ensure that the parties cannot present an alternate state history to the auditor as it achieves fork-consistency [115].

[8] We assume the warden will only sign as closing an announcement that used to be valid in an attempt to commit fraud. Otherwise, the party will claim the warden's when the smart contract verifications fail.

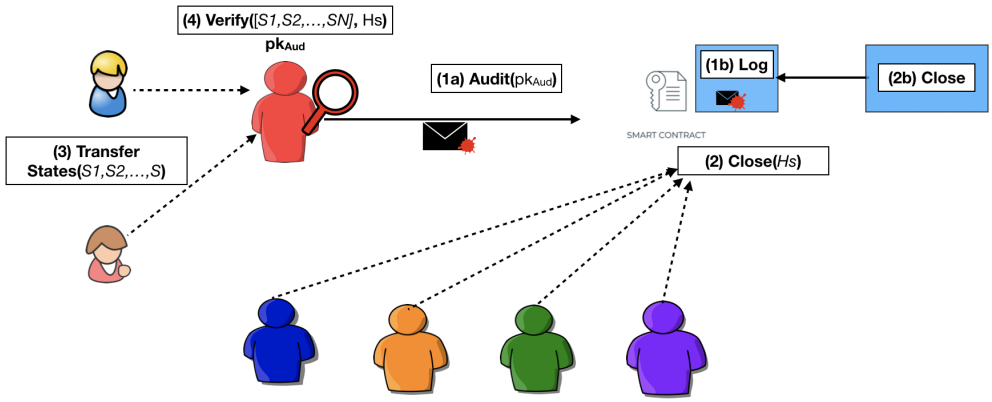


Figure 5.2: Typical workflow of Brick+ for an audit update. (1) The auditor starts the audit by posting the request on chain, (2) the committee closes the channel, and (3) the parties transfer the state to the auditor. Then (4), the committee posts the head of the hash-chain of the channel on-chain to claim their reward and finally (5) the auditor cross-checks the claims of the party and the committee.

Audit functionality. The *Audit* functionality is initiated by an authorized third party, the auditor, who publishes an access request on-chain. Then, the wardens of the channel verify the validity of the access request and initiate the closing of the channel by publishing the closing announcements (head of the hash-chain of the state history) on-chain (Protocol 12. After the execution of Protocol 12, the closing state is on-chain and both the (honest) wardens and parties of the channel have a consistent view of the channel history. Both parties send to the auditor the entire state history. The auditor then verifies the state history received from both parties by computing the hash-chain and comparing the last hash with the hash that corresponds to the maximum sequence number published by the wardens (i. e., the hash that corresponds to the closing state of the channel). If the parties misbehave and send an alternate state history the auditor can pursue external punishment (e. g., legal action).

Brick+ channel close. In Brick, the parties can close the channel in agreement at any time without the need for committee intervention (Protocol 10). However, Brick+ does not allow this function; a channel can only close in collaboration with the committee. Otherwise, the channel parties can update the channel and alter the history without accountability. In case of malicious behavior of the committee (which is outside the threat model of this work), we assume that the parties can alert the auditor who can pursue external punishment to enforce the wardens to be responsive.

Protocol 13: Audit

Data: Auditor A of channel c, audit smart contract with access on the information published on the blockchain, from Protocol 7.
Result: Audit of the channel.

1. The auditor A publishes on-chain the access request for channel c.

2. Each warden of the channel, upon verifying the validity of the access request, initiates the closing phase of the channel (Protocol 12), and publishes on chain the stored announcement (head of the hash-chain of state history).

3. After the execution of Protocol 12, the channel is closed on-chain in a committed state s. Then, both parties of the channel send to the auditor the state history.

4. The auditor collects from the blockchain the hashes published by the wardens and selects the hash that corresponds to the maximum sequence number (i. e., corresponds to the closing state s). Then, the auditor, upon receiving the state history from both parties, verifies it by re-creating the hash-chain. If a party does not respond to the access request or presents a different state history the auditor pursues external punishment.

5.5.2 BRICK+ Security Analysis

In this section, we prove the BRICK+ goals, namely *safety, liveness, privacy* and *auditability*, assuming f byzantine and $2f + 1$ rational wardens. Throughout this section, we assume the richest channel party is rational, thus it does not deviate from honest behavior if it will be discovered and punished. Furthermore, we assume the auditor is also rational, meaning that the auditor will only deviate from the protocol to gain more profit (hence the need for a smart contract to do the fair exchange between the auditor and the committee). However, the auditor will not punish a party arbitrarily with no proof, since the auditor is supposed to be an external trusted authority (e. g., judge, regulator, tax office etc.).

Theorem 5.17. BRICK+ *channels achieve safety and liveness in asynchrony assuming the richest party, the auditor, and* $2f + 1$ *wardens are rational, while the rest are byzantine (wardens and other party).*

Proof. Immediately follows from Theorem 5.16. □

Theorem 5.18. BRICK+ *achieves privacy assuming byzantine wardens and parties that want to maintain privacy.*

Protocol 14: Update for BRICK+

Data: Channel parties A, B, state s, previous head of hash chain H_p.
Result: Create announcement $\{M, \sigma(M), H_s\}$ (sequence number of new state and head of hash chain).

1. Both parties A, B sign and exchange: $\{H_s = H(H_p, H(s_i, r_i), i)\}$, where r_i is a random number and s_i the current state.

2. After receiving the signature of the counterparty on H_s, the party uses it to create the announcement $\{M, \sigma(M), H_s\}(M = i)$.

Proof. Suppose an external party learns about the state of the channel during the protocol execution. This means that either the external party intercepted a message (between the parties of the channel or between the parties and the committee) or it is a warden. In the first case, we assume secure communication channels thus a computationally-bounded adversary cannot get any information from the messages between the parties. In the latter case, the wardens receive during the *Update* phase a message with $H_s = H(H_p, H(s_i, r_i), i)$ for any valid state update (assuming honest parties that do not intentionally reveal the state). If the warden extracts the state of the channel s_i from the $H(s_i, r_i)$ (since it knows H_p), the warden reverted the hash function. Therefore, the hash function is not pre-image resistant for a computationally-bounded adversary and hence not cryptographically-secure. This contradicts the system model, where we assumed cryptographically-secure hash functions.

Furthermore, the audit request, which is the first step of the audit functionality, does not leak any information on the channel since the auditor is an external to the channel party. According to Protocol 13, the parties publish the closing state on-chain to close the channel (Protocol 12). Thus, privacy is preserved since by definition it is only guaranteed until at least one of the parties initiates the closing of the channel. □

Theorem 5.19. BRICK+ *achieves auditability in asynchrony assuming f byzantine wardens, while the rest ($2f + 1$ wardens, auditor, at least the "richest" party of the channel) are rational.*

Proof. Since both the wardens and a party of the channel are rational, Protocol 13 will not complete (a valid closing state will not be published) unless the request for access is valid, hence the auditor is authorized. Thus, to prove BRICK+ satisfies auditability, it is enough to prove that every committed state is verifiable by a third party.

Towards contradiction, suppose s is the earliest (least fresh) committed state that is not verifiable (since there is at least one). This means that state s is

replaced by another state s' either in the history of all parties or in the hash-chain that produced the hash-head corresponding to the closing state. If s is not part of the hash-chain, but it is committed, then the parties misbehaved and sent to at least one warden a different state. Note that the warden cannot misbehave since the announcement must have the signatures of both parties. Furthermore, two different states cannot be simultaneously committed since this would require two quorums of $2f + 1$ wardens that signed different state updates, thus at least $f + 1$ byzantine wardens. Contradiction to our threat model. Therefore, state s is not part of the state history provided by the parties. In both cases, the auditor punishes the parties (externally). And since we assume one party of the channel is rational (and the external punishment exceeds the potential gain of cheating), the party will not misbehave. Thus, every committed state is verifiable. □

5.6 Conclusion, Limitations and Extensions

Summary. In this chapter, we introduced Brick, the first payment channel that remains *secure under network asynchrony and concurrently provides correct incentives*. The core idea is to incorporate the conflict resolution process within the channel by introducing a rational committee of external parties, called wardens. Hence, if a party wants to close a channel unilaterally, it can only get the committee's approval for the last valid state. Brick provides sub-second latency because it does not employ heavy-weight consensus. Instead, Brick uses consistent broadcast to announce updates and close the channel, a light-weight abstraction that is powerful enough to preserve safety and liveness to any rational parties.

Furthermore, we considered permissioned blockchains, where the additional property of auditability might be desired for regulatory purposes. We introduced Brick+, an off-chain construction that provides auditability on top of Brick without conflicting with its privacy guarantees. We formally defined the properties our payment channel construction should fulfill, and proved that both Brick and Brick+ satisfy them. We also designed incentives for Brick such that honest and rational behavior aligns.

Below, we discuss the rationale of Brick design, its limitations, and possible extensions.

Byzantine players. If both channel parties are byzantine then the wardens' collateral can be locked arbitrarily long since the parties can simply crash forever. This is why in the threat model, we assume that at least the richest channel party is rational to correctly align the incentives. We further demand byzantine fault-tolerance to guarantee a truly robust protocol against arbitrary faults. We assume at most f out of the $3f + 1$ wardens are byzantine, which is necessary to maintain safety, as dictated by well known lower bounds for asynchronous

consistent broadcast. Nevertheless, users of Brick can always assume $f = 0$ and configure the smart contract parameters accordingly.

Warden unilateral exit. If both parties are malicious, they might hold the wardens' collateral hostage. A similar situation is indistinguishable from the parties not transacting often. As a result the wardens might want to exit the channel. A potential extension can support this in two ways. First, we can enable committee replacement, meaning that a warden can withdraw its service as long as there is another warden willing to take its place. In such a case, we simply replace the collateral and warden identities with an update of the funding transaction on-chain, paid by the warden that requests to withdraw its service. Second, if a significant number (e.g, $2f + 1$) of wardens declare they want to exit, the smart-contract can release them and convert the channel to a synchronous channel [73]. The parties will now be able to close the channel unilaterally by directly publishing the last valid state. If the counterparty tries to cheat and publishes an old state, the party (or any remaining warden) can catch the dispute on-time and additionally claim the (substantial) closing fee.

Committee selection. Each channel has its own group of wardens, i. e., the committee is independently selected for each channel by the channel parties. The scalability of the system is not affected by the use of a committee since each channel has its own independent committee of wardens. The size of the committee for each channel can vary but is constrained by the threat model. If we assume at least one honest party in the channel, a single rational warden is enough to guarantee the correct operation of Brick. Otherwise, we require more than 7 wardens to avoid hostage situations from colluding channel parties (Section 5.3.2). Note that the cost for security for the parties is not dependent on the committee size, but on the value of the channel. If the parties chose a small committee size, the collateral per warden is high, thus the update fees are few but high. On the other hand, if the parties employ many wardens, the collateral per warden is low, thus the update fees are many but low.

Consensus vs consistent broadcast. Employing consistent broadcast in a blockchain system typically implies no conflict resolution as there is no liveness guarantee if the sender equivocates. This is not an issue in channels since a valid update needs to be signed by both parties and we provide safety guarantees only to honest and rational parties[9]. The state updates in channels are totally ordered by the parties and each sequence number should have a unique corresponding state. Thereby, it in not the role of the warden committee to enforce agreement, but merely to verify that agreement was reached, and act as a shared memory for

[9]Of course if a party crashes we cannot provide liveness, but safety holds.

the parties. As a result, consistent broadcast is tailored for Brick as it offers the only necessary property, equivocation protection.

Brick Security under execution fork attacks. We can extend Brick to run asynchronous consensus [91] during the closing phase in order to defend against execution fork attacks [116]. This would add an one-off overhead during close but would make Brick resilient against extreme conditions [117]. For example, in case of temporary dishonest majority the adversary can attack the persistence of the underlying blockchain, meaning that the adversary can double-spend funds. Similarly in channels, if the adversary can violate persistence, the dispute resolution can be reversed, hence funds can be cheated out of a party. However, in Brick the adversary can only close on the last committed state or the freshest valid (not committed) state. With consensus during close, Brick maintains safety (i. e., no party loses channel funds) even when persistence is violated. A malicious party can only close the channel in the state that the consensus decides to be last, thus a temporary take-over can only affect the channel's liveness. Therefore, Brick can protect both against *liveness and persistence attacks*[10] on the underlying blockchain adding an extra layer of protection, and making it safer to transact on Brick than on the blockchain.

Update fees. Similarly to investing in stocks for a long period of time, many invest in cryptocurrencies; resulting in large amounts of unused capital. Acting as a warden can simply provide more profit (update fees) to the entities that own this capital complementary to owning such cryptocurrencies.

Currently, the update fees are awarded to wardens on every state update via a unidirectional channel. Ideally, these rewards would be included in the state update of the channel. But even if we include an increased fee on every state update, the parties can always invoke Optimistic Close, and update the channel state to their favor when closing. Thus, the incentives mechanism is not robust if the update rewards of the wardens are included in the state updates.

Collateral. The collateral for each warden in Brick is v/f, where v is the total value of the channel and f the number of byzantine wardens. This is slightly higher than the lowest amount $v/(f+1)$ for which security against bribing attacks is guaranteed in asynchrony when both channel parties and wardens are rational. Towards contradiction, we consider a channel where each warden locks collateral $C < v/(f+1)$. Suppose now a rational party p owns 0 coins in the freshest state and v coins in a previous state. Due to asynchrony, p controls the message delivery, hence f wardens may consider this previous state as the freshest one. Consequently, if p bribes $f+1$ wardens, which costs less than $(f+1)v/(f+1) = v$,

[10] We assume the channel to be created long before these attacks take place, so the adversary cannot fork the transaction that creates the channel.

the party profits from closing the channel in the previous state in collaboration with the bribed and "slow" wardens, violating safety.

In a synchronous network, this attack would not work since the other parties would have enough time to dispute. However, under asynchrony (or offline parties [73]) there is no such guarantee. Further, note that in a naive asynchronous protocol with f byzantine wardens, the previous attack is always possible for any collateral because a rational party can direct the profit from the collateral of the byzantine wardens to bribe the rational wardens. We circumvent this problem in Brick by changing the closing conditions.

Finally, a trade-off for replacing trust is highlighted: online participation with synchrony requirements or appropriate incentive mechanisms to compel the honest behavior of rational players.

Decentralization. In previous payment channel solutions a party only hires a watchtower if it can count on it in case of an attack. Essentially, watchtowers are the equivalent of insurance companies. If the attack succeeds, the watchtower should reimburse the cheated channel party [73]. After all, it is the watchtower's fault for not checking the blockchain when needed. However, in light of network attacks (which are prevalent in blockchains [118, 119]), only a few, centrally connected miners will be willing to take this risk. Brick provides an alternative, that proactively protects from such attacks and we expect to provide better decentralization properties with minimal overhead and fast finality.

Bitcoin compatibility. We believe Brick can be implemented in Bitcoin assuming t honest wardens (Protocol 11) using chained transactions. In contrast, we conjecture that the incentive-compatible version of Brick (Protocol 12) cannot be deployed without timelocks in platforms with limited contracts like Bitcoin.

Part V

Payment Channel Networks

Chapter 6

Algorithmic Design of PCNs

6.1 Introduction

6.1.1 Motivation

While the efficiency of payment channel networks is undisputed, payment networks have a reputation to be capital hungry and as such difficult to deploy. Furthermore, complex routing algorithms are needed to discover routes with enough capital capacity from sender to receiver. To address these issues in practice, multiple proposals emerged that suggest the use of a central operator [68–70, 80].

In this chapter, we want to better understand this demand for capital, studying the issue from an algorithmic perspective. We want to know the complexity an operator of a payment network, a Payment Service Provider (PSP), will face when setting up a payment network.

In our setting, we assume that the PSP can open channels between two nodes[1] and therefore knows how much capital is on each edge and how it is distributed between the nodes. The nodes are the PSP's customers. The PSP, as the creator of the payment channel network (PCN), bares the costs of opening channels. Further, we assume the PSP acts as a creditor, and hence locks the necessary capital in the network and then periodically gets paid by the customers (either in crypto or fiat money). The PSP's objective is to maximise its profit under capital constraints.

6.1.2 From Payment Channels to Network Design

Algorithmically speaking, a PCN is a graph, where each undirected edge (u, v) is a payment channel between the parties u, v. When a channel (an edge) is

[1] Technically this can be achieved using multi-party channels [60], where the two parties and the PSP join a three-party channel funded only by the PSP.

established, PSP capital is locked into the channel on each side of the edge. This capital can then be moved on the channel, from u to v or vice versa. Transactions can also be multi-hop, moving capital on each edge of the path, in the direction of the path of the transaction. The only constraint is that the capital on any side of any edge must be non-negative at all times.

The PSP needs to decide how to design the PCN, i.e., which edges (channels) the PSP should establish. Moreover, the PSP needs to decide how much capital to assign to these newly established edges, in particular, how much capital on each side of every edge.

Establishing a new channel not only involves capital (which is going to be reclaimed eventually), but will also cost (since each newly established channel needs to be registered with the blockchain). We model this channel opening cost as a constant, given that the fee the blockchain asks is (more or less) constant. The total cost is then the number of open channels (the edges of the network) times this constant cost to open each channel.

Our goal is to define a strategy for the PSP regarding which transactions to execute in order to maximize profit (fees from transactions minus costs to set up channels) and minimize capital (cryptomoney that is temporarily locked into channels). Note that there is a trade-off between profit and capital, as more capital may allow to accept more transactions, earning fees for each transaction, hence increasing profit.

In this work, we study two main variations of this problem. First, we examine the case where the PSP assigns a fee on every channel, much like tolls in a road network (Section 6.2). Note that a PSP competes with the blockchain: customers only prefer the alternative network if the total fees in the payment path cost less than the blockchain fee. The PSP's goal is to decide the graph structure and the fee assignments in order to maximize its profit under capital constraints.

Second, we investigate the case where the PSP requests a fixed fee for each transaction executed in the PCN (Sections 6.3,6.4,6.5,6.6). The PSP wants to maximize its profit, hence the fee is marginally lower than the blockchain fee. In this case, we discuss the following questions: What is the minimum capital needed to be able to accept a given set of transactions? What is the maximum profit the PSP can achieve with a given capital? These questions are at the heart of understanding the Pareto-nature of the trade-off between profit and capital in payment networks.

6.1.3 Contribution

We introduce the first algorithmic framework for the payment channel network (PCN) design problem. We study two variations of the problem: In model (a), we assume that a PSP imposes fees on every channel of its PCN separately; In

model (b), we assume the PSP asks for a fixed fee for each transaction executed on its PCN.

For model (a) we present the following results: First, we provide a linear program formulation for the problem on trees when the PSP wants to facilitate all transactions, proving that this problem variation is in the complexity class P. Then, we show that the optimal fee assignment for any path has only 0/1 values on the fees, assuming 1 is the cost of publishing a transaction on the blockchain, and we present an efficient dynamic programming algorithm to compute the optimal fees. In addition, we prove that the star network is a near-optimal solution of the general network design problem, when we allow an additional node to be added as a payment hub and assume the optimal network is connected. This implies that a PSP can achieve almost maximum profit by creating a payment hub, the construction of which has already been studied in [69, 70].

For model (b) we provide the following results: First, we study the offline problem, i.e., we are given the future sequence of transactions. We show that maximizing the profit given the capital assignments is NP-hard, even for a single channel. Then, we present a fully polynomial time approximation scheme for the single channel case. Later, we consider the case where the PSP wants to maximize its profit and thus execute all profitable transactions. We prove that a hub (a star graph) is a 2-approximation with respect to the capital. Moreover, we show the problem is NP-complete under graph restrictions.

In addition, we examine online variants of model (b). First, we examine a single channel when transactions come online assuming the PSP wants to maximize its profit under capital constraints. We show there is no randomized online algorithm against any adaptive online adversary. Then, we consider algorithms against oblivious adversaries; we show that there is neither a competitive deterministic nor a competitive randomized algorithm. We derive our results for the randomized case from analysing deterministic algorithms with advice. Next, we consider resource augmentation [120], an approach that relaxes the capital constraints to achieve better competitiveness. We prove there is no competitive deterministic algorithm, even with double the capital. Furthermore, we approach the problem as a minimization problem, where we want to minimize the number of rejected transactions. Similarly to the maximization problem, we prove there is no competitive randomized algorithm against oblivious adversaries. Last, we study the general channel design problem in the online setting assuming all profitable transactions are executed through the payment network. We show that the star graph yields an $O(\log C)$-competitive algorithm, where C denotes the optimal capital.

6.2 PCN Design with Fees

6.2.1 Model with Diverse Fees

In this section, we define the *Payment Channel Network Design with Fees (PCN-DF)* problem. We assume the PSP can renew the channels and change the network structure in specific epochs to avoid timing attacks; hence, we only consider a limited set of transactions corresponding to an epoch. Now, given a set of transactions between a fixed number of participants, we wish to create a payment network and assign fees to its channels to maximize the profit for the PSP. To formally define the problem, we introduce the following notation.

We define a channel network as a graph $G = (V, E)$ with a set of vertices V and a set of edges E. Each node $v \in V$ denotes one of n participants wishing to use our network, hence $|V| = n$. An edge $e \in E$ between two nodes i and j represents an open channel C_{ij}, with $|E| = m$. Thus, the set of edges E represents the open channels of our network. For simplicity, we assume that all edges of the graph are undirected, as we assume the capacity of every channel to be infinite, in other words, the PSP has deep pockets and is able to fund channels with a significant amount of capital. Further, we define the cost of each edge in the network to be 1. This represents the cost of opening a channel by submitting a funding transaction to the blockchain as described in [11, 12].

Given a sequence of transactions for n participants, we define a transaction matrix $T \in \mathbb{N}^{n \times n}$. An entry $T[i, j]$ denotes the number of transactions from i to j and back. Note that matrix T is symmetric since the transactions' direction do not matter. If there are no transactions for a pair (i, j) of nodes, then the corresponding matrix entry is 0. Transactions where sender and receiver are identical are meaningless, thus the diagonal entries of the matrix are 0.

For each edge $e \in E$ we can assign a fee $f_e \in \mathbb{R}$. We require every fee to be non-negative. Moreover, we require the fees to be at most 1, which is the cost of any transaction on the blockchain. Allowing the fee on an edge to be more than 1 is equivalent to deleting this edge from the network, since the customers will always prefer to use the blockchain where the transaction fee is 1. We denote by f_E a fee assignment for the set of edges E.

To measure the value of a network we introduce a profit function. The profit of a payment network depends on the structure of the underlying graph, the fee assignments, and the transactions carried out between participants in the network. Given a transaction matrix T, we define the profit of a graph $G = (V, E)$ as

follows:

$$p(G, T, f_E) = -m + \sum_{i,j \in V} \sum_{e \in path(i,j)} f_e \cdot X_{ij} \cdot T[i, j],$$

$$\text{where } X_{ij} = \begin{cases} 1, & \text{if the participant chooses to use the network,} \\ & \text{i.e. } \sum_{e \in path(i,j)} f_e \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

where $path(i, j)$ denotes the set of edges of the shortest path (cheapest sum of fees on edges) from sender i to receiver j in the graph G.

We include a pair of nodes (i, j) in the profit calculation only if this sum of fees on the shortest path is at most 1. Finally, we subtract from the profit the number of edges m, since each transaction that opens a channel costs 1 in the blockchain.

Now, we formally define the *PCN design with fees* or PCN-DF problem.

Problem 6.1. (PCN-DF) Given a transaction matrix $T \in \mathbb{N}^{n \times n}$, return a graph $G = (V, E)$ with $|V| = n$, and fee assignments on edges f_E, such that the profit function $p(G, T, f_E)$ is maximized.

In the following two sections we study a relaxed version of the problem, where the network structure is given. Our goal is to calculate the optimal fee assignments. Specifically, in Section 6.2.2, we examine trees; trees are very natural as they connect a set of nodes with a minimal number of edges, and opening each edge costs a blockchain transaction. In addition we want all customers to prefer the PSP network, thus all paths in a given tree must cost less than 1.

6.2.2 A Linear Program for Trees

In this section, we find a solution to PCN-DF restricted to trees. We assume that every transaction makes sense in the tree, i.e., the sum of the fees on the path of every transaction is at most 1. Therefore, by the model stated above, every user of the payment network will always use the network, and no transaction goes directly on the blockchain. It turns out this problem can be solved efficiently, as stated by the following result.

Theorem 6.2. *Given any tree and any transaction matrix, there exists a polynomial time algorithm to optimize the profit if every transaction can connect using the payment network.*

Proof. To solve this variation of the problem, we can use linear programming to find the optimal profit along with an optimal assignment of fees. In order to do

so for some given tree $G = (V, E)$ and a given transaction matrix T, we need to first determine the objective function that we want to maximize. Moreover, we need to specify suitable inequality constraints.

We compute the objective function by analyzing how many times each transaction uses each edge in the network. This gives us an objective function

$$f(x) = \sum_{i=0}^{m-1} c_i \cdot x_i.$$

The argument of the objective function, a vector $x = (x_0, \cdots, x_{m-1})$ with $m = |E|$ components represents the fees of the edges that we wish to maximize, and c_i denotes the number of times the edge i is used by transactions. Then, to determine the inequality constraints which are imposed by the constraint that each transaction must have a total fee of at most 1, we define one inequality for every transaction t:

$$\sum_{i=0}^{m-1} e_i \cdot x_i \leq 1,$$

where $e_i = 1$ if edge i was used for transaction t, and 0 otherwise. Solving this linear program (in polynomial time) finds the optimal vector x of fees. □

In the following section, we remove the additional assumption that all the transactions should be facilitated by the PSP's network. The problem, now, is more complicated since the selection of transactions cannot be expressed as a linear program (but only as an ILP). Thus, we study the problem in more restricted graph structures: paths.

6.2.3 Dynamic Program for Paths

In this section we present Algorithm 15, a polynomial-time dynamic program that achieves optimal profit in chain networks. We prove that the optimal solution has only fees that are either 0 or 1. First we compute tensor M, where $M[i, j, k]$ is the profit from all transactions in the interval $[i, k]$ when the fee of edge j is 1 and every other fee is 0. Then, we compute matrix P, the maximum entry of which is the optimal profit. $P[lastX, x - 1]$ denotes the maximum profit when setting the fee of the edge with index $lastX$ to 1 (it is possible that more edges have a fee of 1 before that, but $lastX$ is the last edge where this is the case) and only using the edges up to $x - 1$. $M[lastX + 1, x, y]$ denotes the profit from edges in the interval $[lastX + 1, y]$ while the x-th edge's fee is 1. For some fixed x, y we iterate over all possible profits of the preceding part of the graph, add it to the profit of the corresponding current interval and only consider the maximal profit (if larger that setting the x-th edge's fee to 1).

To retrieve a fee assignment that has optimal profit, we can do the following: We define an additional matrix E, where the entries of each row are initialized

Algorithm 15: Dynamic Program for Paths

Data: Number of nodes n.
Result: Fee assignment.

/* Initialization */
m = number of edges, i.e., $n-1$
Set all entries of $M[m,m,m]$ to 0
Set all entries of $P[m,m]$ to 0

/* Compute tensor M */
for *every* $1 \leq i \leq j \leq k \leq m$**:**
    $p = 0$
    for *every entry T[u,v] in* T**:**
        if $u \leq j < v$**:**
            $p = p + T[u,v]$
    $M[i,j,k] = p$

/* Compute the dynamic programming table */
for *every* $1 \leq x \leq y \leq m$**:**
    $P[x,y] = M[1,x,y]$
    for $lastX = 1$ **to** $x-1$**:**
        if *P[lastX,x-1] + M[lastX+1,x,y] >P[x,y]*:
            $P[x,y] = P[lastX, x-1] + M[lastX+1, x, y]$
            Store edges with a fee of 1, i.e., x and edges that have a fee of 1 for $P[lastX, x-1]$
profit = maximum entry in P
fee assignment = edges with a fee of 1 stored for the maximum table entry

with the number of the row. Now, every time we update an entry $P[x,y]$, we set $E[x,y] = [x, E[lastX, x-1]]$. These denote the edges that are assigned a fee of 1 to attain the calculated profit. When the algorithm has ended, we read the entry $E[x',y']$, where x', y' are the indices of the maximum value in P, and set the fee of the edge contained in $E[x',y']$ to 1 in the optimal fee assignment.

Correctness and Runtime. We prove the correctness of Algorithm 15 and analyze its time complexity. In Algorithm 15, an edge is either assigned a fee of 1 or 0. The following lemma states that these are indeed the only two values we need to consider.

Lemma 6.3. *For every given path and for every set of transactions, the optimal profit can always be achieved by assigning edges a fee* 0 *or* 1.

Proof. Assume that we are given some optimal fee assignment $f = (f_1, f_2, \ldots, f_m)$ on the path of length m, and but this assignment may use other values, not only 0 or 1. We show that only using 0 and 1 one also can reach the same (or even more) profit.

Based on the given fee assignment f, we compute the set S of all maximal intervals (i.e., there does not exist a pair of intervals (i, j) and (i', j') such that $i \leq i'$ and $j \geq j'$) where the sum of the fees on the edges in that interval is less or equal to 1. That is, an interval (i, j) is in S if and only if it satisfies that $\sum_{k=i}^{j} f_k \leq 1$ and $\sum_{k=i-1}^{j} f_k > 1$ (or $i = 1$) and $\sum_{k=i}^{j+1} f_k > 1$ (or $j = m$). The optimal profit can be obtained by solving a linear program. It is well known that every linear program reaches its optimal at the vertex of the feasible region. Hence, we only need to show that every entry of every vertex of the feasible region defined above is either 0 or 1. Equivalently, we show that every feasible solution is a convex combination of vectors with only 0 and 1.

We prove this by induction on the length of the path. For the base case, when the length is 1, i.e., a single edge, it is trivial. Now assume that this result holds for paths of length smaller than m, and we prove that it also holds for length equals to m. The key observation is that, for any path, there always exists an assignment f' with only 0 and 1 such that $\sum_{k=i}^{j} f'_k = 1$ for every $(i, j) \in S$, as follows:

1. Let $f'_k = 0$ for all k.
2. For k from 1 to m, consider all intervals (i, j) in S such that $i \leq k \leq j$. If all such intervals (i, j) satisfy $\sum_{t=i}^{j} f'_t = 0$, then let $f'_k = 1$.

We define $K := \{k : f'_k = 1\}$ and let $\theta := \min\{f_k : k \in K\}$. Now we write f as a convex combination $f = \theta \cdot f' + (1 - \theta) \cdot f''$. Since $\sum_{k=i}^{j} f'_k = 1$ for every $(i, j) \in S$, it follows that $\sum_{k=i}^{j} f''_k \leq 1$ for every $(i, j) \in S$. By the definition of θ, we know that there exists at least one index t such that $f''_t = 0$ ($f_t = \theta$). According to these two facts, f'' can be considered as a feasible solution for the path of length $n - 1$, which by the induction hypothesis is also a convex combination of vectors with only 0 and 1. The lemma is proved. □

The above lemma is useful in pruning search space, but it is still exponential (2^m) if we do a brute force search. Our dynamic programming method makes the search space polynomial in the number of edges m, which is shown in the following theorem.

Theorem 6.4. *Algorithm 15 returns the optimal solution and the time complexity of the algorithm is $\mathcal{O}(n^5)$.*

Proof. Let $OPT(x, y)$ denote the profit of the sub-path from edge 1 up to and including edge y where we set the fee of edge x ($x \leq y$) to 1. We claim that $OPT(x, y)$ fulfills the following recurrence:

$$OPT(x,y) = \max \begin{cases} M[1,x,y] & \textit{(Case 1)} \\ P[lastX, x-1] + M[lastX+1, x, y] & \textit{(Case 2)} \end{cases}$$

If $OPT(x,y)$ is equal to *(Case 1)*, this means that we reach the maximum profit in the subgraph from edge 1 to y by only setting the fee of edge x to 1 in the entire subgraph. Consequently, every transaction, that only uses edges from this subgraph, can generate profit.

Otherwise, if $OPT(x,y)$ happens to be *(Case 2)*, we know that there are at least two edges with a fee of 1 in the subgraph from edge 1 to y, namely on edge x and on edge $lastX$. Therefore, profit is generated by transactions in the first part of the subgraph, i.e. from edge 1 to $x-1$, and at the same time in the second part, that is from $lastX+1$ to y. However, no transactions, which use edges in both parts of the subgraph, can generate profit, as such a transaction would then cross both edges with a fee of 1.

Because of this, we can iterate over every possible sum of the profits of $P[lastX, x-1]$ and $M[lastX+1, x, y]$ and choose the maximum thereof. Note, that we do not necessarily choose the maximum for both terms, but instead pick the maximal sum or otherwise we might only obtain a locally optimal solution. This method can be used, since we were able to split the subgraph from 1 to y in two parts as explained above. Moreover, we have already precomputed both terms: $M[lastX+1, x, y]$ was computed at the very beginning of the algorithm and $P[lastX, x-1]$ is always an entry of the table that was the result of a prior computation with exactly the same recurrence.

The tensor M can be computed in time $\mathcal{O}(n^5)$. The computation of the table P can be accomplished in time $\mathcal{O}(n^3)$, since we have 3 loops that iterate over parts of the edge indices. Therefore, the complete algorithm can be implemented with runtime $\mathcal{O}(n^5)$. □

6.2.4 Payment Hub: a Near-Optimal Solution

In this section, we present a near-optimal solution to the PCN-DF problem. Please note that *the optimal solution is not always a tree*. For example, if we consider three nodes with many transactions between every pair, the optimal payment network is the triangle with a fee of 1 on each edge. A tree will connect the three nodes with a two-edge path, hence none of the trees achieve maximum profit.

We show that if the optimal network is connected, then the *star graph*, where the center is an additional node acting as a payment hub, is a *near optimal solution*. We believe the network connectivity assumption is reasonable: In a monetary system we expect some nodes to be highly connected, representing big companies that transact with many nodes on the network. These highly connected nodes assist in connecting the entire network in one big connected component.

We denote $opt(T)$ the profit, G_{opt} the graph and $f_{E_{opt}}$ the fee assignment of the optimal solution for a given transaction matrix T. Moreover, we denote $S = (V_S, E_S)$ the star graph that includes all nodes V and an additional one, c, as the center of the star. We assign uniform fees to all the edges, $f_e = 0.5, \forall e \in E_S$.

Theorem 6.5. *If G_{opt} is connected, then $p(S, T, f_{E_S}) \geq opt(G_{opt}, T, f_{E_{opt}}) - 1$.*

Proof. If G_{opt} is one connected component, then $|E_{opt}| \geq n - 1$. For the star graph S, we have $V_S = V + c$ and $|E_S| = n \leq E_{opt} + 1$. Furthermore, the sum of fees on all shortest path is equal to 1 due to the uniform fees equal to 0.5 and the star structure. The profit function maximizes its value when all transactions go through the graph G_{opt} with total fee equal to 1, hence

$$opt(G_{opt}, T, f_{E_{opt}}) \leq \sum_{i,j \in V} T[i,j] - |E_{opt}| \leq \sum_{i,j \in V} T[i,j] - |E_S| + 1 = p(S, T, f_{E_S}) + 1$$

The last equality holds since the sum on every shortest path equals to 1. □

6.3 Model with Fixed Fees

6.3.1 Graph Model

We define a payment network as an undirected graph $G = (V, E)$ with a set of edges E and a set of nodes V. A node $v \in V$ is a participant in the network whereas an edge $e = (u, v) \in E$ is a channel between two nodes $u, v \in V$. For each channel $(u, v) \in E$, we denote by C_l and C_r the capital available to node u and node v, respectively. Therefore, every time a transaction is executed through the channel (u, v), the capitals C_l and C_r are updated according to the distribution of the capital in the latest update transaction of the channel. Note, that the total capital of a channel does not change, i.e., the sum of the capital C_l and C_r remains the same. Further, the capital moves on the channel like the balls in a row of an abacus: if u wants to send capital c to v on channel (u, v) then the new distribution of the capital on the edge (u, v), after the off-chain execution of the transaction, will be $C_l - c$ and $C_r + c$, respectively.

We denote by $t = (s, r, v)$ a transaction t that moves capital of value v from sender $s \in V$ to receiver $r \in V$. Furthermore, in the simple case where we examine only a single channel, i.e., the graph has a single edge, we denote by $\langle C_l; C_r \rangle$ the capital distribution on the single edge.

6.3.2 Problem Variants

Let us now formally define the problems we will study.

Problem 6.6 (General Payment Network Design)**.**

Input: Capital C, profit P, the sequence of n transactions $t_i = (s_i, r_i, v_i)$ with $1 \leq i \leq n$, each containing the sender node s_i, the receiver node r_i, and the value v_i of the transaction t_i.

Output: Strategy $S = \{0,1\}^n$, a binary vector where the i^{th} position is 1 if we choose to execute the i^{th} transaction of the input and 0 else. The graph $G(V, E, C_l, C_r)$ is the network we created to execute the chosen transactions, where V is the set of senders and receivers that participate in any transaction, E is the set of channels we open, and C_l, C_r the capital on each side of each edge. Each transaction can be routed arbitrarily in G, denoted by $S_e = \{-1, 0, 1\}^n, for\ all\ e \in E$, i.e., $S_e(i) = 1$ (or -1) if transaction i is routed through edge e from left to right (from right to left, respectively), and $S_e(i) = 0$ if transaction i is not routed through edge e.

Our goal is to return (if it exists) a strategy S, a graph G and a routing S_e subject to the following constraints:

1. $|S| - |E| \geq P$
2. $\forall e \in E, \forall j \in \{1, 2, \ldots n\},\ -C_r(e) \leq \sum_{i=1}^{j} S_e(i) \cdot v_i \leq C_l(e)$
3. $\sum_{\forall e \in E} C_l(e) + C_r(e) + |E| \leq C$

The first inequality guarantees that the fees of the accepted transactions minus the cost of opening the channels is at least as high as the intended profit. The second inequality makes sure that at any time the capital on each side of each channel is non-negative. The third inequality ensures that the used capital on the channels and the cost of opening the channels is at most the available capital.

Problem 6.6 in all its generality is difficult, as it features many variables. Consequentially, we mostly focus on the most interesting special cases of Problem 6.6: We consider transactions on a single channel between just two nodes. And we consider minimizing the capital assuming all profitable transactions are executed. Formally the problems we examine are the following.

Problem 6.7 (Single Channel)**.** Given a sequence of n transactions $t_i = (s, r, v_i)$, where s and r are the nodes of the single edge e, a capital assignment $C_r(e), C_l(e)$, and a profit P, decide whether there is a strategy S such that $|S| \geq P$ and $\forall j \in [n],\ -C_l(e) \leq \sum_{i=1}^{j} S(i) \cdot v_i \leq C_r(e)$.

Problem 6.8 (Channel Design for All Transactions)**.** Given a sequence of n transactions $t_i = (s_i, r_i, v_i)$, return the graph $G(V, E)$ that achieves maximum profit with minimum capital C.

Problem 6.9 (Capital Assignment and Routing)**.** Given a graph $G(V, E)$, a sequence of n transactions $t_i = (s_i, r_i, v_i)$ and a capital C, determine whether all transactions can be executed in G with the given capital C.

In this chapter, we examine these problems both in the offline and the online setting. In the offline setting, we assume that all future transactions are known a priory, while in the online case the transactions arrive in a stream and we must instantly decide whether to execute them or not. In the online case, our results depend on the adversarial model, i.e., how optimal is the input transaction stream for a given algorithm.

6.3.3 Adversarial Model

In literature [121], there are three types of adversaries concerning online problems: the oblivious adversary, the adaptive online adversary and the adaptive offline adversary. These adversarial models are equally powerful regarding deterministic algorithms but may yield drastically different results when considering randomized algorithms. We define the adversarial models below with respect to our setting.

Definition 6.10 (Oblivious adversary)**.** *An oblivious adversary provides a sequence of transactions before an online algorithm starts its computations.*

Definition 6.11 (Adaptive online adversary)**.** *The adaptive online adversary provides the next transaction based on the decision an online algorithm makes (accept or reject the previous transaction) but serves it immediately.*

Definition 6.12 (Adaptive offline adversary)**.** *The adaptive offline adversary provides the next transaction based on the decision an online algorithm makes (accept or reject the previous transaction) and serves the output at the end such that it acts optimally.*

An adaptive offline adversary (Definition 6.12) knows the randomness of the online algorithm, in contrast to an adaptive online adversary (Definition 6.11).

6.3.4 Assumptions

We assume the fee of a transaction on the blockchain to be constant, without loss of generality equal to 1 coin. The fee of a transaction in the payment network cannot be higher than the fee on the blockchain, or a potential user may prefer the blockchain over the payment network. A rational PSP will ask for a transaction processing fee which is as high as possible but lower than the blockchain fee, hence for $1 - \epsilon$ coins. In our analysis we will usually assume that $\epsilon \to 0$.

6.4 Offline PCN Design

In this section, we study the offline channels network design problem, i.e., we assume we know the future transactions (for the next period). First, we explore

the network topology for the general problem. Then, we examine the case where we are given a specific capital (or even a capital assignment) and we aim to maximize the PSP's profit, hence execute as many transactions as possible. We focus on solving the problem for a single edge of the network, since even in this simple case the problem is challenging. Later, we focus on minimizing the capital given the PSP wants to execute all the profitable transactions.

6.4.1 Graph Topology

We first prove some observations concerning the optimal graph structure. We consider as optimal the solution that maximizes the profit while respecting the capital constrains (optimization version of Problem 6.6).

Lemma 6.13. *The graph of the optimal solution does not contain any node that sends and receives less than two transactions.*

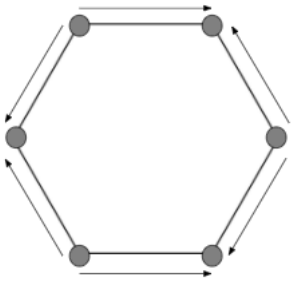
Proof. Assume node u is in the graph and sends and receives less than two transactions. Since u is part of the graph, it has at least one neighbor. Choose an arbitrary neighbor v of u, connect all remaining neighbors of u directly with v (if not already connected), and then remove u. For each neighbor w of node u, increase the capital of edge (w, v) in the new graph by the capital of the removed edge (w, u) (on both sides of the edge). Now all transactions routed originally through edge (w, u) can be routed in the new graph through edge (w, v), the new total capital needed is at most the same as the old capital, and there is enough capital to route all previously routable transactions. We denote by *opt* the profit of the optimal solution. The graph without u has at least one edge less. Thus, the profit of the new graph for (at least) the same set of transactions is at least $opt - (1 - \epsilon) + 1 > opt$, since setting up the channel (edge) (u, v) costs 1 but u's transaction fees are at most $1 - \epsilon$. So we are better off without node u. □

Thus, during preprocessing we can safely remove all transactions that contain a node that is only sender or receiver of a transaction in this one transaction. The time complexity of this procedure is linear in the number of transactions.

Proposition 6.14. *The optimal graph is not necessarily a tree (or forest).*

Proof. Suppose we are given the following sequence of transactions and capital $C = 6a + 6$ where $a > 11$:

$t_i = (v_2, v_1, 1)$, for $1 \leq i \leq a$
$t_i = (v_2, v_3, 1)$, for $a + 1 \leq i \leq 2a$
$t_i = (v_4, v_3, 1)$, for $2a + 1 \leq i \leq 3a$
$t_i = (v_4, v_5, 1)$, for $3a + 1 \leq i \leq 4a$
$t_i = (v_6, v_5, 1)$, for $4a + 1 \leq i \leq 5a$
$t_i = (v_6, v_1, 1)$, for $5a + 1 \leq i \leq 6a$



For this example, we will show that the optimal solution that maximizes the profit given the capital $C = 6a + 6$ returns a graph that contains a cycle.

One solution is the graph $G(V, E)$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_1)\}$. The profit in G is $6a - 6$, since all $6a$ transactions are executed and connect directly and 6 channels are opened. The spent capital is $6a + 6 = C$ since we open 6 edges and lock capital a on the sender side of each edge.

If the graph is not connected we lose at least a transactions and remove at most 6 edges. So, the total profit decreases by at least $a - 6 > 5$, hence the optimal graph is connected.

Suppose now the optimal graph does not contain any cycle. Since the optimal graph is connected, it is a spanning tree. Let's denote by ℓ the number of leaves in the tree with $2 \leq \ell \leq 5$, since the spanning tree has 5 edges. If we want to have at least the profit of the cycle described above, we must deliver at least $6a - 1$ transactions, since we only saved the opening cost of a single edge.

We can see that the capital locked on the edges connecting these leaves is at least $2a\ell - 2$, since, for every node, both transactions involved are either outgoing or incoming. For every other edge, the capital locked on it is at least $a - 1$. Therefore, the total number of locked capital is at least $2a\ell - 2 + (a - 1)(5 - \ell) \geq 7a - 5 > 6a + 6 = C$. □

Due to the complexity of the problem we focus on a single channel. It turns out that even for this degenerate case, the problem is far from trivial.

6.4.2 Single Channel

We now focus on a single channel. We prove that even in this case the problem of choosing the transactions that maximize the profit given capital assignments is NP-hard and present an FPTAS.

Specifically, we are given a sequence of transactions on a single edge of a network and their values, the capital assignment on the edge and a target profit. Our goal is to decide whether we can execute at least as many transactions as the given target profit while respecting the capital constraints. Since the number of

edges is fixed and equal to 1 the profit now is the number of executed transactions (Problem 6.7). The problem is equivalent to a variant of the 0/1 knapsack problem where each transaction represents an item. Each item has profit 1 and either positive or negative size (values). The capacity of the knapsack is represented by the capital assignments and the goal is to maximize the profit while respecting the capacity.

Problem 6.15 (Fixed Weight Subset Sum or FWSS)**.** Given a set of non-negative integers $U = \{a_1, a_2, \ldots, a_n\}$, and non-negative integers A and l, is there a non-empty subset $U' \subseteq U$ such that $|U'| = l$ and $\sum_{a_i \in U'} a_i = A$?

Lemma 6.16. *FWSS is NP-hard.*

Proof. We will reduce Subset Sum (SS) [122] to FWSS.
SS: Given a set of integers $U = \{a_1, a_2, \ldots, a_n\}$, and integer A, is there a non-empty subset $U' \subseteq U$ such that $\sum_{a_i \in U'} a_i = A$?
Given an instance of SS, we define n different instances of FWSS, one for each possible value of l, where the set of integers U and the integer value A are the same for every FWSS instance as in SS. If one of the FWSS instances returns "yes" then we return "yes", else we return "no". If any instance of FWSS returns "yes", then the same subset satisfies the SS problem, thus it must return "yes". If all instances of FWSS return "no", then there is no set satisfying the SS problem, since we checked all possible set sizes. Thus, SS must return "no" as well. The transformation is polynomial to the input. □

Theorem 6.17. *Problem 6.7 is NP-hard.*

Proof. We will reduce Fixed Weight Subset Sum (FWSS) to Problem 6.7.

Assuming we are given an instance of the FWSS, we present a polynomial time transformation to an instance of Problem 6.7. We first define the capital assignment on the edge $C_r(e) = A(l+1)$, $C_l(e) = 0$ and the profit $P = l + n(l+1)$. Then, we define the sequence of transactions as follows: $v_i = a_i + A$, $\forall 1 \leq i \leq n$ and $v_i = -A/n$, $\forall n < i \leq n(l+2)$. We will prove that there is a non-empty set that satisfies the FWSS problem if and only if we can choose transactions that satisfy the capital constraints and profit in the aforementioned instance.

Assume we have a "yes" instance of the problem. Then, we have chosen at least $P = l + n(l+1)$ transactions to execute. We will show that this corresponds to choosing l positive transactions that sum up to $A(l+1)$, thus to a solution of the FWSS problem. Towards contradiction, we examine the following cases:

- If the number of positive transactions is less than l, the total profit is less than $l + n(l+1)$, since there are only $n(l+1)$ negative transactions.
- If the number of positive transactions is more than l, then we violate the capital constraints, since $\sum_i v_i \geq A(l+1) + \sum_i a_i > A(l+1) = C_r(e)$, where i corresponds to the chosen transactions.

- Suppose the l chosen transactions' values sum to more than $A(l+1)$. Then, the capital constraint is violated.
- Suppose the l chosen transactions' values sum to less than $A(l+1)$; suppose the sum is $Al+\sigma$ with some $\sigma < A$. Then, then negative transactions to be executed can be at most $\frac{lA}{A/n} + \frac{\sigma}{A/n} < ln + n$. Thus, the profit is strictly less than $l + ln + n$. Contradiction.

Thus, a "yes" instance of our problem implies a "yes" instance of the FWSS problem. For the other direction, we will prove that if there is no subsequence of transactions of size at least P that satisfies the capital constraints, then there is no subset of size l that sums to A in FWSS. Equivalently, we will show that if there is a subset of size l that sums to A in FWSS, then there exists a subsequence of transactions of size at least P that satisfies the capital constraints. Suppose there is a non-empty set $U' \subseteq U$ such that $|U'| = l$ and $\sum_{a_i \in U'} a_i = A$. Then we can execute the l transactions that correspond to the chosen a_i's with exactly the $C_r(e)$ capital, which will be transfered on $C_l(e) = A(l+1)$. Then, we can execute all the negative transactions since they are $n(l+1)$ many with values A/n, thus we need $A(l+1) = C_l(e)$ capital. Therefore, we can execute $P = l + n(l+1)$ transactions, achieving the required profit while satisfying the capital constraints. □

Both FWSS and Problem 6.7 are also polynomially verifiable, hence NP-complete.
The classic dynamic programming approach that typically yields a polynomial time algorithm when profits are fixed is not efficient since in this variation we cannot optimize using the minimum value at each step due to negative values. Instead, we present a fully polynomial time approximation scheme (FPTAS).

Theorem 6.18. *Algorithm 16 is a fully polynomial time approximation scheme for Problem 6.7.*

Proof. The running time of the algorithm is $O(\frac{n^3}{\epsilon})$, which is polynomial in both n and $\frac{1}{\epsilon}$.
We will prove that the profit of the output of Algorithm 16 is at least $(1-\epsilon)$ times the optimal. We denote by S the set of transactions returned by the algorithm, O the set returning the optimal profit and $prof(X)$ the profit from the set of transactions X. Since we scaled down by K and then rounded down, for every transaction i we have that $Kv'_i \leq v_i$. Therefore, the profit of the optimal set can decrease at most nK, $prof(O) - prof'(O)K \leq nK$. The dynamic program returns the optimal set for the scaled instance. Thus, $prof(S) \geq prof'(O)K \geq prof(O) - nK = prof(O) - \epsilon V \geq (1-\epsilon)prof(O)$, since $prof(O) \geq V$. □

Scaling to many channels. Unfortunately, even when the graph is a tree, Algorithm 16 does not scale efficiently. Creating an m-dimensional tensor for the

Algorithm 16: Maximum Profit

Data: Number of transactions n, values of the sequence of transactions $v_i \in \mathbb{R}, \forall 1 \leq i \leq n$, capital C, approximation factor ϵ.

Result: Binary vector $S = \{0,1\}^n$ that indicates which transactions to execute.

Let $K = \frac{\epsilon V}{n}$, where $V = \max_{1 \leq i \leq n} v_i$

For all transactions $1 \leq i \leq n$ define $v'_i = \lfloor \frac{v_i}{K} \rfloor$

Let $T(i,j) = 0$, for all $1 \leq i \leq n$ and $1 \leq j \leq \frac{n^2}{\epsilon}$

for $i = 1$ *to* n**:**

    for $j = 1$ *to* $\frac{n^2}{\epsilon}$**:**

$$T(i,j) = \begin{cases} \max\{T(i-1,j), 1 + T(i-1, j-v'_i)\} & ,if \ \frac{C}{K} \geq j - v'_i > 0 \\ T(i-1,j) & ,else \end{cases}$$

        Store for every $T(i,j)$ a n-binary vector $S_{i,j}$ that has value 1 in the k-th position if the k-th transactions is chosen to be executed

Return vector $S_{i,j}$ for the maximum $T(i,j)$ such that $\sum_{k=1}^{n} S_{i,j}(k) \cdot v_k \leq C$

dynamic program, where m are the edges of the tree, has time complexity $O(C^m n)$ where C is the maximum capital from all edges. Even if we bound the capital by a polynomial on n, the algorithm remains exponential due to the number of edges on the exponent. In the general case where the graph could contain cycles, the problem becomes even more complex. Now, we need to additionally consider all possible routes for each transaction, which adds an exponential factor on the running time of the algorithm.
Since Problem 6.6 is complex, we study special cases that might be useful in practice and provide an insight to the general problem.

6.4.3 Channel Design for Maximum Profit

In this section, our goal is to find the minimum capital for which we can achieve maximum profit, i.e., execute all profitable transactions (Problem 6.8). At first, we note some simple observations for the graph structure. Then, we prove that any star graph is a 2-approximate solution with respect to the capital, but even the "best" star is not an optimal solution. Last, we prove the problem is NP-hard when there are graph restrictions.
Throughout this section, we refer to the optimal solution of Problem 6.8 as the *optimal network for maximum profit.*

Proposition 6.19. *When the capital is unlimited, the optimal network for maximum profit does not contain cycles.*

Proof. Suppose the optimal graph contains a cycle. We choose an arbitrary edge of the cycle, denoted $e = (i, j)$, where i, j the nodes of edge e. We remove edge e and reroute all transactions using edge e through the path from i to j. This path exists since removing an edge from a cycle cannot disconnect the graph. The profit of the graph without e is the profit of the optimal graph plus 1 (the cost of opening edge e). This is a contradiction, since the optimal network maximizes the profit. □

Proposition 6.20. *When the capital is unlimited, there exists an algorithm, with time complexity $\Theta(n)$, where n denotes the number of transactions, that returns the optimal network for maximum profit.*

Proof. We present the following algorithm:

1. Traverse the input transactions and create a list L that contains the number of transactions between every two nodes (potential edges).

2. Traverse list L as follows:
   If the number of transactions is at least 2 and adding the edge between the two nodes does not form a cycle, add the edge to the (initially empty) graph.

The algorithm guarantees there is a path between every two nodes that want to execute at least two transactions (Lemma 6.13). Thus, all transactions that increase the PSP's profit are executed. Moreover, the algorithm does not contain a cycle, so the output graph is minimal regarding the edges. The first argument maximizes $|S|$ and the second minimizes $|E|$ for the given sequence of transactions. Therefore, the algorithms returns a graph that achieves maximum profit ($max\ |S| - |E|$).
The running time of the algorithm is linear to the number of transactions, $\Theta(n)$. □

Lemma 6.21. *The optimal network for maximum profit is not necessarily a connected graph.*

Proof. Suppose the graph of the optimal solution is always connected. Assume now we are given the following sequence of transactions: $t_1 = (v_1, v_2, 1), t_2 = (v_1, v_2, 1), t_3 = (v_3, v_4, 1), t_4 = (v_3, v_4, 1), t_5 = (v_1, v_3, 1)$. Since the optimal graph for this example is connected, there are at least three edges in the graph. Thus, the optimal profit is $|S| - |E| \leq 5(1 - \epsilon) - 3 = 2 - 5\epsilon$. However, if we consider the graph with edges (v_1, v_2) and (v_3, v_4), the PSP's profit is $2 - 4\epsilon$, greater than the profit of the optimal solution, which is a contradiction. □

We refer to transactions that increase the PSP's profit as *profitable transactions*. We assume all nodes participate in at least two transactions (Lemma 6.13).

Proposition 6.22. *Not all transactions are profitable transactions.*

Proof. Suppose all transactions are profitable transactions. Lets assume we are given the following sequence of transactions: $t_1 = (v_1, v_2, 1), t_2 = (v_1, v_2, 1), t_3 = (v_3, v_4, 1), t_4 = (v_3, v_4, 1), t_5 = (v_1, v_3, 1)$. It is straightforward to see that any solution that executes all transactions must return a connected graph. However, we showed in the proof of Lemma 6.21 that the optimal graph for this example is not connected. Thus, there are transactions that are not executed in the optimal solutions and hence they are not profitable transactions. □

Despite Lemma 6.21, we note that payment channels are monetary systems. As such, large companies are expected to participate in the network as highly connected nodes, ensuring that the optimal graph is one connected component. Thus, for the rest of the section we can safely assume that the optimal graph is connected.

We will now define some formal notation to prove that choosing any star as the graph to route all transactions requires at most twice the capital of the optimal graph. This immediately implies we have a 2-approximation to Problem 6.8.

Now, suppose we can update the capital of an edge before executing each transaction. This way we can guarantee there is enough capital on all channels for each transaction execution. These updates are for free, like assigning tokens, and we use them as a stepping stone to calculate the total capital (amortized analysis). Let us denote $c_G(uv, i)$ the additional capital required at the edge (u, v), for transaction t_i with direction from u to v on graph G. Now, we have that the total capital on graph G, denoted by C_G, is

$$C_G = \sum_{\forall (u,v)} \sum_{\forall i} c_G(uv, t_i)$$

Further, let *opt* denote the optimal graph and V the set of nodes contained in *opt*.

We will show that the capital used to route a sequence of transactions on any star that contains the same set of nodes as the optimal graph is at most twice the capital used by the optimal solution for the same sequence.

Theorem 6.23. *Any star graph yields a 2-approximate solution for Problem 6.8.*

Proof. To prove the theorem, we just need to prove that for any sequence of transactions $t_1, t_2, \ldots, t_n$, for any star graph $S(V)$, $C_S \leq 2C_{opt}$. We will show that we can execute on the star graph the same sequence of transaction as the optimal solution with twice as many tokens (amortized capital). Initially we have zero tokens on all edges on both the optimal and the star graph. Every time a

new transaction t_i comes the optimal solution finds a path from sender to receiver. For every edge (u, v) on this path the optimal solution assigns $c_{opt}(uv, t)$ tokens. Then, we assign on the star, S, $c_{opt}(uv, t)$ tokens on the edges mu and vm, where m is the central node on S. The only exceptions are the sender and receiver nodes, s and r respectively, where the tokens are initially placed on sm and mr to execute the transaction. Thus, for every transaction the sum of the tokens used on the star graph are twice the sum of the tokens used on the optimal solution. Therefore, the overall required capital on the star is at most twice the optimal capital, $C_S \leq 2C_{opt}$.
To complete our proof, we need to show we assigned in total enough tokens to execute the given sequence of transactions. When a new transaction comes from s to t, we only need to guarantee there enough tokens on sm and mt. Obviously, if a transaction needs additional tokens to be executed on the optimal graph then the aforementioned strategy guarantees the additional tokens for the star graph as well. If there are already some tokens on the optimal graph for the sender s then either s was previously an intermediate node or a receiver node. In both those cases the same amount of tokens would have been stored on sm as well. With a similar argument, if there were some tokens for the last edge to reach the receiver r on the optimal graph then r was either an intermediate node or a sender. Again, in both those cases the same amount of tokens would have been assigned to mr on the star. □

Proposition 6.24. *The star graph is not an optimal solution for Problem 6.8.*

Proof. We present a sequence of transactions for which any star graph requires larger capital to execute all transactions than the optimal solution, as illustrated in Figure 6.1. Moreover, finding the optimal solution is not trivial in our example, even though we only consider unitary transactions.
The sequence of transactions is the following

$$t_1 = (v_4, v_2, 1), t_2 = (v_4, v_7, 1), t_3 = (v_2, v_6, 1), t_4 = (v_5, v_2, 1), t_5 = (v_6, v_5, 1),$$

$$t_6 = (v_4, v_5, 1), t_7 = (v_3, v_2, 1), t_8 = (v_2, v_1, 1), t_9 = (v_1, v_3, 1), t_{10} = (v_3, v_4, 1)$$

Figure 6.1 illustrates the optimal graph (not necessarily unique). The capital needed to execute all transactions is 6, which is the least possible since any tree with seven nodes has six edges and all of them are used at least one time with unitary values (in this example). There are seven different stars, one for each node as a center. It is easy to see that the capital needed for each one of them is at least 7, which is strictly larger than the optimal. □

The centralized nature of the star is quite convenient for a payment network operated by a PSP. The *star alleviates the problem of participation incentives* detected on decentralized payment networks; now the participants of the network can be online only when they want to execute a transaction. Although the star

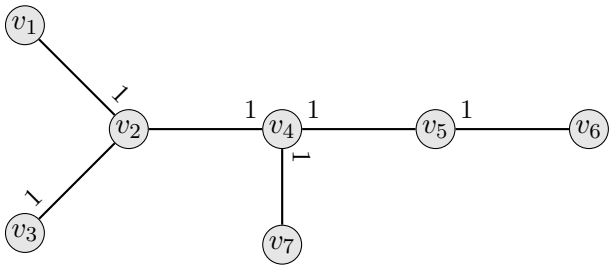


Figure 6.1: The optimal graph and capital assignment to execute all transactions in the given sequence. The capital locked on each channel is illustrated by the number (1) above each edge and the position indicates the direction (in the initial state).

graph is not optimal, it is a good enough solution for a PSP, since the capital the PSP needs to lock in the channels is at most twice the minimum. Thus, payment hubs are an economically viable solution for the throughput problem on cryptocurrencies.

6.4.4 Channel Design with Graph Restrictions

An interesting variation of the problem is when the network has restrictions (Problem 6.9). Instead of allowing all possible channels, we assume some of them cannot occur in real life. In this case, we are given a graph with all the potential channels, the sequence of transactions and the capital, and we want to find the induced subgraph that maximizes the profit. We prove that the problem of deciding whether all given transactions can be executed in the given graph with a fixed capital is NP-complete.

The graph is given so the capital needed to open the channels is fixed in each given instance. Thus, we assume the capital corresponds solely to the capital we lock on the edges but not the one we require to open the channels.

Theorem 6.25. *Problem 6.9 is NP-complete.*

Proof. We will reduce Partition, a known NP-complete problem [122], to Problem 6.9.
Partition: Given a finite set $A = \{a_1, a_2, \ldots, a_m\}$ and a size $s(a_i) \in \mathbb{Z}^+$ for each $a_i \in A$, is there a subset $A' \subseteq A$ such that $\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A - A'} s(a_i)$?
Given an instance of Partition we define an instance of Problem 6.9 as follows: Let $v = \sum_{a_i \in A} s(a_i)$.We consider the graph $G(V, E)$ with four nodes $V = \{b, c, d, e\}$ and edges $E = \{eb, bd, dc, ec\}$. We define the capital $C = 2v$ and the sequence of $n = m + 4$ transactions

$$\{(d, b, v/2), (b, e, v/2), (d, c, v/2), (c, e, v/2), (e, d, s(a_1)), (e, d, s(a_2)), \ldots, (e, d, s(a_n))\}$$

We will prove that this instance of Partition is a "yes" instance if and only if all transactions can be executed in G with the given capital C. We denote by T the last m transactions defined above. Suppose all the transactions in the instance we defined can be executed using the capital C. This implies we can divide T into two groups $T', T - T'$ that each sum to at most $C/2$ and route each group through one of the two paths from e to d in G. Since $\sum_{t_i \in T} v_i = C/2$, we have $|T'| = |T - T'| = C/4$. We define the set $A' = \{a_{i-4} | t_i \in T'\} \subseteq A$. It holds that

$$\sum_{a_i \in A'} s(a_i) = \sum_{t_i \in T'} v_i = C/4 = \sum_{t_i \in T - T'} v_i = \sum_{a_i \in A - A'} s(a_i)$$

For the opposite direction, suppose we cannot execute all transactions in G with capital C. Since the first four transactions can always be executed with capital C, we cannot execute all transactions in T with capital C. Thus, in the optimal solution we cannot partition the transactions in T in two groups $T', T - T'$ with equal sum. Since we defined the values of transactions in T to be the sizes of the elements in A, we conclude there is no subset $A' \subseteq A$ such that $\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A - A'} s(a_i)$. □

6.5 Online Single Channel

In this section, we study the online case, assuming no prior knowledge for the future transactions. When there is a transaction request, we instantly decide whether to execute it or not through our network, assuming we have enough capital on the edges of the path we want to route the transaction. If there is not enough capital on some of the edges, we can refund a channel, which costs 1 coin, the same as opening a new channel.

Similarly to the offline case, we first consider the online version of Problem 6.7. In other words, we study whether there is a profitable strategy for accepting and rejecting transactions (online) given capital constraints.

To this end, we define the competitive ratio of an algorithm, denoted by c, as the value that upper bounds the ratio between the profit of the optimal offline solution, $OPT(I_x)$, and the profit of the online algorithm, $ALG(I_x)$, for any input sequence I_x, i.e.,

$$\forall x,\ c \geq \frac{OPT(I_x)}{ALG(I_x)}$$

The profit, in both cases, represents the number of accepted transactions.

The rest of the section is structured as follows: First, we discuss the existence of competitive algorithms against adaptive adversaries. Later, we mainly focus on algorithms, deterministic or randomized, against oblivious adversaries. In particular, we show lower bounds on the advice bits for competitive deterministic algorithms with advice. Then, we use the relationship between advice and

randomization to discuss the competitiveness of randomized online algorithms. Last, we consider resource augmentation, i.e. we assume an online algorithm has more resources (more capital on both sides) than an optimal offline algorithm.

6.5.1 Algorithms against Adaptive Adversaries

Lemma 6.26. *There is no competitive deterministic algorithm against adaptive online adversaries.*

Proof. Suppose we have a channel with $C_r = C_l = 5$. Transactions from left to right have positive values, those from right to left have negative values. Let us consider two different transaction sequences:

1. $(1, 5, -10, 10, -10, 10, \ldots)$
2. $(1, 4, -10, 10, -10, 10, \ldots)$

Apart from the second transaction, both sequences are identical: The first transaction has value 1, starting with the third transaction we always move the complete capital with every transaction. The only difference is the second transaction.

If some online algorithm accepts the first transaction, then the adversary presents the first sequence; if the online algorithm denies the first transaction, then the adversary reveals the second sequence. Therefore, no matter whether this online algorithm accepts the first transaction or not, it can at most accept one transaction, while the optimal offline algorithm can accept almost all transactions (in case of the first sequence, the offline algorithm only needs to deny the first transaction, in case of the second sequence it will accept all transactions). □

Theorem 6.27. *There is no competitive randomized algorithm against adaptive online adversaries.*

Proof. Assume that there is a competitive randomized algorithm against any adaptive online adversary. Theorem 2.1 and Theorem 2.2 in [121] state that the existence of a randomized c-competitive algorithm against any adaptive online adversary and the existence of a randomized d-competitive algorithm against any oblivious adversary imply the existence of a cd-competitive randomized algorithm against any adaptive offline adversary. Additionally, this implies that there is a cd-competitive deterministic algorithm. Since a c-competitive algorithm against any adaptive online adversary is c-competitive against any oblivious adversary, the existence of a c-competitive randomized algorithm against any adaptive online adversary implies the existence of a c^2-competitive deterministic algorithm. From Lemma 6.26 we know that there are no deterministic online algorithms against adaptive online adversaries which contradicts our assumption and proves that there is no competitive randomized algorithm. □

From here on, we discuss algorithms against oblivious adversaries since there are no algorithms against stronger adversaries.

6.5.2 Algorithms with Advice

Next, we examine algorithms with advice. We refer to an algorithm as *optimal*, if the competitive ratio is one, hence it performs as well as the optimal offline algorithm. Next, we study optimal deterministic algorithms with advice, i.e. we examine how many advice bits are necessary for an optimal deterministic algorithm. We prove a tight lower bound of $n-2$ advice bits for optimal deterministic algorithms. Later, we study the relation between the necessary number of advice bits and the competitiveness of an algorithm.

Proposition 6.28. *There exists an optimal deterministic algorithm with $n-2$ advice bits.*

Proof. The advice (oracle) gives the strategy for the first $n-2$ transactions. For the last two transactions, the algorithm proceeds greedily. If the optimal offline algorithm accepts none of the last two transactions, none of them can be accepted. If one or two transactions can be accepted, our algorithm will accept and reject them accordingly such that it is optimal. □

Proposition 6.29. *There is no optimal deterministic algorithm with less than $n-2$ advice bits.*

Proof. Assume an algorithm ALG reading less than $n-2$ advice bits exists. Then, ALG reads no advice bits for $n=3$. We construct following sequences of transactions with initial capital distribution $\langle 2;1\rangle$:

1. $(l,r,2)$,$(l,r,1)$,$(l,r,1)$
2. $(l,r,2)$,$(r,l,3)$,$(r,l,3)$

For the second sequence of transactions, the optimal offline algorithm accepts the first transaction, whereas, for the first sequence, the first transaction is rejected. Since ALG is deterministic, the decision made on the first transaction will always be the same. Consequently, ALG cannot be optimal. □

We notice that the number of advice bits necessary for optimal algorithms is very high. Therefore, we consider, next, algorithms with competitive ratio greater than one, and show a relation between the competitive ratio and the required advice bits of an algorithm.

Theorem 6.30. *An algorithm with strictly less than $f(n)$ advice bits has a competitive ratio of $\Omega\big(\frac{n}{f(n)}\big)$.*

Proof. Let the initial capital distribution be $\langle 2^{f(n)}; 1\rangle$ and $f(n)$ be a function such that $f(n) \in o(n)$. *loop* denotes $n - f(n) - 1$ transactions that move the total capital $(2^{f(n)} + 1)$ from one side to the other. We define a sequence of transactions as

$$I_x = (l, r, 1), (l, r, 2), (l, r, 4), \ldots, (l, r, 2^{f(n)-1}), (r, l, x + 1), loop$$

such that the set of instances is defined as

$$\mathcal{I} = \{I_x \mid x \in \{0, \ldots, 2^{f(n)} - 1\}\}$$

Now, an algorithm must accept *loop*, otherwise the competitive ratio is $c \geq \frac{n-f(n)}{f(n)+1}$. Assume that there is an algorithm A reading at most $k < f(n)$ advice bits. For instance I_x, A must transfer x capital with the transactions of the prefix to the right side such that it can accept *loop*. The number of strategies the advice can describe is smaller than the number of instances in the constructed input set. Thus, we know that for two different inputs the same strategy is used. We conclude, A cannot accept *loop* for one element in $\mathcal{I}$ and has competitive ratio $c \geq \frac{n-f(n)}{f(n)+1}$. □

6.5.3 Randomized Algorithms against Oblivious Adversaries

We now study the competitive ratio of randomized algorithms. To that end, we use the results from section 6.5.2 to provide lower bounds on the competitive ratio of randomized algorithms, since algorithms with advice and randomized algorithms are closely related. According to [123], Yao's Principle [124] bounds the expected competitive ratio of randomized algorithms from below with

$$R \geq max \left(min_i \frac{\mathbb{E}_{y(j)}[OPT(\sigma_j)]}{\mathbb{E}_{y(j)}[ALG_i(\sigma_j)]}, min_i \frac{1}{\mathbb{E}_{y(j)}[\frac{ALG_i(\sigma_j)}{OPT(\sigma_j)}]} \right)$$

where both arguments in the maximum function are proven lower bounds. $\mathbb{E}[X]$ is the expectation of a random variable X, OPT is the optimal offline algorithm and ALG_i is the optimal strategy for the input sequence σ_i. Moreover, $y(j)$ denotes the distribution of j and σ_j is an input sequence for all j. We denote by c the expected competitive ratio and by I_x the input sequence. We change the other variables accordingly.

Theorem 6.31. *Every randomized algorithm is* $\Omega\big(\frac{n}{f(n)+\frac{n}{2^{f(n)}}}\big)$*-competitive.*

Proof. Let $\langle 2^{f(n)}; 1\rangle$ be the initial capital distribution and $\mathcal{I} = \{I_x \mid x \in \{0, \ldots, 2^{f(n)} - 1\}\}$ the set of sequences of transactions for $f(n) \in o(n)$ where $I_x = p, (r, l, x + 1), loop$ and $p = (l, r, 1), (l, r, 2), \ldots, (l, r, 2^{f(n)-1})$. *loop* is the movement of the capital back and forth. Then, we define the set of strategies as $\mathcal{A} = \{A_x \mid x \in \{0, \ldots, 2^{f(n)} - 1\}\}$. A non-optimal strategy accepts at most

$f(n)$ transactions, whereas the optimal strategy accepts at most n and at least $n - f(n)$ transactions. We will refer to this in the following calculation as $(*)$.

$$
\begin{aligned}
c &\geq min_i \frac{1}{\mathbb{E}_{y(j)}[\frac{A_i(I_j)}{OPT(I_j)}]} && \text{Yao's principle} \\
&= \frac{1}{max_i \mathbb{E}_{y(j)}[\frac{A_i(I_j)}{OPT(I_j)}]} \\
&\geq \frac{1}{max_i \mathbb{E}_{y(j)}[\frac{A_i(I_j)}{n-f(n)}]} && OPT(I_j) \geq n - f(n) \\
&= \frac{n - f(n)}{max_i \mathbb{E}_{y(j)}[A_i(I_j)]} \\
&\geq \frac{n - f(n)}{\sum\limits_{k=0,k\neq i}^{2^{f(n)}-1} (\frac{1}{2^{f(n)}}(f(n)+1)) + \frac{n}{2^{f(n)}}} && y(j) \text{ uniform and } (*) \\
&= \frac{n - f(n)}{(2^{f(n)} - 1)(\frac{f(n)+1}{2^{f(n)}}) + \frac{n}{2^{f(n)}}} \\
&= \frac{n - f(n)}{f(n) + 1 - \frac{f(n)+1}{2^{f(n)}} + \frac{n}{2^{f(n)}}} \\
&\geq \frac{n - f(n)}{2(f(n) + \frac{n}{2^{f(n)}})}
\end{aligned}
$$

Thus, we have shown that $c \in \Omega(\frac{n}{f(n)+\frac{n}{2^{f(n)}}})$. □

Corollary 6.32. *There is no competitive randomized algorithm against oblivious adversaries.*

Proof. Follows immediately from Theorem 6.31 for $f(n) = o(n)$. □

6.5.4 Resource Augmentation

We now approach the problem from another perspective, and consider a different analysis approach called resource augmentation as described in [125]. In this approach, an online algorithm may have more resources than the optimal offline algorithm in order to improve the performance of an algorithm against an adversary. In our setting, we allow the online algorithm to have $h \geq 1$ times more capital on both sides. An online algorithm starts with $\langle hC_l; hC_r \rangle$ if the optimal offline algorithm starts with the initial capital distribution $\langle C_l; C_r \rangle$. In this section, we discuss the existence of deterministic algorithms for different values of h.

Theorem 6.33. *There is no competitive deterministic algorithm for $h < 2$.*

Proof. Let the initial capital distribution for the online algorithm be $\langle hC; 0\rangle = \langle 2C - \epsilon; 0\rangle$ such that $h = 2 - \frac{\epsilon}{C}$. Then, we define the sequences

$$(l, r, C - \frac{\epsilon}{2}), (r, l, 1), (r, l, 1), \ldots, (r, l, 1)$$
$$(l, r, C - \frac{\epsilon}{2}), (l, r, C), (r, l, C), \ldots, (l, r, C)$$

Assume there is a competitive algorithm that rejects the first transaction. Then, none of the consecutive transactions with low value (first sequence) can be accepted. This contradicts the assumption and implies that if a competitive deterministic algorithm exists, the first transaction must be accepted. Suppose there is a competitive algorithm that accepts the first transaction.

The capital distribution after accepting the first transaction is $\langle C - \frac{\epsilon}{2}; C - \frac{\epsilon}{2}\rangle$. Then, the following transactions in the second sequence cannot be accepted, since the the movement of C back and forth is not possible anymore. This contradicts the assumption that there is a competitive deterministic algorithm that accepts the first transaction. We conclude, that there is no deterministic algorithm for $h = 2 - \frac{\epsilon}{C}$. For $C = n$, $C = 2^n$, or even bigger C, h goes to 2 from below.

Thus, there is no competitive deterministic online algorithm for any $h < 2$. □

6.5.5 Minimizing the Number of Rejected Transactions

So far, we defined the competitiveness as the ratio between the number of accepted transactions of the optimal offline algorithm and an online algorithm, thus as an online maximization problem. In this section, we examine the problem from another point of view; that of a minimization problem. To this end, we define the competitive ratio of an algorithm, denoted by c, as the value that upper bounds the ratio between the cost of the online algorithm, $ALG(I_x)$, and the cost of the optimal offline solution, $OPT(I_x)$, for any input sequence I_x, i.e.,

$$\forall x,\ c \geq \frac{ALG(I_x)}{OPT(I_x)}$$

The cost, in both cases, represents the number of rejected transactions.

Similarly to sections 6.5.2 and 6.5.3, we first lower bound the competitive ratio for a given upper bound on the advice bits and then use this result to show there is no competitive randomized algorithm against oblivious adversaries.

Lemma 6.34. *An algorithm with strictly less than $f(n)$ advice bits has a competitive ratio of $\Omega\big(\frac{n}{f(n)}\big)$.*

Proof. Let $\langle 2^{f(n)}; 1\rangle$ be the initial capital distribution. We define

$$\mathcal{I} = \{I_x \mid x \in \{0, \ldots, 2^{f(n)} - 1\}\}$$

where

$$I_x = (l, r, 1), (l, r, 2), \dots (l, r, 2^{f(n)-1}), (r, l, x+1), loop$$

for $f(n) \in o(n)$ and *loop* being a sequence of transactions moving all the capital from left to right back and forth.

An algorithm that does not accept *loop* has a competitive ratio of at least $c \geq \frac{n-f(n)-1}{f(n)}$. Assume there is an algorithm reading $k < f(n)$ advice bits with a better competitive ratio. Since there are $2^{f(n)}$ different instances in $\mathcal{I}$ and fewer strategies are expressible by the advice, one strategy is used for two elements of the input set. We previously showed that transferring i is the only strategy accepting *loop* for an instance I_i. Then, for I_j where $j \neq i$ the strategy that transfers i with the transactions of the prefix cannot accept *loop*. As a result, the competitive ratio is at least $c \geq \frac{n-f(n)-1}{f(n)}$ which contradicts the assumption that there is an algorithm with a better competitive ratio reading $k < f(n)$ advice bits. □

As stated in [123], Yao's Principle [124] bounds the expected competitive ratio for minimisation problems from below with

$$\bar{R} \geq max\left(min_i \frac{\mathbb{E}_{y(j)}[ALG_i(\sigma_j)]}{\mathbb{E}_{y(j)}[OPT(\sigma_j)]}, min_i \mathbb{E}_{y(j)}[\frac{ALG_i(\sigma_j)}{OPT(\sigma_j)}]\right)$$

where both arguments in the maximum function are proven lower bounds. The notation is the same as in section 6.5.3.

Theorem 6.35. *Every randomised algorithm (minimising the number of rejected transactions) is* $\Omega(\frac{n}{f(n)})$*-competitive.*

Proof. We construct the same set of instances as in the proof of Lemma 6.34 with the same initial capital distribution. Accordingly, we define the set of strategies to be $\mathcal{A} = \{A_x \mid x \in \{0, \dots, 2^{f(n)} - 1\}\}$ where A_x is the optimal strategy for I_x. As discussed in previous proofs, choosing a non-optimal strategy for an input results in the rejection of at least all of the $n - f(n) - 1$ transactions in *loop*. We know that the optimal offline algorithm rejects at most $f(n)$ and at least no transactions.

$$\begin{aligned}
c &\geq \min_i \frac{\mathbb{E}_{y(j)}[A_i(I_j)]}{\mathbb{E}_{y(j)}[OPT(I_j)]} && \text{Yao's Principle} \\
&\geq \min_i \frac{\mathbb{E}_{y(j)}[A_i(I_j)]}{f(n)} \\
&= \min_i \frac{\sum_{j=0, j\neq i}^{2^{f(n)}-1}(A_i(I_j)) + OPT(I_j)}{2^{f(n)} f(n)} && y(j) \text{ uniform} \\
&\geq \frac{(2^{f(n)} - 1)(n - f(n) - 1)}{2^{f(n)} f(n)}
\end{aligned}$$

$$= \frac{(1 - \frac{1}{2^{f(n)}})(n - f(n) - 1)}{f(n)}$$
$$= (\frac{n}{f(n)} - 1 - \frac{1}{f(n)})(1 - \frac{1}{2^{f(n)}}) \in \Omega(\frac{n}{f(n)}) \qquad \square$$

Corollary 6.36. *There is no competitive randomised algorithm (that minimises the number of rejected transactions) against oblivious adversaries.*

Proof. Follows from Theorem 6.35 for $f(n) = 1$. $\square$

6.6 Online PCN Design for Maximum Profit

We assume again that we want to execute all transactions, thus the optimal graph does not contain cycles (Proposition 6.19). Our objective is to minimize the capital, given all transactions will be executed through our payment network. Wlog, we assume the PSP is a node in the network. Similarly to the offline case, we show that constructing a star network to connect the nodes with payment channels is a good solution. Specifically, we present a log-competitive algorithm that takes advantage of the star graph structure.

In Algorithm 17, we gradually form a star where the center is the PSP. At each step, we check whether there is enough capital on the edges to and from the center to execute the current transaction. If the capital on an edge is smaller that the value of the current transaction, we refund the channel and add to the capacity of this edge twice the value of the current transaction.

Theorem 6.37. *Algorithm 17 is $\Theta(\log C_{opt})$-competitive.*

Proof. The star is a 2-approximation to the optimal offline solution, thus we start with a competitive ratio of 2. The way we update the capacities, each time adding twice the value of the transaction if the capacity is less than the transaction's value, yields also a competitive ratio of two on the edges' capacities. Moreover, at each such step we at least double the capacity of an edge thus we reach the edge's optimal capital, C_e, in $\log C_e$ steps. If we sum over all edges, in total we refund the channels at most $(n-1)\log C_{edges}$ times, where n is the number of nodes in the network and C_{edges} the edges' optimal capital of the offline solution. Therefore, Algorithm 17 returns $C \leq (n-1)\log C_{edges} + 4C_{edges}$, while the offline solution requires $C_{opt} = (n-1) + C_{edges}$. This yields a competitive ratio of $\Theta(\log C_{opt})$. $\square$

Algorithm 17: Online Maximum Profit

Data: Online sequence of transactions $t_i = (s_i, r_i, v_i)$.
Result: Capital C.

```
   /* Initialization                                                    */
1  E ← ∅
2  C ← 0
   /* Star formation and capital calculation                            */
3  for each transaction t_i:
4      if s_i is not connected to s:
5          E ← E ∪ (s_i, s) ;                          // s denotes the PSP
6          c_{s_i,s} ← v_i,
7          c_{s,s_i} ← v_i
8          C ← C + 1
9      elif c_{s_i,s} < v_i:
10         c_{s_i,s} ← c_{s_i,s} + v_i
11         c_{s,s_i} ← c_{s,s_i} + v_i
12         C ← C + 1
13     else:
14         c_{s_i,s} ← c_{s_i,s} − v_i
15         c_{s,s_i} ← c_{s,s_i} + v_i
16     For the case of r_i we follow a similar (invert) procedure.
17 for all i ≠ s:
18     C ← C + c_{i,s} + c_{s,i}
19 Return capital C
```

6.7 Conclusion, Limitations, and Extensions

Summary. In this chapter, we introduced the first graph theoretic framework for payment channel networks. We studied the problem for a specific epoch, i.e., for a fixed number of transactions. This restriction is due to privacy issues, such as timing attacks on the payment network that can leak information on the customers' personal data. Our goal was to understand how a central authority, the PSP, would desing a PCN, and what are the obstacles the PSP would face to maximize its profit. To that end, we examined multiple variants of the problem, taking into account diverse or fixed fees, and considering both the offline and online setting.

For the PCN design with diverse fees, we presented algorithms that calculate the optimal fee assignments given a path or a tree as a graph structure. More importantly, we proved the star is a near-optimal solution when we allow an additional node to act as an intermediary for the customers. This result is also

supported by the analysis provided under fixed fees, where we prove that stars achieve an approximation ratio of 2 with respect to the capital. In other words, hubs are not only an implementable and privacy-guaranteed solution, as mentioned in [68–70], but also a satisfactory solution for a PSP from the profit-maximization point of view.

In case the PSP requests for a fixed fee, which seems like a more reasonable assumption, we tried to maximize the profit (the number of accepted transactions minus the number of generated channels) and to minimize the (temporarily locked) capital needed to execute these transactions. Due to its multi-objective nature, there are several variations of this problem. We mainly focused on two interesting variations:

1. How to choose transactions to execute on a single channel with given capital assignments to maximize the profit,

2. How to design a network and assign capital to accept all transactions and minimize the needed capital.

It turns out, both problems are challenging; we showed that the first problem and a variation of the second problem are both NP-hard. We proposed a dynamic programming based algorithm for the single channel problem and showed that it is an FPTAS.

For the online version, we presented a flurry of impossibility results even for the simplest case of choosing transactions in a single channel. In particular, we proved there is no randomized online algorithm against any adaptive online adversary, and even considering algorithms with advice we showed there is no competitive randomized algorithm against an oblivious adversary. Furthermore, we showed that even with twice as much capital there is no competitive deterministic algorithm against an oblivious adversary. Finally, similar impossibility results were proven when the goal is to minimize the number of rejected transactions – instead of maximizing the number of accepted transaction. For the online PCN design for maximum profit, we presented an $O(\log C)$-competitive online algorithm based on the star structure.

Given the numerous impossibility results, we deduce the payment channel network design is a challenging problem from an algorithmic perspective. Although quite similar to existing network design problems on communication networks [126], the nature of payment channels, that allow the capital to transfer back and forth, adds to the complexity of the problem, and hence existing solutions do not apply.

Open questions and conjectures. Although in this chapter we presented numerous results, there are still many questions and conjectures we did not address successfully. Some of them are listed below:

- Is the PCN-DF problem NP-hard? It seems that it is difficult to compute the optimal network and fee assignment even when the graph structure is resticted to a tree.
- PCN-DF problem under capital restrictions: Assuming the existence of a PSP, how would the optimal graph look like and how would the funds be locked on the edges of the graph to facilitate a given sequence of transactions? This problem seems more challenging, since the transaction order matters.
- We conjecture that in the offline PCN design, where the PSP maximizes its profit (all transactions are executed in the tree-structured PCN), finding the tree that minimizes the capital is NP-hard.
- In the online case, which is the most realistic and interesting in practice, can we provide better algorithms? Is the log-competitive ratio tight, or can the PSP design a graph and assign the capital in a more efficient manner? We conjecture that using learning techniques might produce better results in practice.

PCN design with delays. Suppose a transaction that is not executed instantly through the PCN incurs a cost to the PSP. For instance, a customer might be charged less for the delay or they might switch networks with some probability. This model introduces a delay-profit trade-off. Can we design better algorithms, in this model, even in the single channel case?

PCN dynamic design with fees. Suppose payment channels can be refunded without cost (for instance via rebalancing). Now let us assume a sequence of transactions and a specific capital. In this case, we want to design an algorithm that maximizes the profit given the fact that we can open and close channels on-the-fly. We speculate we can employ existing results from network problems on temporal graphs [127] to show hardness of approximation.

Competition. It would be interesting to consider competition between two or multiple PSPs, where each PSP has its own capital restrictions. Logically, the competition will drive the fee for each transaction "just above cost". But it is not clear what is the cost for each PSP, and it seems to depend on the available capital. This opens an entirely new direction of questions that require a game-theoretic approach. Section 7 will shed light on some of these questions.

Chapter 7

Game-Theoretic Analysis of PCNs

7.1 Introduction

7.1.1 Motivation

Payment channel networks, or PCNs, are currently a hot topic in blockchain research. In practice, the first PCNs have been deployed, and are being actively used. Prominent examples are Bitcoin's Lightning network [12, 61] with more than 35,000 active channels, or Ethereum's Raiden network [128].

As Bitcoin's Lightning network is growing quickly, we need to understand these newly forming PCNs. Which channels will be created, and what will the network topology eventually look like? Network creation games [129] are a perfect tool to understand these questions, since they capture the degradation of the network's efficiency when participants act selfishly.

In a network creation game, the incentive of a player is to minimize its cost by choosing to whom to connect. In our model, players weigh the benefits they receive from using payment channels against the channels' creation cost, and selfishly initiate connections to minimize their cost. There are two types of benefits for each player: (i) the forwarding fees the player receives for the transactions it routed through its channels, (ii) the reduced cost for routing the player's transactions through the PCN in comparison to publishing the transactions on the blockchain (blockchain fee). On the other hand, establishing a channel costs the blockchain fee. Thus, a player has to balance all these factors to decide which channels to establish to minimize its cost. Our goal is to gain a meaningful insight on the network structures that will emerge and evaluate their efficiency, in comparison to centralized structures designed by a central authority that previous work has shown to be almost optimal.

7.1.2 Contribution

In this chapter, we provide a game-theoretic approach to analyze the creation of blockchain PCNs. Specifically, we adopt *betweenness centrality*, a natural measure for fees a player is expected to receive by forwarding others' transactions on a path of payment channels. On the other hand, we employ *closeness centrality* as an intuitive proxy for the transaction fees encountered when executing transactions through other players in the network. We reflect the cost of payment channel creation by associating a *price with link creation*. Therefore, we also generalize previous work on network creation games as our model combines both betweenness and closeness centralities.

Under this model, we study the topologies that emerge when players act selfishly. A specific network structure is considered a Nash equilibrium when no player can decrease its cost by unilaterally changing its connections. We examine various such structures and determine the parameter space in which they constitute a Nash equilibrium. Moreover, we determine the social optima depending on the correlation of betweenness and closeness centrality. When possible, we bound the price of anarchy, the ratio of the social costs of the worst Nash equilibrium and the social optimum [130], to obtain insight into the effects of lack of coordination in PCNs when players act selfishly. Furthermore, we briefly discuss the price of stability, the ratio of the social costs of the best Nash equilibrium and the social optimum [131], specifically concerning the parameter values that most accurately represent blockchain PCNs.

7.2 Preliminaries and Model

In this section, we first present the assumptions for our PCN creation game, and then introduce the necessary notation and the game-theoretic model.

7.2.1 PCN Creation Games

In our model, we assume a player single-handedly initiates a channel to a subset of other players. Incoming channels are always accepted and once installed, the channels can be used to send money in both directions (from sender to receiver, and vice versa). While any player can typically choose the amount to lock in a channel, we assume that the locked capital placed in all channels is high enough to be modeled as unlimited. In particular, we assume that all players are major (large companies, financial institutions etc.) that have thus access to large amounts of temporary capital. It is natural to assume only major players to participate in the network creation game. Typically, a market is created when there is demand for a service. Thus eventually, the PCN will be dominated by service providers that will individually connect with clients and act as intermediaries for all transactions.

In this work, we only consider the flow of transactions through these service providers. Therefore, the cost of opening a channel in our model solely reflects the permanent cost, i. e., the blockchain fee, and is set to 1 (wlog). Furthermore, since we assume major players only, the transactions between the players can be considered uniform.

Together, the payment channels form a PCN. In the network, players receive a payment when transactions are routed through their channels. This payment is a transaction fee, which is typically proportional to the value of the routed transaction, to compensate the intermediate node for the loss of its channel's capital capacity. However, we consider a fixed fee for all nodes, independent of the routed value, since we assume unlimited channel capital. Furthermore, the channel funds cannot deplete as we assumed all channels are funded with unlimited capital. This means that all paths in the PCN between sender and receiver are viable.

7.2.2 Formal Model

A payment channel network (PCN) can be formally expressed by an unweighted undirected graph consisting of V nodes, representing the set of players, and E edges, representing the set of payment channels between the players.

A PCN game consists of n players $V = \{0, 1, \dots, n-1\}$, denoted by $[n]$. The strategy of player u expresses the channels it chooses to open and is denoted by s_u, and the set $S_u = 2^{[n]-\{u\}}$ defines u's strategy space. We denote by $G[s]$ the underlying undirected graph of $G_0[s] = \left([n], \bigcup_{u\in[n]}\{u\} \times s_u\right)$, where $s = (s_0, \dots, s_{n-1}) \in S_0 \times \cdots \times S_{n-1}$ is a strategy combination. Note that while a channel can possibly be created by both endpoints, this will never be the case in a Nash equilibrium.

Betweenness centrality. The fees received by a player for providing gateway services to other players' transactions are modeled by the player's betweenness centrality. Betweenness centrality was first introduced as a measure of a player's importance in a social network by Freeman et al. [132]. According to [132], the betweenness centrality of a player u in a graph $G(V, E)$ is $\sum_{\substack{s,r\in V \\ s\neq r\neq u, m(s,r)>0}} \frac{m_u(s,r)}{m(s,r)}$, where $m_u(s, r)$ is the number of shortest paths between sender s and receiver r that route through player u and $m(s, r)$ is the total number of shortest paths between s and r. Additionally, $s \neq r \neq u$ indicates that $s \neq r$, $s \neq u$ and $r \neq u$.

Intuitively, the betweenness centrality of player u is a measure of the expected number of sender and receiver pairs that would choose to route their transactions through the player in a PCN. Providing an insight into the transaction fees a player is expected to receive, the betweenness centrality lends itself to reflect

the motivation of a player in a PCN to maximize the payments secured through providing transaction gateway services.

However, in our model, the betweenness of player u is measured as follows:

$$\text{betweenness}_u(s) = (n-1)(n-2) - \sum_{\substack{s,r\in[n]:\\ s\neq r\neq u, m(s,r)>0}} \frac{m_u(s,r)}{m(s,r)}.$$

We subtract u's betweenness centrality, as defined by Freeman et al. [132], from u's maximum possible betweenness centrality to ensure that the social cost is always positive – avoiding cases where price of anarchy is undefined.

Closeness centrality. Furthermore, we model the fees encountered by a player when having its transactions routed through the network with its closeness centrality. Closeness centrality measures the sum of distances of player u to all other players and is given by

$$\text{closeness}_u(s) = \sum_{r\in[n]-u} \left(d_{G[s]}(u,r) - 1\right),$$

for a player u, where $d_{G[s]}(u,r)$ is the distance between u and r in the graph $G[s]$. With the transaction fees fixed per edge in our model, the distance to a player r estimates the costs encountered by player u when sending a transaction to player r. Therefore, the sum of distances to all other players is a natural proxy for the fees u faces for making transactions when assuming uniform transactions.

Thus, the combination of betweenness and closeness centralities accurately encapsulates the incentives inherent to players in a blockchain PCN.

Cost. The cost of player u under the strategy combination s is $\text{cost}_u(s) = |s_u| + b \cdot \text{betweenness}_u(s) + c \cdot \text{closeness}_u(s)$, where $b \geq 0$ is the betweenness weight and $c > 0$ the closeness weight. Letting $c > 0$ ensures that the graph is always connected, as a player's cost is infinite in a disconnected graph. Additionally, the model assumes the same price for all nodes and embeds this into coefficients b and c. While this does not exactly encapsulate reality, it is a reasonable assumption. Different paths offer similar services to payers. In such a setting, Bertrand competition [133] suggests that competition will drive the prices from different players to be within a close region of each other.

Social optimum. The objective of player u is to minimize its cost, $\min_{s_u} \text{cost}_u(s)$. The social cost is the sum off all players' costs, $\text{cost}(s) = \sum_{u\in[n]} \text{cost}_u(s)$. Thus, the social optimum is $\min_s \text{cost}(s)$.

7.3 PCN Creation Game

To gain an insight into the efficiency of emerging topologies when players act selfishly, we will first analyze the social optimum for our model. After studying if and when prominent graphs are Nash equilibria, we conclude by bounding the price of anarchy and the price of stability.

7.3.1 Social Optimum

By the definition of the cost function, the social cost is

$$\text{cost}(s) = \sum_{u\in[n]} \text{cost}_u(s) = |E(G)| + b \sum_{u\in[n]} \text{betweenness}_u(s) + c \sum_{u\in[n]} \text{closeness}_u(s),$$

for any graph where no channel is paid by both endpoints. This constraint is met for all Nash equilibria. To lower bound the social cost, we will first simplify the social cost expression. Lemma 7.2 shows how to express the social cost directly in terms of the number of edges and the sum of the players' closeness centrality costs, facilitating further analysis.

Lemma 7.1 (Theorem 1 [134])**.** *The average betweenness $\overline{B}(G)$ in a connected graph G can be expressed as: $\overline{B}(G) = (n-1)(\bar{l}(G)-1)$, where $\bar{l}(G)$ is the average distance in G.*

Lemma 7.1 is proven in [134] and relates the average betweenness and distance in a connected graph. We take advantage of Lemma 7.1 to simplify the social cost expression.

Lemma 7.2. *The social cost in G is given by $cost(s) = |E(G)| + b \cdot n \cdot (n-1)(n-2) + (c-b) \cdot \sum_{u\in[n]} closeness_u(s)$.*

Proof. According to Lemma 7.1 the social cost can be expressed as follows for all $b \geq 0$ and $c > 0$.

$$\begin{aligned}
\text{cost}(s) =& |E(G)| + b \sum_{u\in[n]} \text{betweenness}(u) + c \sum_{u\in[n]} \text{closeness}(u) \\
=& |E(G)| + b \sum_{u\in[n]} \left((n-1)(n-2) - \sum_{\substack{s,r\in[n]: \\ s\neq r\neq u, \\ m(s,r)>0}} \frac{m_u(s,r)}{m(s,r)} \right) \\
&+ c \sum_{u\in[n]} \sum_{r\in[n]-u} \left(d_{G[s]}(u,r) - 1 \right)
\end{aligned}$$

$$
\begin{aligned}
&= |E(G)| + b \cdot n \cdot (n-1)(n-2) - b \cdot n \cdot \overline{B}(G) + c \cdot n \cdot (n-1)(\bar{l}(G) - 1) \\
&= |E(G)| + b \cdot n \cdot (n-1)(n-2) + (c-b) \cdot n \cdot (n-1)(\bar{l}(G) - 1) \\
&= |E(G)| + b \cdot n \cdot (n-1)(n-2) + (c-b) \cdot \sum_{u \in [n]} \sum_{r \in [n]-u} \left(d_{G[s]}(u,r) - 1 \right) \quad \square
\end{aligned}
$$

The distance of a vertex v of a connected graph G is $d(v) := \sum_{u \in [n]-v} d_G(v,u)$. The distance of a connected graph G is $d(G) := \sum_{v \in [n]} d(v)/2$. If G is not connected, then $d(v) = \infty$ for any v, and $d(G) = \infty$.

Lemma 7.3 (Theorem 2.3 [135]). *If G is a connected graph with n vertices and k edges then $n \cdot (n-1) \leq d(G) + k \leq \frac{1}{6} \cdot \left(n^3 - 5 \cdot n - 6 \right)$.*

Lemma 7.3 provides bounds for the distance of a graph G,

$$
d(G) = \frac{1}{2} \sum_{u \in [n]} \sum_{r \in [n]-u} d_G(u,r)
$$

which is useful for finding the social optimum for our game. In [135] Lemma 7.3 is proven and the path graph achieves the upper bound. This can be used to find the social optimum. Dependent on the weights b and c, the social optimum for our PCN creation game is given in Theorem 7.4 and Figure 7.1 illustrates this in the parameter space.

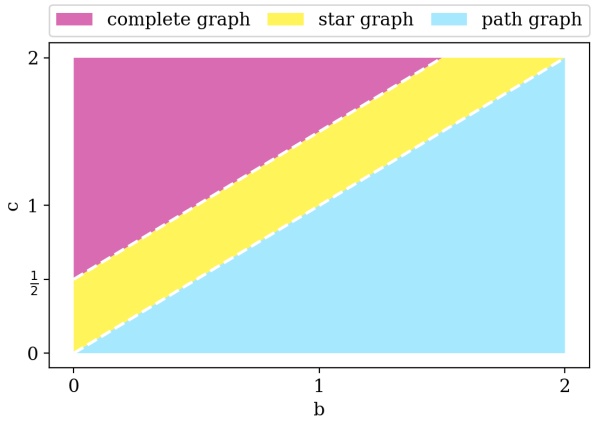


Figure 7.1: Parameter map for social optimum of game.

Theorem 7.4. *The social optimum is a complete graph for $c > \frac{1}{2} + b$, a star graph for $b \leq c \leq \frac{1}{2} + b$ and a path graph for $c < b$.*

Proof. Using Lemma 7.2 we can lower bound the social cost for $c \geq b$ as follows:

$$\begin{aligned}\text{cost}(s) =& |E(G)| + b \cdot n \cdot (n-1)(n-2) + \underbrace{(c-b)}_{\geq 0} \sum_{u \in [n]} \sum_{r \in [n]-u} \left(d_{G[s]}(u,r) - 1\right) \\ \geq& |E(G)| + b \cdot n \cdot (n-1)(n-2) + (c-b)(n \cdot (n-1) - 2|E(G)|) \\ =& (1 - 2 \cdot (c-b)) \cdot |E(G)| + b \cdot n \cdot (n-1)(n-2) + (c-b)(n \cdot (n-1))\end{aligned}$$

since every pair of nodes that is not connected by an edge is at least distance 2 apart [129]. This lower bound is achieved by any graph with diameter at most 2. It follows that for $c > \frac{1}{2} + b$ the social optimum is a complete graph, maximizing $|E|$, and for $b \leq c \leq \frac{1}{2} + b$ the social optimum is a star, minimizing $|E|$.

To find the social optimum for $c < b$, we rewrite the social cost as

$$\begin{aligned}\text{cost}(s) =& |E(G)| + b \cdot n \cdot (n-1)(n-2) - (b-c) \cdot \sum_{u \in [n]} \sum_{r \in [n]-u} \left(d_{G[s]}(u,r) - 1\right) \\ =& |E(G)| - 2 \cdot (b-c) \cdot d(G) + b \cdot n \cdot (n-1)(n-2) + (b-c) \cdot n \cdot (n-1)\end{aligned}$$

For a connected graph the social cost is then minimized for a tree, as $|E(H)| - a \cdot d(H) > |E(G)| - a \cdot d(G)$ if G is a subgraph of H and $a > 0$. For any tree, the number of edges is $n - 1$. Using Lemma 7.3, we get that

$$\begin{aligned}\text{cost}(s) =& |E(G)| + b \cdot n \cdot (n-1)(n-2) - (b-c) \sum_{u \in [n]} \sum_{r \in [n]-u} \left(d_{G[s]}(u,r) - 1\right) \\ \geq& \left(1 + \left(\frac{2}{3}b + \frac{1}{3}c\right) n \cdot (n-2)\right)(n-1)\end{aligned}$$

is a lower bound for the social cost which is achieved by a path graph. □

In areas most accurately describing PCNs, we expect the weights b and c to be smaller than the cost of channel creation and close to each other. For these cases, we observe *the star graph is the social optimum.*

7.3.2 Nash Equilibria

To find a Nash equilibrium, one could follow a naive approach: start with a fixed graph structure and then continuously compute a player's best response in the game. However, Theorem 7.5 shows that it is NP-hard to calculate a player's best response.

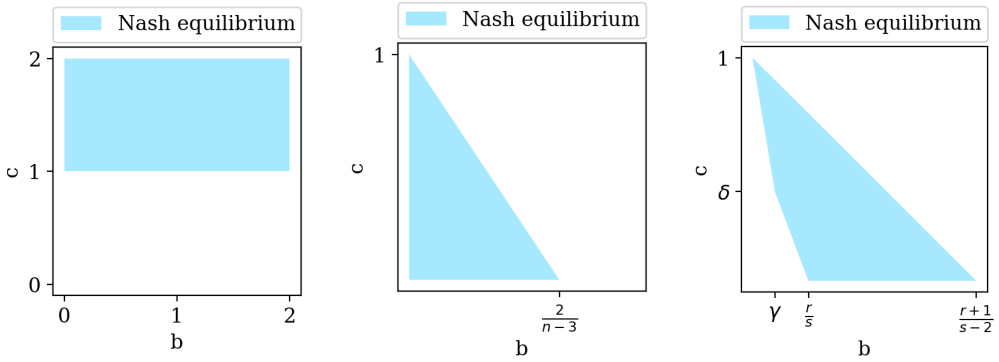
Theorem 7.5. *Given a strategy $s \in S_0 \times \cdots \times S_{n-1}$ and $u \in [n]$, it is NP-hard to computed the best response of u.*

Proof. The following proof is adapted from Proposition 1 in [129].

Given the configuration of the rest of the graph, player u has to compute her best response; a subset of players to build channels to such that her cost is minimized. For $b = 0$ and $0.5 < c < 1$ and no incoming links from the rest of the graph, we know that the diameter of G can be at most 2. Additionally, making more than the minimum number of required links, only improves the distance term by c, which is strictly smaller than the cost of establishing a link. Thus, u's strategy is a dominating set for the rest of the graph.

The cost of u is minimized when the size of the subset is minimized. The minimum size dominating set corresponds to u's best response. Hence, it is NP-hard to compute a player's best response by reduction from the dominating set. □

Therefore, with this in mind, we analyze prominent graph topologies theoretically, to see if and when they are Nash equilibria in our game. The results are illustrated in Figure 7.2.



(a) complete graph (b) star graph ($n \geq 4$) (c) biclique

Figure 7.2: Parameter map for prominent graphs. In Figure 7.2c, r and s are the subset sizes ($3 \leq r \leq s$). With $\alpha = \frac{s \cdot (s-1)}{r \cdot (s-2)}$ and $\beta = \frac{1}{s-r+1}\left(\frac{s \cdot (s-1)}{r} - \frac{(r-2)(r-1)}{s+1}\right)$, (γ, δ) is the intersection between $1 = \frac{s}{r}b + \frac{s+r-3}{s-1}c$ and $1 = \min\{\alpha, \beta\} \cdot b + c$.

Complete Graph

For large values of c the complete graph is the only Nash equilibrium as stated in Theorem 7.6. Additionally, the complete graph is also a Nash equilibrium for $c = 1$, but it is not necessarily the only one. However, for small values of c, which are the values we expect to encounter in a PCN creation game, the complete graph is not a Nash equilibrium, as stated in Theorem 7.7.

Theorem 7.6. *For $c > 1$, the only Nash equilibrium is the complete graph.*

Proof. The addition of an edge by a player never increases its betweenness cost. Thus, by the definition of the cost function any Nash equilibrium cannot be missing any edges whose addition would reduce a players closeness by more than 1, the cost of building an edge. As $c > 1$, no edge can be missing in the graph and the only Nash equilibrium is the complete graph. □

Theorem 7.7. *For $c < 1$ and $n \geq 3$, the complete graph is never a Nash equilibrium.*

Proof. In a complete graph the removal of an edge by a player does not change its betweenness cost and its closeness cost is increased by c. Thus, the cost of a player would decrease when removing one edge. Therefore, the complete graph is not a Nash equilibrium for $c < 1$. □

Figure 7.2a visualizes the combination of these results, i. e., when the complete graph is a Nash equilibrium in our game. We observe that for some weight combinations the complete graph is both the social optimum and a Nash equilibrium. However, most PCNs are not expected to fall into this area of the parameter space.

Path Graph

While the path graph is the social optimum for a significant area of the parameter space, we show it can only be a Nash equilibrium for small sets of players. For $n = 3$, the path graph is a Nash equilibrium for all $c \leq 1$, as it is the only possible connected graph that is not the complete graph.

Proposition 7.8. *For $n = 4$, the path graph is a Nash equilibrium if and only if $1 \leq b + 2 \cdot c$.*

Proof. In a game with four players, a path graph with outgoing links from the endpoints is never a Nash equilibrium. Such an endpoint could simply reduce the player's cost by c through exchanging its current channel with a channel to the player currently two edges away.

In the remaining path graphs, all channels are initiated by the central nodes. A player can never increase its cost by removing or exchanging edges. The minimum change of cost can be achieved by an endpoint initiating a channel to the other endpoint. This change in cost is given by

$$\Delta\text{cost}(\text{add 1 link}) = 1 - b - 2 \cdot c.$$

We follow that for $n = 4$, the path graph is a Nash equilibrium for $1 \leq b + 2 \cdot c$. □

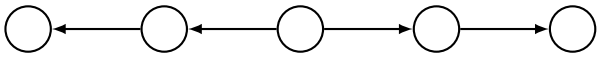


Figure 7.3: Path graph.

Proposition 7.9. *For $n = 5$, the path graph is a Nash equilibrium if and only if $1 \leq 2 \cdot b + 4 \cdot c$.*

Proof. As in the case with four players, a path graph with five nodes in which the endpoints initiate channels is never a Nash equilibrium. For instance, an endpoint could reduce its cost by $2 \cdot c$ through connecting to the player currently three edges away instead.

Additionally, in a path graph with five players, the players neighboring the endpoints cannot initiate channels to the central node in a Nash equilibrium. Replacing such a link with a link to the other neighbor of the central node would lead to a cost reduction of c.

Thus, it only remains to consider a path graph with channels initiated as shown in Figure 7.3.

Here, the minimum change in cost is achieved by an endpoint initiating a new channel to the player four edges away. We have

$$\Delta\text{cost}(\text{add 1 link}) = 1 - 2 \cdot b - 4 \cdot c.$$

The path graph with five nodes is a Nash equilibrium for $1 \leq 2 \cdot b + 4 \cdot c$. □

Propositions 7.8 and 7.9 identify when the path graph is a Nash equilibrium for networks with four and five players respectively. These bounds partly overlap with areas in which the path graph is the social optimum. While this partial correspondence between the Nash equilibrium and social optimum appears promising for the coordination of our game, Theorem 7.10 suggests to the contrary.

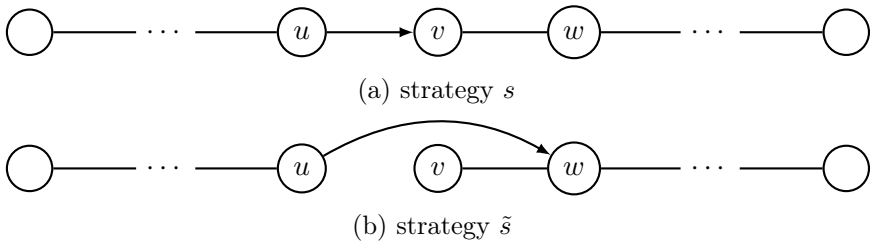
Theorem 7.10. *For $n \geq 6$, the path graph is never a Nash equilibrium.*

Proof. To show that the path graph is never a Nash equilibrium for $n \geq 6$, we will show that at least one player in a path graph consisting of more than six players can always reduce its cost by changing strategy.

In a path graph with at least six players, at least one player u has an outgoing edge to a player v at least two steps from the end of the path on the opposite side of player u. This is illustrated in Figure 7.4a and we consider this to be strategy s. In this case it is always more beneficial for player u to connect to player w instead of player v. Let's refer to this strategy as strategy $\tilde{s}$ (Figure 7.4b).

The change in cost for this strategy is given as

$$\Delta\text{cost}_u(s \text{ to } \tilde{s}) = -c \cdot (m - 2),$$



(a) strategy s

(b) strategy $\tilde{s}$

Figure 7.4: Strategy deviation of player 1.

where m is the number of edges player v is away from the endpoint on the opposite side u. Thus, the change in cost is negative and the path graph cannot be a Nash equilibrium for $n \geq 6$. □

Hence, the path graph is not expected to be a Nash equilibrium for PCNs that typically consist of many nodes.

Circle Graph

The results we find for the circle graph are similar to those for the path graph. For small values of n, the circle graph can be a Nash equilibrium depending on the weights b and c. The circle graph and the complete graph are the same for $n = 3$. Thus, for $n = 3$ the circle graph is a Nash equilibrium if and only if $c \geq 1$.

Proposition 7.11. *For $n = 4$, the circle graph is a Nash equilibrium if and only if $c \leq 1 \leq b + 2 \cdot c$.*

Proof. Adding a link to the only player one is not directly connected to, does not decrease a player's betweenness cost. It is not beneficial for a player to initiate an additional link, if the closeness cost reduction is not bigger than the link cost of one. Thus, a player does not add an additional edge for

$$c \leq 1.$$

A player's change in cost when removing a single link is given by

$$\Delta\text{cost}_u(\text{remove 1 link}) = -1 + b + 2 \cdot c.$$

Hence, a player cannot reduce its cost through the removal of a single link if $1 \leq b + 2 \cdot c$.

Additionally, a player with two outgoing links will never eliminate both without adding another link. The player's cost would be infinite otherwise. Exchanging a single link with a new link to the player one was not previously connected to, never yields a negative change in cost and is therefore never a player's best response.

Finally, the change in cost when removing two links and adding a new link to the player one was not previously connected to is given by

$$\Delta\text{cost}_u(\text{remove 2 \& add 1 link}) = -1 + b + c,$$

but this bound is more restrictive than the previous one, and there is no need for a player to have more than one outgoing edge.

Thus, the circle graph with $n = 4$ is a Nash equilibrium for $c \leq 1 \leq b+2\cdot c$. □

Proposition 7.12. *For $n = 5$, the circle graph is a Nash equilibrium if and only if $b + c \leq 1 \leq 2 \cdot b + 4 \cdot c$.*

Proof. The change in cost for the addition of links to the players, a player was not directly connected to previously, is given by

$$\Delta\text{cost}_u(\text{add } m \text{ links}) = m - m \cdot b - m \cdot c,$$

where $m \in \{1, 2\}$. Thus, a player can reduce its cost by adding more links when $1 \leq b + c$.

If a player in the circle graph removes one outgoing link the change in cost is

$$\Delta\text{cost}_u(\text{remove 1 link}) = -1 + 2 \cdot b + 4 \cdot c.$$

A player with an outgoing link benefits from the removal if $1 \geq 2 \cdot b + 4 \cdot c$. On the other hand, a player with two outgoing edges will never remove both links without adding a new link as the graph would become disconnected otherwise. Additionally, the player never benefits more from exchanging links as the change in cost is non-negative. When replacing both player's links by a new link, the change in cost is

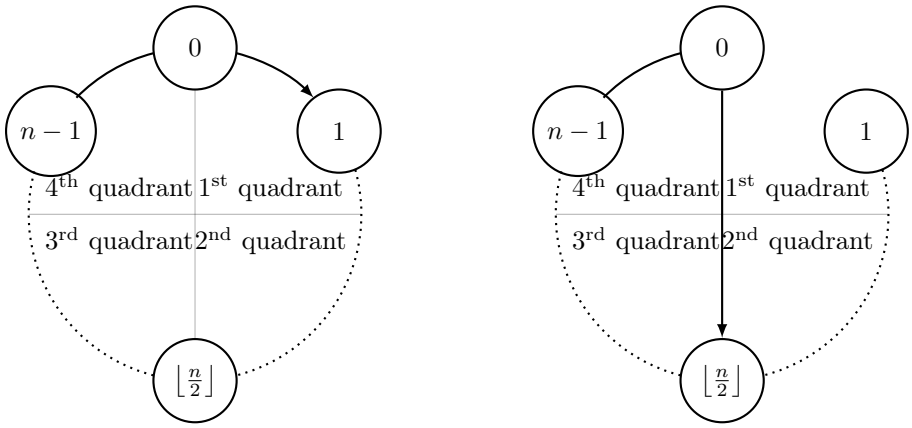
$$\Delta\text{cost}_u(\text{remove 2 \& add 1 link}) = -1 + 2 \cdot b + 2 \cdot c.$$

However, this leads to a more restrictive bound than just removing one link and no player in a circle graph needs more than one outgoing link.

We have shown that for $n = 5$ the circle graph is a Nash equilibrium if and only if $b + c \leq 1 \leq 2 \cdot b + 4 \cdot c$. □

Propositions 7.11 and 7.12 show that for small n, the circle graph can be a Nash equilibrium depending on the weights b and c. However, for large n the circle graph is never a Nash equilibrium, as stated in Theorem 7.13.

Theorem 7.13. *There exists a $N > 0$, such that for all $n \geq N$ the circle graph is never a Nash equilibrium.*



(a) strategy s (b) strategy $\tilde{s}$

Figure 7.5: Strategy change of player 0.

Proof. We will show that any player with one outgoing edge in a circle graph with $n \geq N$ players, has an incentive to change strategy. Thus, the circle graph cannot be a Nash equilibrium.

Consider the circle graph in Figure 7.5a. Without loss of generality, assume that player 0 has one outgoing edge to player 1. As the equations for 0's betweenness and closeness differ for n even or odd, we will use asymptotic notation throughout the following analysis.

In the circle graph (strategy s), the betweenness of 0 is

$$\text{betweenness}_0(s) = \frac{3}{4} \cdot n^2 + o\left(n^2\right)$$

and the 0's closeness is

$$\text{closeness}_0(s) = \frac{1}{4} \cdot n^2 + o\left(n^2\right).$$

Now, player 0 removes the link to player 1 and initiates a new link to player $\lfloor \frac{n}{2} \rfloor$, seen in Figure 7.5b. We will refer to this strategy as $\tilde{s}$. The first part of 0's betweenness cost reduction comes from the shortest paths of players in the 1st and 2nd quadrant to the 4th quadrant, as well as the other way around; the quadrants are as shown in Figure 7.5b. These shortest paths go through the shortcut and subtract

$$2 \cdot \left(\frac{n}{2} + o(n)\right) \cdot \left(\frac{n}{4} + o(n)\right) = \frac{n^2}{4} + o\left(n^2\right),$$

from 0's betweenness cost. The second part stems from nodes in the 1st and 3rd quadrant using node 0 as a gateway in the cycle. We have a further betweenness cost reduction of

$$\frac{1}{4} \cdot \left(\frac{n}{2}\right)^2 + o\left(n^2\right) = \frac{n^2}{16} + o\left(n^2\right).$$

Thus, the betweenness of 0 with strategy $\tilde{s}$ is at most

$$\text{betweenness}_0(\tilde{s}) = n^2 - \frac{n^2}{4} - \frac{n^2}{16} + o\left(n^2\right) = \frac{11}{16} \cdot n^2 + o\left(n^2\right).$$

The closeness of player 0 to players in the 3rd and 4th quadrant is

$$\frac{1}{4} \cdot \left(\frac{n}{2}\right)^2 + o\left(n^2\right) = \frac{n^2}{16} + o\left(n^2\right),$$

and to players in the 1st and 2nd quadrant 0's closeness is

$$\frac{1}{2} \cdot \left(\frac{n}{2}\right)^2 + o\left(n^2\right) = \frac{n^2}{8} + o\left(n^2\right).$$

Therefore, 0's closeness is

$$\text{closeness}_0(\tilde{s}) = \frac{n^2}{16} + \frac{n^2}{8} + o\left(n^2\right) = \frac{3}{16} \cdot n^2 + o\left(n^2\right).$$

Player 0's change in cost is

$$\begin{aligned}\Delta\text{cost}_u(s \text{ to } \tilde{s}) &= \left(\frac{11}{16}n^2 - \frac{3}{4}n^2 + o\left(n^2\right)\right) \cdot b + \left(\frac{3}{16}n^2 - \frac{1}{4}n^2 + o\left(n^2\right)\right) \cdot c \\ &= -\left(\frac{1}{16}n^2 + o\left(n^2\right)\right)(b + c).\end{aligned}$$

As player 0 would choose strategy $\tilde{s}$ over strategy s for $\Delta\text{cost}_u(s \text{ to } \tilde{s}) < 0$, there exists a $N > 0$, such that for $n \geq N$ player the circle graph is never a Nash equilibrium. □

Star Graph

The star graph is the social optimum for a significant part of our parameter space. In a star graph the player in the center has minimal closeness and betweenness costs; all other players have maximal betweenness cost. While this does not directly appear to be a stable network, Theorem 7.14 suggests that the star graph is a Nash equilibrium for smaller values of b and c. These results are depicted in Figure 7.2b.

Theorem 7.14. *For $n \geq 4$, the star graph is always a Nash equilibrium if and only if $0 \leq 1 - \frac{n-3}{2}b - c$.*

Proof. To show that the star is always a Nash equilibrium for $n \geq 4$ and $0 \leq 1 - \frac{n-3}{2}b - c$, we will consider a star graph consisting of n players $V = \{0, 1, \ldots, n-1\}$. Without loss of generality we assume that player 0 is the center of the star.

No player in the star graph has an incentive to remove an edge, as this would lead to infinite cost. Thus, player 0 has no incentive to change strategy, as it is connected to everyone.

Next we consider star graphs where all links are initiated by player 0 and star graphs where at least one link is initiated by another player separately.

If all links are initiated by player 0, players $1, 2, \ldots, n-1$ are all in an equivalent position and it is therefore sufficient to solely consider player 1. Player 1 would only add links, if this leads to a decrease in its cost. Initiating an edge to player 0 would only increase its cost. Additionally, for the remaining $n-2$ players, it only matters to how many player 1 connects. The change in cost when adding m, where $1 \leq m \leq n-2$, edges is given by $\Delta\text{cost}_1(\text{add } m \text{ links}) = m - \frac{m \cdot (m-1)}{2} b - m \cdot c$. Thus, player 1 will change strategy if $\Delta\text{cost}_1(\text{add } m \text{ links}) < 0$. The change in cost is minimized for $m = n-2$.

In star graphs where at least one player other than 0 initiates a link, players that have no outgoing links are in the same position as those analyzed previously. Thus, it suffices to consider player i, where $i \neq 0$, that has one outgoing link. In addition to only initiating new links, player i can remove the link to player 0 and initiates l, where $1 \leq l \leq n-2$, new links. The change in cost is then given as $\Delta\text{cost}_i(\text{add } l \text{ links}) = (l-1) - \frac{l \cdot (l-1)}{2} b - (l-1) \cdot c$. However, this leads to more restrictive bounds and there is no need for players other than player 0 to have outgoing links.

Thus, the star is a Nash equilibrium if and only if $0 \leq 1 - \frac{n-3}{2} b - c$. □

We note that the areas where the star is both a Nash equilibrium and the social optimum overlap partially.

Complete Bipartite Graph

The star graph is a complete bipartite graph where one group has size one. In this section, we analyze more general complete bipartite graphs or bicliques $K_{r,s}$, where r is the size of the smaller subset and s is the size of the larger subset. In a complete bipartite graph, every node from one subset is connected to all nodes from the other subset.

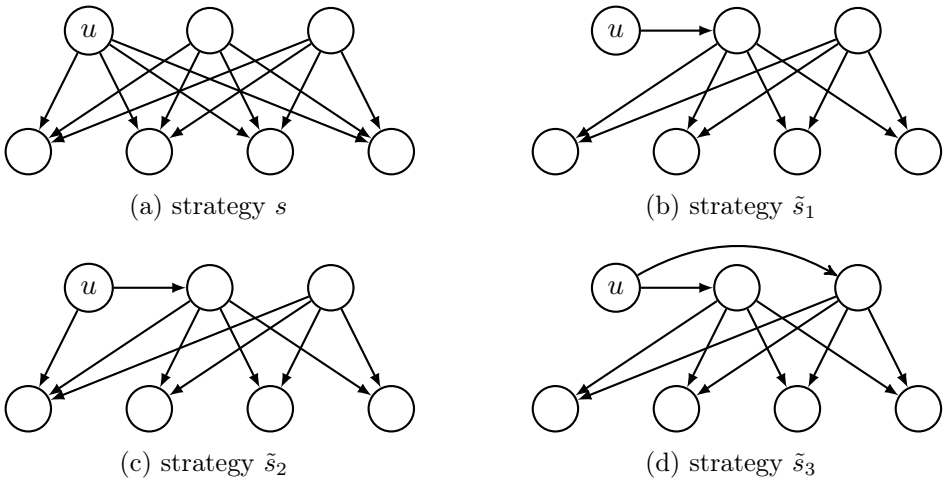
Theorem 7.15. *The complete bipartite graph $K_{r,s}$ with $3 \leq r \leq s$ is stable if and only if $\frac{s-2}{r+1} b + c \leq 1 \leq \min\left\{\frac{s}{r} b + \frac{s+r-3}{s-1} c, \min\{\alpha, \beta\} \cdot b + c\right\}$, where $\alpha = \frac{s \cdot (s-1)}{r \cdot (s-2)}$ and $\beta = \frac{1}{s-r+1}\left(\frac{s \cdot (s-1)}{r} - \frac{(r-2)(r-1)}{s+1}\right)$.*

Proof. Additional links can only be created within a subset in a complete bipartite graph. Similarly to adding links in a star graph, the change in cost when adding m

links is given by $\Delta\text{cost}_u(\text{add } m \text{ links}) = m - \frac{m \cdot (m-1)}{l+1} b - m \cdot c$, where $l \in \{r, s\}$ is the size of the subset not including the player.

A player changes strategy when $\Delta\text{cost}_u(\text{add } m \text{ links}) < 0$. The change in cost is minimized when m is maximized and $l = r$. m can therefore be $s-1$ at most. Thus, the upper bound for $K_{r,s}$ being a Nash equilibrium is $1 \geq \frac{s-2}{r+1} b + c$.

Players in the subset of size r, benefit more from a link to the other subset, as their betweenness cost is smaller. Thus, players from the larger subset with outgoing links would change strategy sooner. In the case where the subsets are of equal size, the link direction does not matter. Hence, to find a lower bound for b and c we only consider complete bipartite graphs, in which all links are established from the smaller subset, as seen in Figure 7.6a. Without loss of generality we will only consider player u in the following analysis. It is not reasonable for player u to remove all its links without adding any new links, as its cost would become infinite. Depending on the other parameters, it might be optimal to remove all its previous links and only connect to one player in its subset (Figure 7.6b), connect to one player in its subset and one player from the other subset (Figure 7.6c), or to remove all its previous links and instead connect to all other players in its subset (Figure 7.6d).



(a) strategy s (b) strategy $\tilde{s}_1$

(c) strategy $\tilde{s}_2$ (d) strategy $\tilde{s}_3$

Figure 7.6: Strategy deviations of player u.

When player u changes to strategy $\tilde{s}_1$, seen in Figure 7.6b the change in cost is as follows:

$$\Delta\text{cost}_u(s \text{ to } \tilde{s}_1) = -(s-1) + \frac{s \cdot (s-1)}{r} b + (s + r - 3) \cdot c$$

as player u initiates $s-1$ less links than before, losing all its previous betweenness. Additionally, player u is one edge further away from all other players except for

the one the player connects to directly. Thus, the above strategy is less preferable than the complete bipartite graph for player u, if

$$1 \leq \frac{s}{r}b + \frac{s+r-3}{s-1}c.$$

Player u's change to strategy $\tilde{s}_2$ (Figure 7.6c) leads to $s-2$ less links initiated by her. The player is further away from $s-1$ players from the other subset and closer to one in its own. All transaction-routing potential is lost. Therefore, the change in cost is given by

$$\Delta\text{cost}_u(s \text{ to } \tilde{s}_2) = 2 - s + \left(\frac{s\cdot(s-1)}{r}\right)b + (s-2)\cdot c.$$

Hence, for this strategy to be less preferable than the complete bipartite graph,

$$1 \leq \left(\frac{s\cdot(s-1)}{r\cdot(s-2)}\right)b + c = \alpha\cdot b + c.$$

When severing all previous links and connecting to all players in its subset instead, strategy $\tilde{s}_3$ (Figure 7.6d), player u builds $s-r+1$ less links than before. Furthermore, player u is closer to players previously in its own subset and further away from the rest. While player u can now transmit transactions of players previously in the player's subset, player u is no longer a preferable intermediary for players previously in the other subset. Therefore, the change in cost is given by

$$\Delta\text{cost}_u(s \text{ to } \tilde{s}_3) = r - s + 1 + \left(\frac{s\cdot(s-1)}{r} - \frac{(r-1)(r-2)}{s+1}\right)b + (s-r+1)\cdot c.$$

Hence, for this strategy to be less preferable than the complete bipartite graph for player u,

$$1 \leq \frac{1}{(s-r+1)}\left(\frac{s\cdot(s-1)}{r} - \frac{(r-1)(r-2)}{s+1}\right)b + c = \beta\cdot b + c.$$

To summarize, the complete bipartite graph $K_{r,s}$ is a Nash equilibrium for

$$\frac{s-2}{r+1}b + c \leq 1 \leq \min\left\{\frac{s}{r}b + \frac{s+r-3}{s-1}c, \min\{\alpha,\beta\}\cdot b + c\right\}. \qquad \square$$

The parameter map for the complete bipartite graph is drawn in Figure 7.2c. There (γ,δ) is the intersection between $1 = \frac{s}{r}b + \frac{s+r-3}{s-1}c$ and $1 = \min\{\alpha,\beta\}\cdot b + c$.

7.3.3 Price of Anarchy

The ratio between the social optimum and the worst Nash equilibrium is the price of anarchy (PoA), formally,

$$\text{PoA} = \frac{\max_{s\in N}\text{cost}(s)}{\min_{s\in S}\text{cost}(s)},$$

here S is the set of all strategies and N is the set of strategies that are Nash equilibria.

The price of anarchy provides an insight to the effects of lack of coordination, i. e., measures the performance degradation of the system when players act selfishly in comparison to central coordination. When the price of anarchy is low, selfish actors do not heavily degrade network efficiency. In contrast, a high price of anarchy indicates that network formation by a central authority would significantly increase efficiency.

For $c > 1$, we can determine the price of anarchy exactly, as we established both the social optimum and the (unique) Nash equilibria for $c > 1$.

Corollary 7.16. *For $c > 1$ and $c > \frac{1}{2} + b$, the price of anarchy is $PoA = 1$.*

Proof. The only Nash equilibrium for $c > 1$ is the complete graph as stated by Theorem 7.6. As the social optimum for $c > \frac{1}{2} + b$ is also the complete graph (Theorem 7.4), the price of anarchy is PoA $= 1$, for $c > 1$ and $c > \frac{1}{2} + b$. □

Corollary 7.17. *For $c > 1$ and $b \leq c \leq \frac{1}{2} + b$, the price of anarchy is*

$$PoA = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + (c + b \cdot (n-1))(n-2)}.$$

Proof. For $c > 1$ and $b \leq c \leq \frac{1}{2} + b$ the only Nash equilibrium is the complete graph (Theorem 7.6) and according to Theorem 7.4, the social optimum is the star graph. Thus, the price of anarchy is given by

$$\begin{aligned}\text{PoA} =& \frac{\text{cost(complete graph)}}{\text{cost(star graph)}} = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right)(n-1) \cdot n}{(1 - 2(c-b) + (c-b) \cdot n + b \cdot (n-2) \cdot n)(n-1)} \\ =& \frac{(\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + (c + b \cdot (n-1))(n-2)}\end{aligned}$$

□

Corollary 7.18. *For $1 < c < b$, the price of anarchy is*

$$PoA = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + \left(\frac{2}{3}b + \frac{1}{3}c\right) \cdot n \cdot (n-2)}.$$

Proof. For $1 < c < b$ the only Nash equilibrium is the complete graph as stated in Theorem 7.6 and the social optimum is a path graph (Theorem 7.4). The price of anarchy is given by

$$\begin{aligned}\text{PoA} =& \frac{\text{cost(complete graph)}}{\text{cost(path graph)}} = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right)(n-1) \cdot n}{\left(1 + \left(\frac{2}{3}b + \frac{1}{3}c\right) \cdot n \cdot (n-2)\right)(n-1)} = \\ =& \frac{(\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + \left(\frac{2}{3}b + \frac{1}{3}c\right) \cdot n \cdot (n-2)}\end{aligned}$$

□

Combining the results of Corollary 7.16, 7.17 and 7.18 allows us to upper bound the price of anarchy to a constant for $c > 1$, as stated in in Corollary 7.19. This upper bound is asymptotically tight, as the price of anarchy is always greater or equal to one (hence at least constant) by definition.

Corollary 7.19. *For $c > 1$, the price of anarchy is $PoA = \mathcal{O}(1)$.*

Proof. For $c > 1$ and $c > \frac{1}{2} + b$, the price of anarchy is one and therefore it is also $\mathcal{O}(1)$.

We have that for $c > 1$ and $b \leq c \leq \frac{1}{2} + b$,

$$\text{PoA} = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + (c + b \cdot (n-1))(n-2)} = \mathcal{O}\left(\frac{b \cdot n^2}{b \cdot n^2}\right) = \mathcal{O}(1),$$

and for $1 < c < b$,

$$\text{PoA} = \frac{\left(\frac{1}{2} + (n-2) \cdot b\right) \cdot n}{1 + \left(\frac{2}{3}b + \frac{1}{3}c\right) \cdot n \cdot (n-2)} = \mathcal{O}\left(\frac{b \cdot n^2}{b \cdot n^2}\right) = \mathcal{O}(1).$$

Thus, for $c > 1$ we have $\text{PoA} = \mathcal{O}(1)$. □

For small b and c we can also upper bound the price of anarchy as follows:

Theorem 7.20. *For $c + b < \frac{1}{n^2}$, the price of anarchy is $PoA = \mathcal{O}(1)$.*

Proof. For $c + b < \frac{1}{n^2}$, all Nash equilibria are trees. Unless the distance to a player is infinite, no player in the network will have an incentive to build an edge.

As both the maximum possible change in $\text{betweenness}_u(s)$ and $\text{closeness}_u(s)$ for a node u in a connected graph is less than n^2 and all Nash equilibria are connected, $\Delta\text{cost}_u(s) > -n^2 \cdot c - n^2 \cdot b + 1$. We require $\Delta\text{cost}_u(s) \geq 0$ such that u does not benefit from initiating an additional channel. Thus, for $c + b \leq \frac{1}{n^2}$ all Nash equilibria are spanning trees.

For $c + b \leq \frac{1}{n^2}$ the social optimum is also a spanning tree, as it is either the star or path graph. It easily follows that for $c + b \leq \frac{1}{n^2}$ and all spanning trees $\text{cost}(s) = \Theta(n)$ and therefore the price of anarchy is $\mathcal{O}(1)$. □

Finally, for $c + b \geq \frac{1}{n^2}$ and $c < 1$, we show an $\mathcal{O}(n)$ upper bound for the price of anarchy.

Theorem 7.21. *For $c + b \geq \frac{1}{n^2}$ and $c < 1$, the price of anarchy is $PoA = \mathcal{O}(n)$.*

Proof. The price of anarchy is

$$\text{PoA} = \mathcal{O}\left(\frac{|E(G)| + n^3 \cdot b + (c-b) \cdot \sum_{u \in [n]} \sum_{r \in [n]-u} (d_G(u,r) - 1)}{n^3 \cdot b + n}\right).$$

We can say that $d_G(u,r) < \Theta\left(\frac{2}{\sqrt{c+b}}\right)$, as player u would connect to player r otherwise. Player u would become closer to half the nodes on the path otherwise and reduce its betweenness cost through the routing potential gained by the link addition. Therefore we have,

$$\text{PoA} = \mathcal{O}\left(\frac{|E(G)| + n^3 \cdot b + n^2 \frac{c-b}{\sqrt{b+c}}}{b \cdot n^3 + n}\right).$$

It follows that

$$\mathcal{O}\left(\frac{n^3 \cdot b}{n^3 \cdot b + n}\right) = \mathcal{O}(1), \quad \text{and} \quad \mathcal{O}\left(\frac{n^2 \frac{c-b}{\sqrt{b+c}}}{n^3 \cdot b + n}\right) = \mathcal{O}\left(\frac{c-b}{n^2 \cdot b + 1}\right) = \mathcal{O}(1),$$

as $c + b \geq \frac{1}{n^2}$ and $c < 1$. Thus, it only remains to consider $\mathcal{O}\left(\frac{|E(G)|}{b \cdot n^3 + n}\right)$.

As $|E(G)| = \mathcal{O}(n^2)$ for any Nash equilibrium, we have $\text{PoA} = \mathcal{O}(n)$. □

7.3.4 Price of Stability

The price of stability (PoS), a close notion to price of anarchy, is defined as the ratio between the social optimum and the best Nash equilibrium,

$$\text{PoS} = \frac{\min_{s \in N} \text{cost}(s)}{\min_{s \in S} \text{cost}(s)},$$

where S is the set of all strategies and N is the set of strategies that are Nash equilibria. The price of stability expresses the loss in network performance in stable systems in comparison to those designed by a central performance. Corollary 7.22 gives insight into the price of stability in regions of the parameter space previously discussed in the context of the price of anarchy.

Corollary 7.22. *For $c > 1$ and $b + c < \frac{1}{n^2}$, the price of stability $PoS = \mathcal{O}(1)$.*

Proof. As the price of stability is smaller than or equal to the price of anarchy, we can follow from Corollary 7.19, that the price of stability is $\mathcal{O}(1)$ for $c > 1$. Additionally, Theorem 7.20 indicates that $\text{PoS} = \mathcal{O}(1)$ for $b + c < \frac{1}{n^2}$. □

However, we expect blockchain PCNs to fall into the remaining area, where $c+b \geq \frac{1}{n^2}$ and $c < 1$. In particular, considering the underlying uniform transaction scenario and the fixed blockchain fee equal to one (wlog), a competitive transaction fee would be $\frac{1}{n}$. Thus, an appropriate allocation for the weights is $b = \frac{1}{2n}$ and $c = \frac{1}{n}$, as the betweenness term counts each sender and receiver pair twice. For these weights the star is the social optimum (Theorem 7.4), as well as a Nash equilibrium (Theorem 7.14). Hence, the price of stability for PCNs is one; indicating that an optimal PCN is stable in a game with selfish players. Thus, PCNs can be stable and efficient.

7.4 Conclusion, Limitations, and Extensions

Summary. In this chapter, we introduced a game-theoretic model to encapsulate the creation of PCNs. To this end, we generalized previous work, as our model is more complex and demands a combination of betweenness and closeness centralities that have thus far only been studied independently in network creation games.

First, we identified the social optimum for the entire parameter space of our game. Depending on the weights placed on the betweenness and closeness centralities either the complete graph, the star graph or the path graph is the social optimum. In the area of the parameter space that most accurately reflects PCNs, we found the star graph to be the social optimum.

Next, we examined the space of possible Nash equilibria. After establishing that finding the best response of a player is NP-hard, we analyzed prominent graphs and determined if and when they constitute a Nash equilibrium. We showed that the complete graph is the only Nash equilibrium if players place a large weight on their closeness centrality; reflecting payment channels in which players execute many transactions or value privacy highly. On the other hand, both the path and circle graph are Nash equilibria only for small number of players and thus are not expected to emerge as stable structures in PCNs. On the contrary, the star graph emerges as a Nash equilibrium for the areas of our parameter space most accurately representing PCNs. In addition, we observed that depending on the size of the subsets, the complete bipartite graph is also a Nash equilibrium in similar regions of the parameter space as the star graph.

Last, combining our results, we bounded the price of anarchy for a large part of the parameter space. In particular, we proved that when the closeness centrality weight is high, meaning that the players execute transactions frequently or demand privacy, the price of anarchy is constant; indicating little loss in network performance for selfish players. On the other hand, for small weight on the closeness centrality, we showed an $\mathcal{O}(n)$ upper bound on the price of anarchy. Nevertheless, the price of stability in PCNs is equal to one, since the star is both the social optimum and a Nash equilibrium for suitable parameters; demonstrating that blockchain PCNs can indeed be both stable and efficient, when forming more centralized network structures.

Improving upper bounds. While we upper bound the price of anarchy for the entire parameter space, the upper bound we find for the price of anarchy when $b + c \geq \frac{1}{n^2}$ and $c < 1$ is not necessarily tight. Thus, it would be interesting to see whether this bound can be improved.

Bilateral channels and capital assignment. In our model, the channel is set up unilaterally by a single node. The game could be adapted to a fractional

model, allowing two parties of a channel to divide the cost between them, as is the case in most PCN. In close relation, our model fixes the capital placed in the channels. Allowing nodes to decide on the amount of capital deposited in the channels would be another possible extension.

Non-uniform weights. Additionally, to more accurately describe the objectives of players in a PCN, weights could be added to the model. For one, weights associated with players in the closeness term would represent a player's inclination to connect with others. Players with whom the player is likely to exchange transactions would be associated with higher weights in the closeness term than those players with whom the player will hardly ever exchange transactions. For betweenness centrality, weights would be related to pairs of nodes, expressing the likeliness of transactions being exchanged between them.

Part VI

Related Work and Concluding Remarks

Chapter 8

Related Work

In this chapter, we summarize the work that is related to the protocols and theoretical analysis described in this thesis. We cover various topics on scaling blockchains under the lens of theory. In particular, we discuss formal models for (sharding) distributed ledgers, payment channel constructions, algorithmic and game-theoretic analysis of payment channel networks.

8.1 Formalization of Sharding

The Bitcoin backbone protocol [13] was the first to formally define and prove a blockchain protocol, more specifically Bitcoin in a PoW setting. Later, Pass et al. [14], and recently Lewis-Pye [15], showed that there is no PoW protocol that can be robust under asynchrony. With Ouroboros [36] Kiayias et al. extended the ideas of backbone to the Proof-of-Stake (PoS) setting, where they showed that it is possible to have a robust transaction ledger in a semi-synchronous environment as well [51]. Similarly, our work extends the backbone framework to define a robust sharded ledger. We also analyze state-of-the-art sharding protocols under our framework.

Recently, a few systemization of knowledge papers on sharding [136] as well as consensus [137] and cross-shard communication [138] which have also discussed part of sharding, have emerged. Unlike these, our focus is not a complete list of existing protocols. Instead, we formally define what a secure and efficient sharded ledger is, and evaluate existing sharding protocols under this lens, by either proving or disproving their claims in our framework. Our goal is to provide a powerful tool for evaluating the security and performance of sharding protocols under "common grounds".

8.2 Payment Channel Constructions

Payment channels were first introduced by Spilman [10] as unidirectional channels and were later established as bidirectional channels [11, 12]. Currently, there exist a variety of bidirectional payment channels constructions, some applicable only on platforms that allow for arbitrary smart contracts such as Ethereum [65, 66, 70], and some applicable also on blockchain systems with limited scripting languages like Bitcoin [11, 12, 61–64, 72]. In this thesis, we present two different payment channel constructions, Cerberus channels [109] that are Bitcoin-compatible, and Brick [107] that is only applicable on platforms that enable smart contracts.

To guarantee safety, traditional payment channels require participants to be frequently online, actively watching the blockchain. To alleviate this necessity, recent proposals introduced third-parties in the channel to act as proxies for the participants of the channel in case of fraud. This idea was initially discussed by Dryja [71], who introduced the Monitors or Watchtowers [76] operating on the most notable payment network, the Bitcoin Lightning network [12]. Later, Avarikioti et al. proposed DCWC [72], a less centralized distributed protocol for the watchtower service, where every full node can act as a watchtower for multiple channels depending on the network topology. In both these works, the watchtowers are paid upon fraud. Hence, the solutions are not incentive-compatible since in case a watchtower is employed no rational party will commit fraud on the channel and thus the watchtowers will never be paid. This means there will be no third parties offering such a service unless we assume they are altruistic.

In parallel, McCorry et al. [73] proposed Pisa, a protocol that enables the delegation of Sprites [65] channels' safety to third-parties called Custodians. Although Pisa proposes both rewards and penalties similarly to Cerberus and Brick, it fails to secure the channels against bribing attacks. Particularly, the watchtower's collateral can be double-spent since it is not tied to the channel/party that employed the watchtower. In contrast to all previous payment channels constructions, both Cerberus and Brick are incentive-compatible, i. e., both provide the necessary incentives mechanisms for watchtowers (rewards and punishment) to guarantee their correct operation.

Although Cerberus channels are incentive-compatible, similarly to Watchtowers, DCWC, and Pisa, they require timelocks to guarantee safety and therefore make strong synchrony assumptions that sometimes fail in practice. Specifically, all these solutions demand a synchronous network and a perfect blockchain, meaning that transactions must not be censored, to guarantee the safety of channels. Brick, on the other hand, guarantees the correct operation of payment channels in full asynchrony, meaning that the channel parities can go securely offline for an arbitrary amount of time.

In a work similar to Brick, Lind et al. proposed Teechain [139], a layer 2 payment network that operates securely under asynchrony using hardware trusted

execution environments (TEEs) to prevent parties from misbehaving. In contrast, Brick eliminates the need for TEEs with the appropriate incentive mechanisms.

In a different line of work, Khabbazian et al. [140] proposed a lightweight watchtower design in which watchtowers do not need to store the signed revocation transactions, but instead can extract them directly from the commitment transactions that appear on the blockchain. This optimization is independent and complementary to our work, and can also be conceptually applied to Cerberus channels to improve the storage requirements for the watchtower service.

To summarize, we exhibit the differences of Brick and Cerberus channels to the other channel constructions and watchtower solutions in Table 8.1. We highlight that Brick and Cerberus channels are the only payment channel constructions that are secure under rational participants. Moreover, we observe that Brick is the only solution that maintains security under an asynchronous network and offline channel parties while assuming rational watchtowers. Further, Brick is secure even when the blockchain substrate is censored, and also when the network is congested. Finally, an extension to Brick that we describe in Section 3.3 enables protection against small scale persistence attacks making it more secure than the underlying blockchain.

Table 8.1: Comparison with previous work.

Protocol Safe under	Monitors [71]	DCWC [72]	Pisa [73]	Cerberus [109]	Brick [107]
Rational Players	✗	✗	∼[1]	✓	✓
Offline Parties	✓	✓	$T \gg t_d$[2]	$T \gg t_d$[2]	✓
Asynchrony	✗	✗	✗	✗	✓
Censorship	✗	✗	✗	✗	✓
Congestion	✗	✗	✗	✗	✓
Forks	✗	✗	✗	✗	✓[3]
Privacy	✓	✓	✓	✗	✓
Bitcoin Compat.	✓	✓	✗	✓	✗

State Channels, Payment Networks, and Sidechains. Payment channels can only support payments between users. To extend this solution to handle smart contracts [141] that allow arbitrary operations and computations, *state channels* were introduced [65]. Recently, multiple state channel constructions have emerged [65–67]. Sprites [65] improve on the worst-case delay for releasing

[2]The watchtower needs to lock collateral per-channel, equal to the channel's value. Current implementation of Pisa does not provide this.

[3]The party needs to be able to deliver messages and punish the watchtower within a large synchrony bound T.

[3]Possible if consensus is run for closing the channel as described in Section 3.3.

the collateral of the intermediate nodes on the payment network for multi-hop payments. Perun [66, 142] proposes a virtual payment hub, where every party can connect to and hence establish a "virtual channel" with any other party connected to the hub. Similarly, Counterfactual [67] presents generalized state channels (not application-specific) with the same functionality as "virtual channels".

However, all these constructions use the same foundations, i. e., the same concept on the operation of two-party channels. And as the fundamental channel solutions are flawed the whole construction inherits the same problems (synchrony and availability assumptions). Brick's design could potentially extend to an asynchronous state channel solution if there existed a valuation function for the states of the contract (i. e., a mapping of each state to a monetary value for the parties) to correctly align incentives[4]. In this case, the channel can evolve as long as the parties update the state, while in case of an uncooperative counterparty the honest party can always pessimistically close the channel at the last agreed state and continue the execution on-chain.

Another solution for scaling blockchains is sidechains [77–79]. In this solution, the workload of the main chain is transferred in other chains, the sidechains, which are "pegged" to the main chain. Although the solution is kindred to channels, it differs significantly in one aspect: in channels, the states updates are totally ordered and unanimously agreed by the parties thus a consensus process is not necessary. On the contrary, sidechains must operate a consensus process to agree on the validity of a state update. Brick lies in the intersection of the two concepts; the states are totally ordered and agreed by the parties, whereas wardens merely remember that agreement was reached at the last state announced.

8.3 Algorithmic Design of PCNs

An extension to payment channels is payment channel networks (PCNs). The core idea of PCNs is that users that do not have a direct channel can route payments using the channels of other users. The most famous and active payment network is Lightning [12] currently operating more than 35,000 channels by over 15,000 nodes with a sum of more than 1,000₿ [143].

Tikhomirov et al. [144] present a quantitative analysis of the Bitcoin Lightning network, discussing security vulnerabilities, privacy issues, and scalability limitations. In general, payment channel networks have been extensively studied, especially from a privacy-preserving point of view [145–149]. However, all these works are orthogonal to ours since we focus on the algorithmic and game-theoretic study of PCNs.

[4]Note that the same holds for Cerberus channels. However, the main challenge in Cerberus's design is the Bitcoin-compatibility as Pisa can be modified to be incentive-compatible assuming smart contracts and a perfect blockchain.

Another popular research direction concerns the design of routing algorithms for the implemented decentralized PCNs, such as the Lightning and Raiden [128]. Prihodko et al. [150] present Flare, an efficient routing algorithm for the Lightning network by collecting information on the network's local topology. Malavolta et al. [151] introduce the IOU credit network SilentWhispers that employees landmark routing to discover multiple paths and multi-party computation to decide how much capital to lock on each path. Roos et al. [152] propose SpeedyMurmurs, a routing algorithm for payment networks that uses embedding-based path discovery to find routes from sender to receiver. All these protocols assume a network structure created by the individuals participating in the network. The goal is to discover the network topology and possible routes from sender to receiver of every transaction. Our objective, in the one hand, is to design the optimal network structure and fee allocation assuming a central authority, the PSP (Chapter 6). On the other hand, we study the PCN creation from a game-theoretic perspective, assuming non-depleting channels (Chapter 7). Hence, works on PCNs' routing algorithms are orthogonal to the results presented in this thesis.

In a recent work, Sali and Zohar study the optimization of maintenance costs of payment networks from an algorithmic and game-theoretic perspective [153]. Their analysis and results are similar to ours but in a slightly different model; our work assumes a transaction-series model while their work a network traffic rates model. Another similar and complementary work that recently emerged explores the optimal network topology for virtual payment channels [154]. In this work, Aumayr et al. present a model and optimization framework which allows to compute where to optimally establish virtual payment channels.

An active line of research on payment channels is the construction of secure and private systems that can act as payment hubs. Heilman et al. propose Tumblebit [69], a Bitcoin-compatible construction of a payment hub for fast and anonymous off-chain transactions through an untrusted intermediary. Green and Miers present Bolt [70] (Blind Off-chain Lightweight Transactions) for constructing privacy-preserving unlinkable and fast payment channels. Dziembowski et al. proposed Perun [66], a virtual channel hub that allows the users connected to the hub to directly interact off-chain via virtual channels establish through the virtual hub. Finally, Khalil et al. introduced Nocust [68] whereas Poon and Buterin introduced Plasma [80], which are layer-2 commit-chains, i. e., off-chain hubs.

Although all these works also assume a central coordinator that enables the off-chain payments through a centralized network, they do not analyze how expensive the construction of a payment hub is for a PSP. In this thesis, we answer the following questions: Is a payment hub a good solution for a PSP? How much capital is required to build a payment hub compared to a capital-optimal network? These answers are highly relevant to the economic viability of payment hubs as a practical solution for payment networks, and ultimately whether payment networks can solve the eminent throughput problem of cryptocurrencies.

Our work can be seen as a cryptocurrency variant of traditional network design. It is as such somewhat related to fundamental work starting in the 1970s. For example, Johnson et al. [155] prove that given a weighted undirected graph, finding a subgraph that connects all the original vertices and minimizes the sum of the shortest path weights between all vertex pairs, subject to a budget constraint on the sum of its edge weights is NP-hard. Another similar problem is the optimum communication spanning tree problem [126], where given a set of nodes, the distances and requests between them, the goal is to find the spanning tree that minimizes the cost of communication (for each pair, the request multiplied by the sum of distance). Our channel design problem is similar to these problems since the routing of a transaction matters, and our objective is to minimize the capital on the channels (like the original network design work wants to minimize the sum of the distances). However, in contrast with traditional network design, in PCNs the order of transactions matters, as the capital moves from one side of each channel to the other. Moving capital gives network design a surprising twist, as classic techniques do not work anymore. With the anticipated importance of payment networks, we believe one should have a fresh look at network design.

Furthermore, the analysis of PCNs in the online setting contributes to the research problems on online and randomized algorithms, while we also adopt techniques and insights from well-studied problems similar to the ones explored in this thesis. For instance, the analysis of the advice complexity on the online knapsack problem led to conclusions about randomized online algorithms. Lower bounds were shown for the number of advice bits for deterministic algorithms such that the competitive ratio is improved [156]. Then, the performance of randomized algorithms was derived from this analysis. Moreover, the relationship between deterministic algorithms with advice and randomized algorithms was studied in [157] for various well-known problems. Another work discussed different problems such as load-balancing and the k-server problem and introduces a way to analyse hard problems [123]. Resource augmentation, on the other hand, was introduced in [120] and was used to examine the online knapsack problem [156,158], bin packing [159], and machine scheduling [160].

In this thesis, we introduce a new problem (and its variations) to the algorithmic community, inspired by the very practical problem of scaling blockchain protocols. Our hope is to investigate the interesting aspects of this problem from an algorithmic perspective and steer the algorithmic community towards a new direction that will benefit and advance blockchain research. This thesis aims to act as the first stepping stone towards this direction.

8.4 PCN Creation Games

In this thesis, we further studied the decentralized payment network design, where the network is created by multiple participants – not a single authority. This

model reflects more accurately the currently operating payment networks, which are indeed created by thousands of users rather than a single company, following the decentralized philosophy of cryptocurrencies like Bitcoin.

As already mentioned, various payment channel protocols have been proposed in literature [10–12,65,70,107,109,139], all presenting different solutions on how to create payment channels. We focus on modeling the PCN creation as a game with rational participants, thus our results are independent of the channel construction specifications and apply to all such solutions.

Network creation games were originally introduced by Fabrikant et al. [129]. In their game, referred to as sum network creation game, a player unilaterally creates links to minimize the sum of distances to other players in the network (closeness centrality). Later, Albers et al. [161] improved the upper bound for the price of anarchy and also examined a weighted network creation game. While these works solely focus on a player's closeness centrality, our model is more complex and additionally includes another metric, the players' betweenness centrality that represents the importance of a player in the network.

In parallel, network creation games were expanded to various settings. The idea of bilateral link creation was introduced by Corbo and Parkes [162]. Demaine et al. [163] devised the max game, where players try to minimize their maximum distance to any other player in the game. Intrinsic properties of peer-to-peer networks were considered in the network creation variation conceived by Moscibroda et al. [164,165]. The idea of bounded budget network creation games was proposed by Ehsani et al. [166]. In bounded budget network creation games, players have a fixed budget to establish links. Nodes strive to minimize their stretch, the ratio between the distance of two nodes in a graph, and their direct distance. Moreover, Àlvarez et al. [167] introduced the celebrity game, where players try to keep influential nodes within a fixed distance. However, the objectives in all these games give little insight to the control a player has over a network. This control is desired by players in blockchain payment networks to maximize the fees received for routing transactions, in essence their betweenness centrality.

A bounded budget betweenness centrality game was introduced by Bei et al. [168]. Given a budget to create links, players attempt to maximize their betweenness centrality. Due to their complexity, betweenness network creation games yield limited theoretical results, in comparison to those of the sum network creation game, for instance. In contrast to our work, a players closeness centrality is not taken into account in [168]. Thus, this model is insufficient for our purpose since it does not consider how strategically connected is a player that wants to route many transactions through the payment network.

Buechel and Buskens [169] compare betweenness and closeness centralities; however, not in a network creation game setting, as their notion of stability does not lead to Nash equilibria. We, on the other hand, study the combination of betweenness and closeness incentives in a network creation game setting.

Chapter 9

Conclusion and Future Work

We conclude this dissertation by discussing the outcomes of this thesis and its implications, as well as by presenting potential avenues for future research.

9.1 Summary and Insights

This thesis addresses one of the most eminent problems that blockchain systems face: *scalability*. To understand the magnitude of this issue, Bitcoin, the foremost cryptocurrency today, can handle at most seven transactions per second [5], while current digital monetary systems, such as Visa, handle tens of thousands. Therefore, Bitcoin, and other similar blockchain systems, fail to meet the practical requirements for widespread adaptation in every day life.

The limited transaction throughput of blockchain systems stems from the consensus layer, as every participant must download, store, and disseminate in the network all the transactions. Hence, the system cannot scale. Numerous solutions have been proposed to overcome the scalability issue, either in the consensus layer (layer 1) or off-chain (layer 2). In this thesis, we explore both avenues.

Sharding. Firstly, we investigate prominent sharding systems to gain insight on their key components and identify their security and performance weaknesses. Consequently, we provide a general framework for the security and performance analysis of sharding protocols, and introduce a roadmap to secure and efficient sharding. We believe our framework encapsulates all the critical parts, the essence of what sharding is and achieves, and thus constitutes a powerful tool towards the design, analysis, and comparison of existing and future sharding systems.

From our investigation, we highlight that the main point of failure of sharding systems is the *epoch change*; specifically, the bootstrapping of nodes on the new shard, and the distributed randomness generation protocol. To be able to withstand semi-adaptive adversaries, nodes must be periodically shuffled across shards. When assigned to a new shard, nodes must download the new chain, to be able to verify transactions. As time grows, this bootstrapping process becomes

the performance bottleneck. Verifiable compaction of state must therefore be employed to alleviate this burden. To guarantee progress the compacted state must be broadcast in the network before the epoch change. Furthermore, during each epoch change, a randomness generation protocol is executed among all nodes. These protocols incur high communication complexity to satisfy strong security properties, but as they are executed one-off for each epoch their cost can be amortized within the epoch. Overall, the epoch change is crucial to ensure both security and good performance in sharding systems.

Payment channels. The main body of this thesis focuses on off-chain solutions, the so-called payment channel networks, as they allow blockchains to scale arbitrarily and are therefore more promising. Initially, we identify two major shortcomings of payment channels: (a) First, all payment channel parties must frequently be online, monitoring the blockchain, to ensure that no party can steal funds from the channel (safety). This work can be delegated to hired third-parties, the watchtowers, but all previous watchtower constructions are not *incentive-compatible* and thus insecure. (b) Second, the safety of payment channels heavily depends on the *synchrony assumptions* of the blockchain substrate. If an adversary can successfully attack the liveness property of the blockchain, e. g., censor the revocation transaction of the payment channel, they will successfully compromise the safety of the channel and steal funds. In this thesis, we provide solutions for both problems.

To address the incentive-compatibility of watchtowers, we introduce Cerberus channels, a modification of Bitcoin Lightning channels that provide the correct incentive mechanisms (rewards and punishments) to ensure safety. This is the first solution that is both incentive-compatible and Bitcoin-compatible, disproving the conjecture that such a construction is impossible. To eliminate the synchrony assumptions, on the other hand, we present Brick, the first payment channel construction that is secure in full asynchrony under rational players (channels parties and watchtowers). Brick can withstand up to 1/3 byzantine watchtowers and ensures safety even when the parties of the channel are offline for an arbitrary amount of time, in contrast to any other channel-watchtower solution. Brick requires smart contracts, but also provides security when the underlying blockchain is compromised.

Studying the fundamental weaknesses of payment channels leads to stimulating results that gave us insights on the primitives of their construction. First, we observe that achieving consensus in a payment channel is not as hard as in a blockchain system because there can be no conflicts, i. e., the agreement on the new distribution of capital in the channel is unanimous among the parties. For this reason, *consistent broadcast* is strong enough to guarantee safety in Brick, while it fails to do so in general blockchain systems. Note that a blockchain system can employ consistent broadcast instead of consensus, but then conflicting transactions will be locked forever.

Second, a trade-off appears among different assumptions, each of which can replace the *"trust"* in the system, responsible to guarantee safety. To design a safe payment channel, one of the following must hold: either (a) watchtowers are trusted parties, or (b) we make synchrony and availability assumptions, i. e., the channel parties are online once in a while and the blockchain substrate has persistence and liveness, or (c) we rely on incentive mechanisms to motivate the correct execution of the protocol. The reason one of these conditions must hold is that the watchtowers must eventually execute the protocol faithfully, which can be enforced either by assumption (they are trusted), or by design (cases b,c). In case (b), the parties must be able to punish a misbehaving watchtower on-chain hence they must eventually come online and be able to include the punishment transaction on-chain. On the contrary, in case (c), the design of the protocol ensures the watchtower maximizes their utility when they adhere to the protocol specifications, so "trust" is replaced with monetary incentives.

Third, we observe that in both Cerberus and Brick, *unidirectional channels* are used to pay the update fees to the watchtowers. Unidirectional channels do not suffer from the problems of bidirectional channels as the capital moves only in one direction, hence safety is guaranteed by design. One could say that both Cerberus and Brick leverage the security of unidirectional channels to build secure bidirectional channels, since without the unidirectional channels both constructions would not be incentive-compatible.

Theory of PCNs. Lastly, in this thesis we investigate payment channel networks from a theoretical point of view. Our goal is to understand the design rationale of payment channel networks or PCNs, both from the perspective of a central coordinator (e. g., company, bank, etc.), and the decentralized nature of cryptocurrency participants. For this purpose, we first model the payment channel design though the lens of a payment service provider (PSP). Our aim is to design a network (open channels), impose fees, and select which transactions to be executed in this private network to optimize the PSP's gain, might that be maximizing their profit, minimizing their capital, or another objective function. This is the first work that engages this problem, with the aspiration to introduce and eventually solve interesting algorithmic problems with practical value for the blockchain community as well as for potential companies that want to build off-chain payment hubs.

We conclude this thesis with a game-theoretic analysis of payment channel networks. In particular, we model the PCNs as creation games where the network participants are rational. Our goal is to comprehend the trade-offs the network participants are facing and evaluate which network structures will be stable under specific assumptions, e. g., distribution of transactions. We believe the work presented in this thesis can shed light in the interesting game theoretic problems that arise in payment channel network design.

The theoretic analysis of payment channel networks yields some intriguing, one could even say provocative, results. For instance, both the algorithmic analysis and game-theoretic approach indicate that the *star topology* is a quite stable and almost optimal structure. In a world of decentralization, the most centralized structure possible dominates, implying that the distributed and decentralized nature of cryptocurrencies is possible only in an altruistic world.

9.2 Future Work

In this section, we discuss potential avenues for future work. We focus only on new research directions related to the results of this thesis. Possible extensions and variations of our models and results are discussed separately in the last section of each chapter (conclusion, limitations and extensions).

Privacy-preserving watchtowers. Watchtower solutions, since their conception, greatly focus on maintaining the privacy of the channel's transactions from the watchtowers. In Monitors, not only the transaction value is hidden, but in addition the watchtowers cannot associate transactions with channels unless the corresponding commitment transactions are published.

Both Cerberus and Brick are not entirely privacy preserving. In particular, Cerberus channels completely leak privacy, as the watchtower has access to the commitment transaction and thus knows the exact channel balance. Hiding the transaction values is a first step towards a privacy-preserving solution for Cerberus. We believe this should be possible using more evolved cryptographic schemes (e. g. adaptor signatures) and/or dust outputs. Although Brick does not suffer from this problem, the fact that a channel is updated is leaked to the watchtowers. Whether this problem can be tackled remains an open question.

Asynchronous multi-hop payments. Brick presents a novel channel construction that is safe under asynchrony. Nevertheless, ideally transactions in a payment network can be executed through paths of multiple hops. Currently, this is implemented with Hashed Timelock Contracts or HTLCs, which by definition employ timelocks. Thereby, the security of multi-hop payments depends on synchrony assumptions. Enabling asynchronous multi-hop payments remains an open question.

Privacy implications of Nash equilibria on PCNs. In this thesis we investigated several graph structures as possible Nash equilibria for our PCN creation game. Part of our model though is that the transaction distribution is uniform, meaning that all pairs of nodes in the network want to execute the same number of transactions. The following question naturally arises: What are possible Nash

equilibria graphs for other transaction distributions? We find such a question of the utmost importance. Not only because it can further our understanding of how a payment network will "look like", but mainly because it implies major privacy issues for off-chain networks. Examining the potential graphs as Nash equilibria and mapping them to the distribution of transactions that could yield such topologies, implies that the transparent nature of payment networks can potentially leak the privacy of the off-chain transactions. For this reason, we find intriguing the investigation of the reverse problem: Given a topology, which transaction distributions can yield such a topology as a Nash equilibrium?

Channel creation games with fees. In the game-theoretic analysis we conducted in Chapter 7, we employed the closeness centrality to express the gain of nodes of the PCN that act as proxies for multi-hop transactions. This implies that the transaction fee is a constant fixed value regardless of the transaction value. In our model, such a simplification is reasonable as we also assumed unlimited temporary capital. However, in practise this assumption is unrealistic as capital restriction apply on each channel of the network. Thereby, the transaction fees are chosen by the nodes that operate the channel and vary, typically proportionally to the transaction value.

Allowing the nodes to freely choose the transaction fee, changes the game altogether and requires a new model to capture this change. A first step towards this direction was made by Avarikioti et al. [170], but a thorough analysis has not been conducted and many questions remain open. It is worth mentioning that even in this case the star topology is shown to be a stable structure (Nash equilibrium), in contrast to other structures such as the path, the circle, and the bipartite graph. Nevertheless, the space of possible Nash equilibria is far from exhausted and the analysis of [170] also assumes uniform transactions similarly to the work presented in this thesis.

Formalizing scalability. Scalability, decentralization, and security are the cornerstones of blockchains. These three dimensions form a trilemma, as optimizing one seems to compromise another in all existing solutions. The biggest challenge we face in blockchain research is to design a system that balances all three properties. But blockchain research lacks fundamental definitions and foundational work. As a result, we are not aware of the limitations and trade-offs between the three properties. In a seminal work, Garay et al. [13] set the foundations of what a blockchain is, while recently Lewis-Pye and Roughgarden [15] present a general framework for the security analysis of blockchain protocols. Both approaches discuss the two dimensions of decentralization and security, but completely ignore the scalability dimension. Extending the framework of [15] to incorporate the scaling property of blockchains and extensively study the trade-offs is an open research direction with potentially great impact.

Formalizing blockchain scaling solutions. Various blockchain system are proposed almost in a daily basis solving problems that are not well-defined. In this work, we tried to set the foundations of sharding. But similar works are due concerning various aspects of blockchain protocols. Understanding the fundamentals and conducting axiomatic research on all the blockchain layers, e.g., network, consensus, sharding, layer 2, is of utmost importance. An interesting research problem towards this direction is the formalization of layer 2 constructions, which would significantly improve our understanding, design, and security analysis of off-chain protocols. Thyfronitis-Litos et al. [171] present the first axiomatic analysis for the Bitcoin Lightning network, but an abstraction of this work for general channels is still missing.

Another similar question that arises when investigating the scaling solutions for blockchain protocols is what are the exact differences between the various solutions in a primitive manner. In particular, both sharding protocols and sidechains employ many chains that operate in parallel. However, sharding is an on-chain solution that guarantees the security of transactions by design, while sidechains are considered off-chain and the transactions' security depends on each sidechain. Similarly, channels run in parallel, but do not require consensus for transaction verification as the decision is unanimous among channel parties. As a result, the security of transactions is guaranteed although off-chain, in contrast to both sharding and sidechains. But also channels do not allow for cross-channel transactions. On the other hand, we have solutions that operate in between on-chain and off-chain such as commit-chains (e. g., Nocust [68], Plasma [80]). Most of these solutions periodically publish some information on-chain while most transactions are handled off-chain. A systematic analysis of these primitives to understand the exact definitions, differences, performance and security trade-offs, would be valuable to the blockchain community.

Rational analysis of blockchains. In this thesis, we presented a general framework for the security analysis of sharding systems in the traditional cryptographic setting, assuming participants are honest, i. e., they follow the protocol specification, or byzantine, i. e., the participants deviate arbitrarily from the honest protocol execution. However, participants of blockchain systems are rational, i. e., they deviate from the normal execution, if they can increase their utility. To the best of our knowledge, no general framework exists to describe and analyze protocols under a hybrid model of both rational and byzantine participants.

To add insult to injury, sharding systems are fairly complex and consist of several components and sub-protocols that handle different performance aspects. All these components interact with each other and the security of the overall system depends on the security of their composition. The same holds for most blockchain protocols as they all comprise by the network layer (layer 0), the consensus layer (layer 1), and often the off-chain layer (layer 2), the security of each depending on the security of the underlying layer (e.g. security of layer 1

depends on the guarantees of layer 0). Thus, providing a composable framework for the security analysis of protocols under the hybrid model is critical, and can be a powerful tool with many implications in various fields of computer science.

Bibliography

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

[2] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014.

[3] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *Zerocoin Electric Coin Company, Technical Report*, 2016.

[4] N. Van Saberhagen, "Cryptonote v 2.0," 2013.

[5] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[6] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 931–948.

[7] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[8] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX Symposium on Networked Systems Design and Implementation*, 2019, pp. 95–112.

[9] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *39th IEEE Symposium on Security and Privacy*. IEEE, 2018, pp. 583–598.

[10] J. Spilman, "Anti dos for tx replacement," https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html, accessed: 2020-11-22.

[11] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2015, pp. 3–18.

[12] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2015.

[13] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[14] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[15] A. Lewis-Pye and T. Roughgarden, "A general framework for the security analysis of blockchain protocols," *arXiv preprint: 2009.09480*, 2020.

[16] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.

[17] "BIP65," https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, accessed: 2020-12-19.

[18] "BIP68," https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki, accessed: 2020-12-19.

[19] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: The Works of Leslie Lamport*, 2019, pp. 203–226.

[20] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM, 1994, pp. 68–80.

[21] C. Cachin, "Lecture notes on principles of distributed computing," https://disco.ethz.ch/courses/ss04/distcomp/lecture/chapter9.pdf, 2004.

[22] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[23] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[24] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[25] P. Berman, J. Garay, and K. J. Perry, "Towards optimal distributed consensus (extended abstract)," in *30th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 1989.

[26] M. J. Fischer and N. A. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters*, vol. 14, no. 4, pp. 183–186, 1982.

[27] R. Wattenhofer, *The Science of the Blockchain*, 1st ed. CreateSpace Independent Publishing Platform, 2016.

[28] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Synchronous byzantine agreement with expected O(1) rounds, expected O(n^2) communication, and optimal resilience," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 320–334.

[29] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[30] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM*, vol. 32, no. 4, pp. 824–840, 1985.

[31] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, 1983, pp. 27–30.

[32] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[33] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing*. ACM, 2019, pp. 347–356.

[34] C. Stathakopoulou, T. David, and M. Vukolić, "Mir-bft: High-throughput bft for blockchains," *arXiv preprint: 1906.05552*, 2019.

[35] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation*, 2016, pp. 45–59.

[36] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[37] B. Cohen and K. Pietrzak, "The chia network blockchain," 2019.

[38] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Annual International Cryptology Conference*. Springer, 2015, pp. 585–605.

[39] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium*, 2016, pp. 279–296.

[40] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.

[41] I. Grigg, "EOS - an introduction," 2017.

[42] J. Kwon, "Tendermint: Consensus without mining," 2014.

[43] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[44] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "Sok: Diving into dag-based blockchain systems," *arXiv preprint: 2012.06128*, 2020.

[45] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[46] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: A fast and scalable cryptocurrency protocol," *IACR Cryptology ePrint Archive, Report 2016/1159*, 2016.

[47] Y. Sompolinsky and A. Zohar, "PHANTOM: A scalable blockdag protocol," *IACR Cryptology ePrint Archive, Report 2018/104*, 2018.

[48] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.

[49] J. Sliwinski and R. Wattenhofer, "ABC: Proof-of-stake without consensus," *arxiv preprint: 1909.10926*, 2020.

[50] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," in *IEEE Security and Privacy on the Blockchain*, 2017.

[51] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[52] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[53] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint: 1805.04548*, 2018.

[54] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *IEEE Symposium on Security and Privacy*, 2017, pp. 444–460.

[55] I. Cascudo and B. David, "SCRAPE: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 537–556.

[56] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "HydRand: Practical continuous distributed randomness," *IACR Cryptology ePrint Archive, Report 2018/319*, 2018.

[57] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded merkle tree: Solving data availability attacks in blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 114–134.

[58] M. Al-Bassam, A. Sonnino, and V. Buterin, "Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities," *arXiv preprint: 1809.09044*, 2018.

[59] S. Nakamoto, "Anti dos for tx replacement," https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html, accessed: 2020-11-22.

[60] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Society Open Science*, vol. 5, p. 180089, 2018.

[61] C. Decker, R. Russell, and O. Osuntokun, "eltoo: A simple layer2 protocol for bitcoin," https://blockstream.com/eltoo.pdf, 2018.

[62] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Bitcoin-compatible virtual channels," *IACR Cryptology ePrint Archive, Report 2020/554*, 2020.

[63] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, pp. 801–815.

[64] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized bitcoin-compatible channels," *IACR Cryptology ePrint Archive, Report 2020/476*, 2020.

[65] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *International Conference on Financial Cryptography and Data Security*, 2019, pp. 508–526.

[66] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *IEEE Symposium on Security and Privacy*, 2017, pp. 327–344.

[67] J. Coleman, L. Horne, and L. Xuanji, "Counterfactual: Generalized state channels," https://l4.ventures/papers/statechannels.pdf, 2018.

[68] R. Khalil, A. Gervais, and G. Felley, "NOCUST - a securely scalable commit-chain," *Cryptology ePrint Archive, Report 2018/642*, 2018.

[69] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and Distributed System Security Symposium*, 2017.

[70] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 473–489.

[71] T. Dryja, "Unlinkable outsourced channel monitoring," https://youtu.be/Gzg_u9gHc5Q, 2016.

[72] G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer, "Towards secure and efficient payment channels," *arXiv preprint: 1811.12740*, 2018.

[73] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 2019, pp. 16–30.

[74] A. Miller, "Feather-forks: enforcing a blacklist with sub-50% hash power," https://bitcointalk.org/index.php?topic=312668.0, accessed: 2020-11-22.

[75] F. Winzer, B. Herd, and S. Faust, "Temporary censorship attacks in the presence of rational miners," in *IEEE European Symposium on Security and Privacy Workshops*. IEEE, 2019, pp. 357–366.

[76] "Bitcoin Lightning Fraud? Laolu Is Building a 'Watchtower' to Fight It," https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud, 2018.

[77] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," https://www.blockstream.com/sidechains.pdf, 2014.

[78] A. Kiayias and D. Zindros, "Proof-of-Work Sidechains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 21–34.

[79] P. Gaži, A. Kiayias, and D. Zindros, "Proof-of-Stake Sidechains," in *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 139–156.

[80] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," http://plasma.io/plasma.pdf, 2017.

[81] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Annual International Cryptology Conference*. Springer, 2019, pp. 561–586.

[82] E. Ben-Sasson, "A cambrian explosion of crypto proofs," https://nakamoto.com/cambrian-explosion-of-crypto-proofs/, 2020.

[83] I. Meckler and E. Shapiro, "Coda: Decentralized cryptocurrency at scale," https://cdn.codaprotocol.com/static/coda-whitepaper-05-10-2018-0.pdf, 2018.

[84] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 505–522.

[85] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, "Flyclient: Super-light clients for cryptocurrencies," in *IEEE Symposium on Security and Privacy*, 2020, pp. 928–946.

[86] A. Kattis and J. Bonneau, "Proof of necessary work: Succinct state verification with fairness guarantees," *IACR Cryptology ePrint Archive, Report 2020/190*, 2020.

[87] S. Kadhe, J. Chung, and K. Ramchandran, "Sef: A secure fountain architecture for slashing storage costs in blockchains," *arXiv preprint: 1906.12140*, 2019.

[88] "Bitcoin statistics on transaction utxos," https://bitcoinvisuals.com/, accessed: 2020-11-20.

[89] R. Rana, S. Kannan, D. Tse, and P. Viswanath, "Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation," *arXiv preprint: 2005.09610*, 2020.

[90] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 1987, pp. 427–438.

[91] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures," in *27th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1751–1767.

[92] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source," *IACR Cryptology ePrint Archive, Report 2015/1015*, 2015.

[93] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186.

[94] A. Kiayias and G. Panagiotakos, "On trees, chains and fast transactions in the blockchain," in *5th International Conference on Cryptology and Information Security in Latin America*, 2017, pp. 327–351.

[95] M. Borge, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Proof-of-personhood: Redemocratizing permissionless cryptocurrencies," in *IEEE European Symposium on Security and Privacy Workshops*, 2017, pp. 23–26.

[96] L. Ren, K. Nayak, I. Abraham, and S. Devadas, "Practical synchronous byzantine consensus," *arXiv preprint: 1704.02397*, 2017.

[97] E. Kokoris-Kogias, "Robust and scalable consensus for sharded distributed ledgers," *IACR Cryptology ePrint Archive, Report 2019/676*, 2019.

[98] A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis, "Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers," *arXiv preprint: 1901.11218*, 2019.

[99] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[100] S. Sen and M. J. Freedman, "Commensal cuckoo: Secure group partitioning for large-scale services," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 1, pp. 33–39, 2012.

[101] E. Androulaki, C. Cachin, A. De Caro, and E. Kokoris-Kogias, "Channels: Horizontal scaling and confidentiality on permissioned blockchains," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 111–131.

[102] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *25th Annual Network and Distributed System Security Symposium*, 2018.

[103] W. Martino, M. Quaintance, and S. Popejoy, "Chainweb: A proof-of-work parallel-chain architecture for massive throughput," https://www.kadena.io/whitepapers, 2018.

[104] M. Al-Bassam, "Lazyledger: A distributed data availability ledger with client-side smart contracts," *arXiv preprint: 1905.09274*, 2019.

[105] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the 13th EuroSys Conference*, 2018, pp. 30:1–30:15.

[106] Z. Avarikioti, L. Heimbach, R. Schmid, and R. Wattenhofer, "Fnf-bft: Exploring performance limits of bft protocols," *arXiv preprint: 2009.02235*, 2020.

[107] Z. Avarikioti, E. K. Kogias, R. Wattenhofer, and D. Zindros, "Brick: Asynchronous incentive-compatible payment channels," in *International Conference on Financial Cryptography and Data Security*, 2021.

[108] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 201–226.

[109] Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer, "Cerberus channels: Incentivizing watchtowers for bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 346–366.

[110] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun, "Prcash: Fast, private and regulated transactions for digital currencies," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 158–178.

[111] E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gaillya, P. Jovanovic, L. Gasser, and B. Ford, "Hidden in plain sight: Storing and managing secrets on a public ledger," *IACR Cryptology ePrint Archive, Report 2018/209*, 2018.

[112] J. Feigenbaum, "Multiple objectives of lawful-surveillance protocols (transcript of discussion)," in *Cambridge International Workshop on Security Protocols*. Springer, 2017, pp. 9–17.

[113] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 967–984.

[114] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Germany, Tech. Rep., 1999.

[115] D. Mazieres and D. Shasha, "Building secure file systems out of byzantine storage," in *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*. ACM, 2002, pp. 108–117.

[116] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in Bitcoin," in *19th ACM Conference on Computer and Communications Security*. ACM, 2012, pp. 906–917.

[117] G. Avarikioti, L. Käppeli, Y. Wang, and R. Wattenhofer, "Bitcoin security under temporary dishonest majority," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 466–483.

[118] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 375–392.

[119] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in Bitcoin," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 692–705.

[120] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *Journal of the ACM*, vol. 47, no. 4, pp. 617–643, 2000.

[121] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in online algorithms," in *Algorithmica*, 1990, pp. 379–386.

[122] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.

[123] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 2005.

[124] A. C.-C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *18th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 222–227.

[125] D. Komm, *An Introduction to Online Computation: Determinism, Randomization, Advice*. Springer, 2016.

[126] T. Hu, "Optimum communication spanning trees," *SIAM Journal on Computing*, vol. 3, no. 3, pp. 188–195, 1974.

[127] A. Khodaverdian, B. Weitz, J. Wu, and N. Yosef, "Steiner network problems on temporal graphs," *arXiv preprint: 1609.04918*, 2016.

[128] "Raiden network," https://raiden.network/, 2017, accessed: 2020-11-22.

[129] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker, "On a network creation game," in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, 2003, pp. 347–351.

[130] E. Koutsoupias and C. Papadimitriou, "Worst-case equilibria," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 1999, pp. 404–413.

[131] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden, "The price of stability for network design with fair cost allocation," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1602–1623, 2008.

[132] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1978.

[133] J. Bertrand, *Book review of Theorie Mathematique de la Richesse Social and of Recherches sur les Principes Mathematiques de la Theorie des Richesses.* Journal des Savants, 1883.

[134] S. Álvarez, "The betweenness centrality of a graph," 2007.

[135] R. C. Entringer, D. E. Jackson, and D. Snyder, "Distance in graphs," *Czechoslovak Mathematical Journal*, vol. 26, no. 2, pp. 283–296, 1976.

[136] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies.* ACM, 2019, pp. 41–61.

[137] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Sok: Consensus in the age of blockchains," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies.* ACM, 2019, pp. 183–198.

[138] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," *IACR Cryptology ePrint Archive, Report 2019/1128*, 2019.

[139] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. R. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 63–79.

[140] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Outpost: A responsive lightweight watchtower," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies.* ACM, 2019, pp. 31–40.

[141] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[142] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2019, pp. 625–656.

[143] "Real-time lightning network statistics," https://1ml.com/statistics, accessed: 2020-12-09.

[144] S. Tikhomirov, P. Moreno-Sanchez, and M. Maffei, "A quantitative analysis of security, anonymity and scalability for the lightning network," in *IEEE European Symposium on Security and Privacy Workshops*. IEEE, 2020, pp. 387–396.

[145] M. Romiti, F. Victor, P. Moreno-Sanchez, B. Haslhofer, and M. Maffei, "Cross-layer deanonymization methods in the lightning protocol," *arXiv preprint: 2007.00764*, 2020.

[146] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 455–471.

[147] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "Post-quantum adaptor signature for privacy-preserving off-chain payments," *IACR Cryptology ePrint Archive, Report 2020/1345*, 2020.

[148] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *26th Annual Network and Distributed System Security Symposium*. The Internet Society, 2019.

[149] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A^{2}l: Anonymous atomic locks for scalability and interoperability in payment channel hubs," *IACR Cryptology ePrint Archive, Report 2019/589*, 2019.

[150] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," 2016.

[151] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *24th Annual Network and Distributed System Security Symposium*, 2017.

[152] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *25th Annual Network and Distributed Systems Security Symposium*, 2018.

[153] Y. Sali and A. Zohar, "Optimizing off-chain payment networks in cryptocurrencies," *arXiv preprint: 2007.09410*, 2020.

[154] L. Aumayr, E. Ceylan, M. Maffei, P. Moreno-Sanchez, I. Salem, and S. Schmid, "Demand-aware payment channel networks," *arXiv preprint: 2011.14341*, 2020.

[155] D. S. Johnson, J. K. Lenstra, and A. H. G. Kan Rinnooy, "The complexity of the network design problem," *Networks*, vol. 8, no. 4, pp. 279–285, 1978.

[156] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith, "The online knapsack problem: Advice and randomization," *Theoretical Computer Science*, vol. 527, pp. 61–72, 2014.

[157] D. Komm, *Advice and randomization in online computation.* ETH Zurich, 2012.

[158] K. Iwama and G. Zhang, "Online knapsack with resource augmentation," *Information Processing Letters*, vol. 110, no. 22, pp. 1016–1020, 2010.

[159] J. Csirik and G. J. Woeginger, "Resource augmentation for online bounded space bin packing," *Journal of Algorithms*, vol. 44, no. 2, pp. 308–320, 2002.

[160] C. A. Phillips, C. Stein, E. Torng, and J. Wein, "Optimal time-critical scheduling via resource augmentation," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing.* ACM, 1997, pp. 140–149.

[161] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty, "On nash equilibria for a network creation game," *ACM Transactions on Economics and Computation*, vol. 2, no. 1, pp. 1–27, 2014.

[162] J. Corbo and D. Parkes, "The price of selfish behavior in bilateral network formation," in *Proceedings of the 24th Annual ACM symposium on Principles of Distributed Computing.* ACM, 2005, pp. 99–107.

[163] E. D. Demaine, M. Hajiaghayi, H. Mahini, and M. Zadimoghaddam, "The price of anarchy in network creation games," *ACM Transactions on Algorithms*, vol. 8, no. 2, pp. 1–13, 2012.

[164] T. Moscibroda, S. Schmid, and R. Wattenhofer, "On the topologies formed by selfish peers," in *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing.* ACM, 2006, pp. 133–142.

[165] T. Moscibroda, S. Schmid, and R. Wattenhofer, "Topological implications of selfish neighbor selection in unstructured peer-to-peer networks," *Algorithmica*, vol. 61, no. 2, pp. 419–446, 2011.

[166] S. Ehsani, S. S. Fadaee, M. Fazli, A. Mehrabian, S. S. Sadeghabad, M. Safari, and M. Saghafian, "A bounded budget network creation game," *ACM Transactions on Algorithms*, vol. 11, no. 4, pp. 1–25, 2015.

[167] C. Àlvarez, M. J. Blesa, A. Duch, A. Messegué, and M. Serna, "Celebrity games," *Theoretical Computer Science*, vol. 648, pp. 56–71, 2016.

[168] X. Bei, W. Chen, S.-H. Teng, J. Zhang, and J. Zhu, "Bounded budget betweenness centrality game for strategic network formations," *Theoretical Computer Science*, vol. 412, no. 52, pp. 7147–7168, 2011.

[169] B. Buechel and V. Buskens, "The dynamics of closeness and betweenness," *The Journal of Mathematical Sociology*, vol. 37, no. 3, pp. 159–191, 2013.

[170] G. Avarikioti, R. Scheuner, and R. Wattenhofer, "Payment networks as creation games," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 195–210.

[171] A. Kiayias and O. S. T. Litos, "A composable security treatment of the lightning network," in *33rd IEEE Computer Security Foundations Symposium*. IEEE, 2020, pp. 334–349.