TOWARDS HIGH-THROUGHPUT CRYPTOCURRENCY TRANSACTIONS IN PAYMENT CHANNEL NETWORKS

by

Yuhui Zhang

© **Copyright by Yuhui Zhang, 2021**

All Rights Reserved

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Computer Science).

Golden, Colorado

Date ____________________

Signed: ____________________

Yuhui Zhang

Signed: ____________________

Dr. Dejun Yang
Thesis Advisor

Golden, Colorado

Date ____________________

Signed: ____________________

Dr. Tracy Camp
Professor and Department Head
Department of Computer Science

ABSTRACT

The past decade has witnessed an explosive growth in blockchain, but the blockchain-based technologies have also raised many concerns, among which a crucial one is the scalability issue. Suffering from the large overhead of global consensus and security assurance, even the leading blockchain-based cryptocurrencies can only handle up to tens of transactions per second, which largely limits their applications in real-world scenarios.

Payment channel networks (PCNs) have been proposed to improve the blockchain scalability on the application layer, which offers the off-chain settlement of transactions with minimal involvement of expensive blockchain operations. Unfortunately, as reported in the literature, there are still many challenges that need to be overcome towards high-throughput transactions in PCNs. One challenge is the payment routing algorithm design, which directly determines the success of payment transactions. Another challenge is the node operation protocol design, which can resist the offline issue and improve the cryptocurrency utilization.

In this dissertation, we conduct a comprehensive study towards high-throughput cryptocurrency transactions in PCNs from optimization, game theoretic, and economic perspectives, which provides an advanced suite of algorithms, theories, and mechanisms. In particular, we study the factors that affect the transaction throughput in PCNs. For the payment routing algorithm design domain, we study the payment routing problem from the perspectives of transaction fee, transaction robustness, and channel balance. For the node operation protocol design domain, we motivate the user participation by improving node availability and cryptocurrency utilization. Extensive simulations demonstrate that our approaches can significantly improve the throughput of PCNs.

TABLE OF CONTENTS

ABSTRACT . . . iii

LIST OF FIGURES . . . ix

LIST OF TABLES . . . xi

LIST OF ABBREVIATIONS . . . xii

ACKNOWLEDGMENTS . . . xiii

DEDICATION . . . xiv

CHAPTER 1 INTRODUCTION . . . 1

1.1 Distributed Digital Cryptocurrencies . . . 1

1.2 Scalability Issues in Blockchain-based Cryptocurrencies . . . 2

1.3 Approaches and Challenges . . . 4

CHAPTER 2 FEE MINIMIZATION FOR ROUTING IN PAYMENT CHANNEL NETWORKS . . . 7

2.1 Introduction . . . 7

2.2 Related Work . . . 9

2.3 Background and System Overview . . . 9

2.3.1 Decentralized Ledger . . . 10

2.3.2 Payment Channel . . . 10

2.3.3 Payment Channel Network . . . 10

2.3.4 Hashed Time-Lock Contract (HTLC) . . . 11

2.3.5 Challenges . . . 11

2.4 System Model and Problem Formulation . . . 12

2.4.1 Network Model . . . 12

2.4.2 Payment Model . . . 12

2.4.3 Problem Formulation . . . 13

2.5 CheaPay: An Optimal Algorithm for Routing in PCN . . . 13

2.5.1 Design Rationale . . . 13

2.5.2 Design of CheaPay . . . 14

2.5.3 Analysis of CheaPay . . . 16

2.6 Performance Evaluation . . . 17

2.6.1 Environment Setup . . . 17

2.6.2 Performance Metrics . . . 18

2.6.3 Evaluation of CheaPay . . . 18

2.7 Conclusion . . . 19

CHAPTER 3 ROBUST PAYMENT ROUTING PROTOCOL WITH APPROXIMATION GUARANTEE IN PAYMENT CHANNEL NETWORKS . . . 20

3.1 Introduction . . . 20

3.2 Related Work . . . 23

3.3 System Model and Problem Formulation . . . 24

3.3.1 Design Goals . . . 24

3.3.2 MTFM Problem Formulation . . . 25

3.4 RobustPay$^+$: A Distributed Robust Payment Routing Protocol in PCNs . . . 26

3.4.1 Design Rationale . . . 26

3.4.2 Design Challenges . . . 27

3.4.3 Payment Path Construction . . . 27

3.4.3.1 Design Rationale . . . 27

3.4.3.2 Design of RobustPay$^+$ . . . 29

3.4.3.3 Analysis of RobustPay$^+$ . . . 32

3.4.4 HTLC Establishment . . . 32

3.4.5 Payment Forwarding . . . 34

3.5 Performance Evaluation . . . 35

3.5.1 Environment Setup . . . 35

3.5.2 Performance Metrics . . . 37

3.5.3 Evaluation of RobustPay$^+$ . . . 37

3.6 Conclusion . . . 38

CHAPTER 4 COUNTER-COLLUSION SMART CONTRACTS FOR WATCHTOWERS IN PAYMENT CHANNEL NETWORKS 40

4.1 Introduction 40

4.2 Related Work 42

4.3 Background and Game Model 43

4.3.1 Payment Channel Network (PCN) 43

4.3.2 Watchtower 44

4.3.3 Smart Contract 44

4.3.4 Games and Strategies 44

4.3.5 Monetary Variables 46

4.4 System Model and Assumptions 47

4.4.1 Cryptographic Assumptions 47

4.4.2 Blockchain Assumptions 47

4.4.3 Adversary Model 48

4.5 The Watchtower Contract 48

4.5.1 The Contract 48

4.5.2 The Game and Analysis 49

4.6 The Collusion Contract 50

4.6.1 The Contract 50

4.6.2 The Game and Analysis 51

4.7 The Betrayal Contract 52

4.7.1 Challenge 52

4.7.2 The Contract 53

4.7.3 The Sub-game and Analysis 53

4.8 The Full Game Induced by All Contracts 55

4.8.1 The Game and Analysis 55

4.8.2 Insights on Contract Design 58

4.9 Implementation 58

4.9.1 Scalability . . . 59
4.9.2 Overhead and Financial Cost . . . 59
4.10 Conclusion . . . 60
CHAPTER 5 BALANCE-AWARE THROUGHPUT MAXIMIZING ROUTING IN PAYMENT CHANNEL NETWORKS . . . 61
5.1 Introduction . . . 61
5.2 Related Work . . . 64
5.3 System Model and Problem Formulation . . . 64
5.3.1 Network Model . . . 64
5.3.2 Payment Model . . . 65
5.3.3 Design Goals . . . 65
5.3.4 Problem Formulation . . . 67
5.4 BAR: A Distributed Balance-aware Routing Protocol in PCNs . . . 68
5.4.1 Design Rationale . . . 68
5.4.2 Design Challenges . . . 68
5.4.3 Payment Flow Construction . . . 69
5.4.3.1 Design Rationale . . . 69
5.4.3.2 Design of BAR . . . 70
5.4.3.3 Analysis of BAR . . . 74
5.4.4 HTLC Establishment . . . 74
5.4.5 Payment Forwarding . . . 76
5.5 Performance Evaluation . . . 77
5.5.1 Environment Setup . . . 77
5.5.2 Performance Metrics . . . 77
5.5.3 Evaluation of BAR . . . 77
5.6 Conclusion . . . 79
CHAPTER 6 AN INCENTIVE MECHANISM FOR CRYPTO CAPITAL COMMITMENT IN PAYMENT CHANNEL NETWORKS . . . 80

6.1 Introduction . . . 80
6.2 Related Work . . . 82
6.3 System Model and Problem Formulation . . . 83
6.3.1 Background and System Overview . . . 83
6.3.2 Channel Liquidity Marketplace . . . 83
6.3.3 Incentive Mechanism Model . . . 84
6.3.4 Desired Properties . . . 85
6.4 An Incentive Mechanism for Crypto Capital Commitment in Channel Liquidity Marketplace . . 86
6.5 Overview . . . 86
6.6 Social Welfare Maximization . . . 86
6.7 Incentive Mechanism Design . . . 87
6.8 Analysis . . . 89
6.9 Performance Evaluation . . . 90
6.9.1 Environment Setup . . . 90
6.9.2 Performance Metrics . . . 90
6.9.3 Evaluation of Cumulonimbus . . . 91
6.10 Conclusion . . . 92
REFERENCES . . . 93

LIST OF FIGURES

Figure 1.1 Blockchain-based cryptocurrency 1

Figure 1.2 Comparison of transaction rate 3

Figure 1.3 Payment channel network 4

Figure 1.4 Factors that affect high-throughput transactions in payment channel networks 5

Figure 2.1 Payment channel network with a payment from A to D 8

Figure 2.2 Hashed time-lock contract (HTLC). The sender A sends a payment of 7 to the recipient D via B and C with an HTLC tolerance of 3. Assume that the transaction fee charged by each user is 0.01. Circled numbers represent the sequence of the operations. 11

Figure 2.3 Impact of number of nodes on CheaPay, Cheapest and Widest. 18

Figure 3.1 Impact of unresponsive users on the transactions 21

Figure 3.2 Modified hashed time-lock contract (HTLC). Alice sends a payment of 0.1 to the Bob via B and C with an HTLC tolerance of 3. Note that there are two possible spends from an HTLC output. If Bob can produce the preimage of H_1 within 3 days, Bob can redeem path 1. If Alice can produce the preimage of H_2 after Bob produces the preimage of H within 4 days, Alice can redeem path 4. If Alice sends cancellation before Bob can produce the preimage of H_1 within 3 days, Alice can redeem path 2. After 3 days, Alice is able to redeem path 3, if there is no response from Bob. 34

Figure 3.3 Payment Forwarding in RobustPay$^+$. The sender A sends a payment of 7 to the recipient D. Two node-disjoint payment paths have been constructed. One payment path is $A \to B \to C \to D$; another payment path is $A \to E \to F \to G \to D$. HTLCs are established on both payment paths simultaneously, from the sender A to the recipient D, sequentially. The upper (green) path is not reversible, and the lower (red) path is reversible. The recipient D provides the preimage of H to G on the lower (red) payment path and refunds C on the upper (green) payment path. 35

Figure 3.4 Impact of number of nodes on RobustPay$^+$, CheaPay, Cheapest and Widest. 37

Figure 3.5 Message complexity of RobustPay$^+$ 38

Figure 4.1 Illustration of a watchtower. The watchtower monitors the channel for the offline hiring party. When the counterparty requests to close the channel with a fraud CSP, the watchtowers responds with the latest CSP. 41

Figure 4.2 Example of game tree. Bold edges indicate the actions in the unique sequential equilibrium. 46

Figure 4.3 Game induced by the Watchtower contract 49

Figure 4.4 Game induced by the Watchtower contract and the Collusion contract 51

Figure 4.5 Sub-game induced by the Watchtower contract and the Betrayal contract 54

Figure 4.6 Full game induced by the Watchtower contract, the Collusion contract, and the Betrayal contract 56

Figure 5.1 Example showing the special feature of routing in PCNs. A sends 2Ƀ to F; B sends 4Ƀ to E. Two values between two nodes represent the fund distribution in the channel. The channel $C \leftrightarrow D$ is involved in both payments. Although the bottleneck of (C, D) is 1Ƀ, it can fulfill both payment requests. 62

Figure 5.2 Illustration for channel balance awareness. Two values between two nodes represent the fund distribution between two users before routing. The values in the brackets represent the fund distribution after routing. 66

Figure 5.3 Modified hashed time-lock contract (HTLC). Alice sends 1Ƀ to Bob and Bob sends 1Ƀ to Alice via an HTLC tolerance of 2. Note that there are 3 possible spends from an HTLC output. If Bob does not produce the preimage of H_1 within 1 day, then the HTLC terminates. They can redeem path 2. If Bob produces the preimage of H_1 within 1 day, Bob does not need to reveal it. Then, if Alice produces the preimage of H_2 within another 1 day, they reveal and swap R_i and R_2 via fair exchange and redeem path 2. If Alice dose not produce the preimage of H_2 within another 1 day, Bob reveals the preimage of H_1 to Alice and redeem path 1. 75

Figure 5.4 HTLC Establishment and Payment Forwarding in BAR. A sends a payment of 1 to B. C sends a payment of 1 to D. Circled numbers represent the sequence of the operations. . 76

Figure 5.5 Comparison between BAR and Spider 78

Figure 5.6 Message complexity of BAR 78

Figure 6.1 Illustration for inbound liquidity. In the left, two values between two nodes represent the capital distribution. Although C owns 8Ƀ + 5Ƀ + 4Ƀ = 17Ƀ, C's inbound liquidity from the counterparties (*i.e.,* A, C, and D) is 0. Thus, C cannot receive payments from any other user. In the right, C has an inbound liquidity of 10Ƀ from B by creating a payment channel. Now C can receive payments from A, B, D, and E. For example, C can receive 3Ƀ along $E \to D \to B \to C$. The values in the brackets represent the capital distribution after C receives 3Ƀ from E. 80

Figure 6.2 Impact of number of liquidity makers on Cumulonimbus and Lightning Pool. 91

Figure 6.3 Impact of number of liquidity takers on Cumulonimbus and Lightning Pool. 91

LIST OF TABLES

Table 3.1 Main notations 28

Table 3.2 An Example of Routing Table before Node Splitting 30

Table 3.3 An Example of Routing Tables After Node Splitting 30

Table 3.4 Script of the modified HTLC 33

Table 4.1 Cost of using the smart contracts on Ethereum. We have approximated the cost in USD ($) using the conversion rate of 1 ether = $156.89 and the gas price of 2 Gwei which reflects the real world costs as of April 2020. 59

Table 5.1 Main notations 70

Table 6.1 Main notations 84

LIST OF ABBREVIATIONS

Channel Liquidity Marketplace . . . . CLM

Hashed Time-lock Contract . . . . HTLC

Maximum Transaction Fee Minimization . . . . MTFM

Payment Channel Network . . . . PCN

Transaction Fee Minimization . . . . TFM

Transactions Per Second . . . . TPS

ACKNOWLEDGMENTS

First, I would like to gratefully and sincerely thank my advisor Dr. Dejun Yang for his guidance, patience, and understanding during my PhD studies. This dissertation would not have been possible without the academic freedom and encouragement he has given me during the last several years.

I have been honored to have an amazing group of coauthors and collaborators, Dr. Guoliang Xue, Dr. Jian Tang, and Dr. Ruozhou Yu. Their unselfish help, insights, and feedbacks helped me improve my knowledge in the research area. I would like to thank my committee members, Dr. Xiaoli Zhang, Dr. Dinesh Mehta, and Dr. Chuan Yue, who served conscientiously and supported me along the way. I am also grateful to my colleagues, Dr. Ming Li, Dr. Jian Lin, and Nan Jiang, for offering me advice through the entire process.

Finally and most importantly, none of this would have been possible without the support of my family.

Dedicated to my kitten, Pangpang.

CHAPTER 1
INTRODUCTION

In this chapter, we will introduce the scalability issues of the blockchain-based cryptocurrencies and explain our motivation for studying high-throughput cryptocurrency transactions in payment channel networks (PCNs). We will then discuss the current research results and the remaining challenges. Finally, we will state our objectives and summarize our contributions.

1.1 Distributed Digital Cryptocurrencies

Over the past decades, the Internet has witnessed the advent of many bottom-up, grassroots applications which solve problems in a cooperative and distributed manner. A number of community-driven, non-commercial systems have become widespread. Practically, applicable solutions have often become available soon after the idea for a certain application had first been conceived. However, digital currency is an exception: it has taken more than a quarter of a century for a fully distributed digital currency to become reality, since the early 1980s.

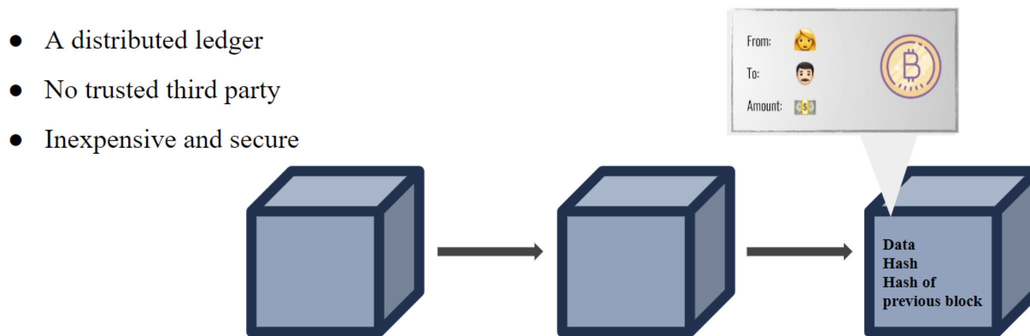


Figure 1.1 Blockchain-based cryptocurrency

The early attempts to build digital currencies, as described in [1, 2], require a central authority, *i.e.*, a bank. Approaches like b-money, RPOW, and bit gold later enables everybody to mine money independently from a bank, but still requiring a central ledger to maintain payment records. In order to completely eliminate the bank, the ledger which records the coin ownership must also be distributed. However, a fundamental and inherent risk of distributed digital currencies in general is the double spending problem. Since digital copies are trivial, one could issue two transactions in parallel, transferring the same coin to different recipients. Thus, the distribution of information and the problem of mutual agreement on a consistent state is a challenge, especially in the presence of selfish and malicious participants. It boils down to the Byzantine Generals problem [3]. This insight pushed the idea to employ quorum systems [4].

Quorum systems, as described in [5], accept the possibility of faulty information and the existence of malicious entities in a distributed environment. They introduce the concept of voting. As long as the majority of any chosen subset of peers are honest, the correct value is obtained by election. However, the approach is vulnerable to the Sybil attack [6]: a malicious entity could set up many peers which subvert the election and inject faulty information. Furthermore, it ignores the propagation delays in distributed systems and leads to temporary inconsistencies.

These difficulties were finally overcome by the Bitcoin design [7]. In November 2008, it was announced by Satoshi Nakamoto to the Cryptography mailing list. After its deployment in 2009, Bitcoin quickly became viral. Nakamoto remained active until about 2010, before handing over the project. Until now, the true identity of Nakamoto remains unknown and is subject to speculation, *e.g.*, whether the name is real or a pseudonym, or if it in fact represents a group of people. That much is certain: Bitcoin combines existing contributions from decades of research [5, 8–11]. Beyond that, it also solves the fundamental problems in a highly sophisticated, original, and practically viable way: it uses a proof-of-work scheme to limit the number of votes per entity, and thus renders decentralization practical.

Bitcoin miners collect transactions in a block and vary a nonce until one of them finds the solution (*i.e.*, a hash) to a given puzzle. The block including the solution and the transactions are broadcast to all other entities, and the distributed ledger (the blockchain) is updated as illustrated in Figure 1.1. The ownership of coins can be determined by traversing the blockchain until the most recent transaction of the respective coin is found. Forks of the blockchain due to malicious manipulations or propagation delays are resolved by considering the longest fork (including most of the work) as consensus. Thus, Sybil and double spending attacks are mitigated by binding additions to the block chain (votes) to proof-of-work contributions. The proof of work also induces a continuous supply of new coins as a reward (and incentive) for miners. Bitcoin does not require a centralized coordinating authority, and practically demonstrates the feasibility of a distributed digital currency.

1.2 Scalability Issues in Blockchain-based Cryptocurrencies

The past decade has seen a blooming of blockchain-based cryptocurrencies with over $\$800B$ of capitalization at its peak, such as Bitcoin [7], Ripple [12] and Ethereum [13]. Such cryptocurrencies leverage the blockchain technology to enable real-time gross settlements [14] across different currencies and assets significantly cheaper than the current central banking system.

However, blockchain-based cryptocurrencies cannot scale for wide-spread use. Specifically, the block propagation and storage requirements pose challenges. The main objective of the peer-to-peer network in the blockchain system is to quickly distribute the information into every part of the network. Variations in

the propagation mechanisms directly affect the formation of the distributed consensus and thus the security of the blockchain system. In general, inconsistent states, *i.e.*, blockchain forks, are undesirable, because they facilitate double spending. To guarantee the unique and synchronized global state, the blockchain broadcasts all the transactions to all participants and requires each participant to know about every single transaction, causing high overhead and local storage requirement. For example, every Bitcoin user now needs almost 20 GB of additional storage each year (*e.g.* 60 GB after three years) [15]. Currently, Bitcoin has an artificial maximum block size of 1 MB, which limits the number of transactions per block and therefore also the growth rate of the blockchain. This limit is enforced to prevent from ballooning the block chain before the protocol is prepared. Scaling to higher transaction rates will eventually consume more resources. For example, to handle a rate of $2,000$ transactions per second (tps), a block size of more than 0.5 GB and an Internet connection of approximately 1 MB/s is required.

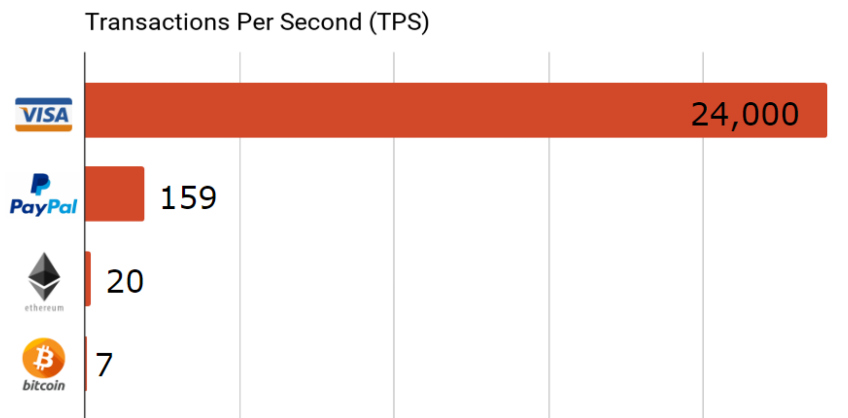


Figure 1.2 Comparison of transaction rate

In addition, in order to append a block to the blockchain to confirm the transactions in the block, a miner has to perform proof-of-work calculations (brute-forcing a number whose hash value starts with a pre-established number of zeros). Proof of work is a key component of blockchain. Inherently, any task suitable as a basis for proof-of-work schemes needs to be difficult to solve, but trivial to verify. It often boils down to a random process of trying to find a solution to a puzzle. Following the increasing computational power, the blockchain system adjusts the difficulty of the puzzle, *i.e.*, the target value, to maintain the block generation rate. For example, Bitcoin maintains a block rate of 10 minutes per block. As a result, the maximum rate of transactions the network can process is limited, for example, the Bitcoin blockchain can only process up to 7 transactions per second (tps) [16], compared to over $47,000$ peak tps handled by Visa [17]. Figure 1.2 shows the comparison of the state-of-the-art payment systems in terms of

the transaction rate.

1.3 Approaches and Challenges

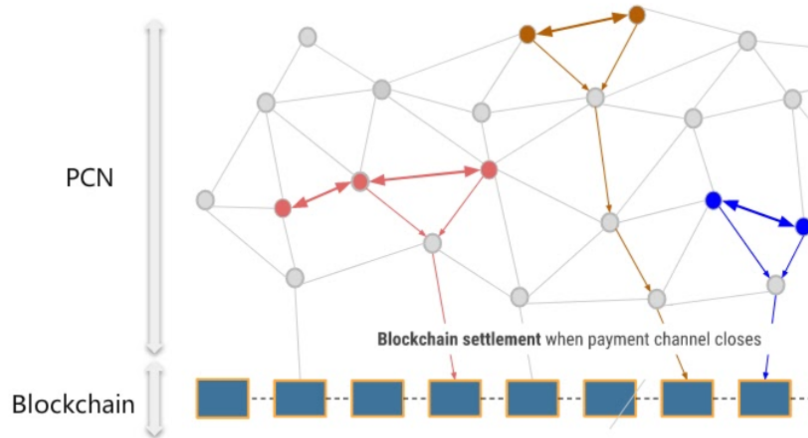


Figure 1.3 Payment channel network

To overcome this scalability issue, the payment channel network (PCN) [18], as shown in Figure 1.3, has been proposed to enable instant and inexpensive payments without requiring expensive and slow blockchain transactions, except registering the initial and final balance of each channel. Examples of such PCNs are Bitcoin's Lightning Network [18] and Ethereum's Raiden Network [19]. However, existing experience reveals that there are still many challenges that need to be overcome towards high-throughput transactions in PCNs. One challenge is the payment routings algorithm design. In PCNs, a sender can send a payment to its recipient through multiple hops of payment channels. The users in the route charge fees for transferring payments through their channels, which are expected to be significantly smaller than the blockchain transaction fees. Therefore, routing in PCNs is essential as it directly determines the success of a payment, which is the ultimate purpose of PCNs. Another challenge is the node operation protocol design. Because cryptocurrency transactions may fail due to node failure (node being offline) and edge failure (insufficient channel balances), it is desired to design protocols that can resist the offline issue and improve the cryptocurrency utilization. To summarize, the transaction throughput is affected by the following factors as shown in Figure 1.4.

- *Transaction Fee*: Since a user charges fees for forwarding payments, it is desired to optimize the transaction fee, when generating a payment path to fulfill a transaction. We first focus on the Transaction Fee Minimization (TFM) problem subject to the timeliness and feasibility constraints and design an optimal distributed routing algorithm.

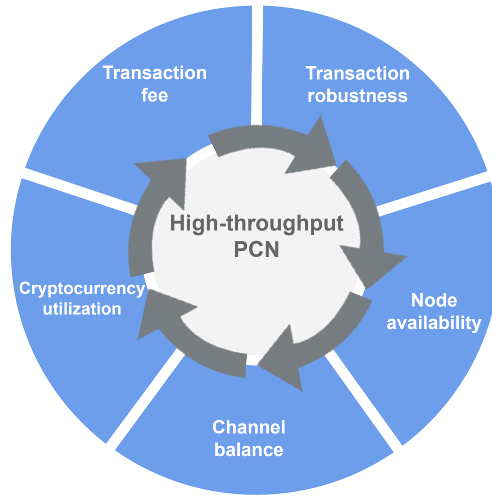


Figure 1.4 Factors that affect high-throughput transactions in payment channel networks

- *Transaction robustness*: A user may become offline intentionally or unintentionally, which makes transactions fail. Thus, we investigate the robust routing in PCNs from an optimization perspective, which is to find two alternative payment paths for a payment request, while minimizing the worst-case transaction fee. Then, we propose a robust payment protocol with security guarantee to prevent the double claim issue.

- *Node availability*: When a user gets offline, the counterparty on its channel may withdraw the channel balance by cheating. To keep channels safe and be able to go offline, users can hire the watchtower services to monitor the blockchain and prevent fraudulent channel closures. However, the watchtower could be bribed by the counterparty and collude with the counterparty. We leverage smart contracts through economic approaches to counter collusions between the watchtowers and the counterparties.

- *Channel balance*: A payment channel needs sufficient channel balance to forward transactions. However, if a user keeps forwarding payments in one direction and runs out of balance, it will lead to channel imbalance. Thus, it is desirable to mitigate the channel imbalance when designing algorithms. We study the problem of balance-aware routing in PCNs, which is to maximize the transaction throughput, while rebalancing the payment channels, subject to the conservation, timeliness, and feasibility constraints.

- *Cryptocurrency utilization*: Since users can earn fees by forwarding transactions, they prefer to create channels where their balances are frequently used for transactions. Because the price of cryptocurrencies changes dramatically, balances that are not well utilized may lead to financial loss. To improve the cryptocurrency utilization, we design an incentive mechanism for trading crypto

capital commitment, such that the users can avoid the expensive overhead of strategizing over others, and the social welfare can be maximized to motivate user participation.

The remainder of the thesis is organized as follows. In Chapter 2, we present an optimal algorithm that minimizes the transaction fee for payment routing. In Chapter 3, we provide a robust payment routing protocol with $2-$approximation guarantee. In Chapter 4, we propose smart contracts for watchtowers to guarantee node availability. In Chapter 5, we give a balance-aware routing protocol with transaction throughput maximization. In Chapter 6, we design an incentive mechanism for crypto capital commitment to improve cryptocurrency utilization.

CHAPTER 2

FEE MINIMIZATION FOR ROUTING IN PAYMENT CHANNEL NETWORKS

The past several years have witnessed an explosive growth in cryptocurrencies, but the blockchain-based cryptocurrencies have also raised many concerns, among which a crucial one is the scalability issue. Suffering from the large overhead of global consensus and security assurance, even the leading cryptocurrencies can only handle up to tens of transactions per second, which largely limits their applications in real-world scenarios. Among many proposals to improve the cryptocurrency scalability, one of the most promising and mature solutions is the payment channel network (PCN), which offers the off-chain settlement of transactions with minimal involvement of expensive blockchain operations. In this work [20], we investigate the problem of payment routing in PCNs from an optimization perspective, which is to minimize the transaction fee of a payment path, subject to the timeliness and feasibility constraints. We present an optimal distributed algorithm CheaPay for this problem. Extensive simulations demonstrate that CheaPay significantly outperforms baseline algorithms in terms of the success ratio and the average accepted value.

2.1 Introduction

The past decade has seen a blooming of blockchain-based cryptocurrencies with over $\$800B$ of capitalization at its peak, such as Bitcoin [7], Ripple [12] and Ethereum [13]. Such cryptocurrencies leverage the blockchain technology to enable real-time gross settlements [14] across different currencies and assets significantly cheaper than the current central banking system. However, blockchain-based cryptocurrencies cannot scale for wide-spread use. To guarantee the unique and synchronized global state, the blockchain broadcasts all the transactions to all participants and requires each participant to know about every single transaction, causing high overhead and local storage requirement. For example, every Bitcoin user now needs almost 20 GB of additional storage each year (e.g. $60GB$ after three years) [15]. In addition, in order to append a block to the blockchain to confirm the transactions in the block, one has to perform proof-of-work calculations (brute-forcing a number whose hash value starts with a pre-established number of zeros). As a result, the maximum rate of transactions the network can process is limited, for example, the Bitcoin blockchain can only process up to 7 transactions per second (tps) [16], compared to over $47,000$ peak tps handled by Visa [17].

To overcome this scalability issue, the payment channel network (PCN) [18], as shown in Figure 2.1, has been proposed to enable instant and inexpensive payments without requiring expensive and slow blockchain transactions, except registering the initial and final balance of each channel (details can be

found in Section 2.3). Examples of such PCNs are Bitcoin's Lightning Network [18] and Ethereum's Raiden Network [19]. In PCNs, a sender can send a payment to its recipient through multiple hops of payment channels. The users in the route charge fees for transferring payments through their channels, which are expected to be significantly smaller than the blockchain transaction fees. Therefore, routing in PCNs is essential as it directly determines the success of a payment, which is the ultimate purpose of PCNs.

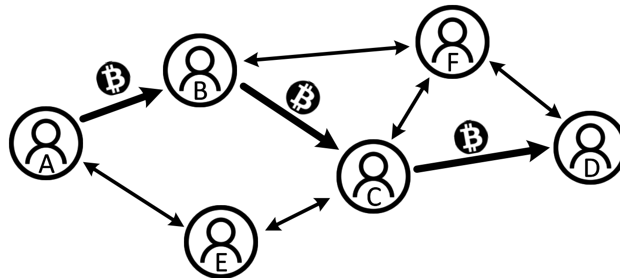


Figure 2.1 Payment channel network with a payment from A to D

There are few existing works on routing in PCNs [21–25]. Unfortunately, they only scratch the surface by either focusing on the privacy [21, 22] or neglecting some important realistic constraints, like the transaction fees [21–25]. At first glance, the PCN seems to resemble a conventional computer ad hoc network. However, there are two important differentiating features, which make routing in PCNs more complex. The first difference is the fact that the fees paid to the users for transferring the payment have to be sent along with the payment to the recipient. These fees will then be collected by the users along the route sequentially while the payment is transferred. This results in different balance requirements, called the *feasibility constraint*, on different edges. The second difference is the fact that each user in PCNs has its own tolerance on the time of participating in payment forwarding, which is determined by the number of hops to the recipient (to be elaborated later in Section 2.3). This is referred to as the *timeliness constraint*, which is not a factor in computer ad hoc networks. Due to these reasons, algorithms for conventional routing problems in computer ad hoc networks cannot be applied to PCNs.

In this work, we focus on the routing problem in PCNs from an optimization perspective. Specifically, we focus on the *Transaction Fee Minimization (TFM)* problem: minimizing the total transaction fee of a path to transfer a payment from the sender to the recipient, while guaranteeing that both the timeliness and feasibility constraints are satisfied for each involved payment channel on the path. *The main contributions of this work are*:

- To the best of our knowledge, we are the first to consider the transaction fee minimization problem for routing in PCNs with the timeliness and feasibility constraints.

- We design an optimal distributed algorithm CheaPay to minimize the total transaction fee for a payment request, while satisfying both the timeliness and feasibility constraints.
- Extensive simulations demonstrate that CheaPay not only minimizes transaction fees, but also achieves superior success ratio and average accepted payment value over baseline algorithms.

The remainder of the work is organized as follows. In Section 2.2, we give a brief review of related work in the literature. In Section 2.3, we present the background and system overview of PCNs. In Section 2.4, we formally describe the system model and give the problem formulation. In Section 2.5, we present the design of our routing algorithm CheaPay and analyze the properties of CheaPay. In Section 2.6, we evaluate the performance of CheaPay by comparing it to baseline algorithms. We conclude this work in Section 2.7.

2.2 Related Work

As of today, there are only limited efforts on studying routing problem in PCNs. In 2016, Flare [23] was proposed as one of the first decentralized routing algorithms for PCNs. In Flare, each node has a routing table consisting of the neighborhood of nodes that are nearby in hop distance and paths to a number of beacons nodes. Malavolta [21] studied the privacy-reserving routing problem and designed a routing scheme, SilentWhisper, based on Landmark Routing [26]. All paths pass a landmark potentially leading to unnecessarily long paths. In addition, this would be against the sole purpose of using blockchain systems: decentralization. Using embedding-based path discovery [27], SpeedyMurmurs [22] overcame these weaknesses and improved SilentWhisper with regard to metrics, including payment success ratio, delay, overhead, path length, and stabilization. Rohrer *et al.* [24] considered a payment as a flow and aggregated multiple paths to utilize the available capacities in the network efficiently. Along this line, Yu *et al.* [25] designed a distributed algorithm to improve the payment success ratio and reduce the system overhead. However, all these works either focus on the privacy [21, 22] or use simplistic models without the hop-dependent constraints [21–25], whereas this work develops a holistic approach to the routing problem in PCNs while considering the timeliness and feasibility constraints, from an optimization perspective. The closest work to ours was done by Engelmann *et al.* [28]. But their simplistic model assumes that the transaction fee depends only on the senders payment not the forwarded payment.

2.3 Background and System Overview

In this section, we provide the necessary background on permissionless blockchains, such as Bitcoin and Ethereum, and present an overview of our payment channel network system.

2.3.1 Decentralized Ledger

Cryptocurrencies like Bitcoin [7], Ethereum [13], and Ripple [12] are based on the blockchain technology, which is an append-only decentralized ledger of transactions shared among mutually distrusted entities. However, the consensus algorithm (e.g. proof-of-work in Bitcoin) that guarantees the unique global state requires large local storage, due to the high levels of data replication and high computational power for adding a block containing transactions to the blockchain.

Scalability Issue. The main concern of decentralized blockchains is that every peer needs to be aware of all transaction of all other peers to not be vulnerable to double-spending. Bitcoin currently only supports up to 7 transactions per second [16] which is not comparable to over $47,000$ peak tps processed by Visa [17]. Therefore, the blockchain-based cryptocurrencies cannot scale for wide-spread use.

2.3.2 Payment Channel

To overcome the scalability issue, off-chain approaches have been proposed to eliminate the need to commit each individual transaction to the blockchain. The use of payment channels is one way to realize the off-chain approach. Two users establish a payment channel by each depositing a certain amount into a joint account and adding this transaction to the blockchain.

Now a transaction between them is essentially a channel balance update agreed upon by them. A channel is protected by multi-signature smart contracts, which ensure validity, nonequivocality and non-repudiation of the on-going transactions. When one party publishes an obsolete balance history to reverse settled transactions or to double-spend, the contract guarantees that the dishonest party is punished by granting all its remaining channel balance to the other party. This economically prevents an adversary from utility gain via dishonest behaviors. When the channel closes because either it is not needed anymore or the deposit is depleted, a closing transaction will be broadcast to the blockchain and will send deposited amount to each user according to the most recent balance.

2.3.3 Payment Channel Network

Unfortunately, payment channel alone cannot solve the scalability issue. Requiring everyone to create a payment channel with everyone else results in a large amount of on-chain transactions broadcast to the blockchain. In order to enable payments between any two users, payments can be routed through multiple hops of channels in the payment channel network (PCN) formed by users connected by payment channels. This, however, can lead to issues that a user denies performing payment transfer after receiving a preceding one, or the recipient denies receiving the payment.

2.3.4 Hashed Time-Lock Contract (HTLC)

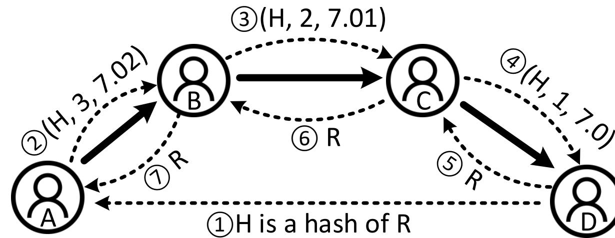


Figure 2.2 Hashed time-lock contract (HTLC). The sender A sends a payment of 7 to the recipient D via B and C with an HTLC tolerance of 3. Assume that the transaction fee charged by each user is 0.01. Circled numbers represent the sequence of the operations.

To address these issues, the Hashed Time-Lock Contract (HTLC) mechanism has been introduced [18], as shown in Figure 2.2. The recipient first generates a random value R and sends its hash H to the sender. The sender, as well as any intermediate user, includes H in the transaction contract, such that the transferred payment can be claimed by the transferee only when the secret R is provided to the transferor. In addition, each transaction is restricted by an HTLC tolerance, such that if the transferor does not receive R within the HTLC tolerance, the transferred fund will be refunded to the transferor after the HTLC expires. The unit of the HTLC tolerance, denoted by δ, is the worse-case bound on time for one on-chain transaction. Every user in the payment path sets a tolerance, which is a smaller HTLC tolerance in the outgoing payment channel than that in the incoming payment channel. For example, in Lightning Network, the tolerance is set as the number of hops until the recipient [18]. As an example, the HTLC $(H, 2, 7.01)$ from B to C in Figure 2.2 means that C can receive a payment of 7.01 from B if C can provide the preimage of H within 2δ. This mechanism ensures that a user can pull the payment from its predecessor after its payment has been pulled by its successor. Note that the HTLC tolerance time is not the time of payment routing, which is fast when users are cooperative and responsive. In addition to the payment to the recipient, an HTLC also includes the transaction fees charged by the intermediate nodes for transferring the sender's payment. The fees are significantly lower than blockchain transaction fees largely due to the time-value of locking up funds in the channel, as well as paying for the chance of channel close on the blockchain.

2.3.5 Challenges

The main challenge of the routing in PCNs is that PCNs cannot be treated as a conventional computer ad-hoc network. In fact, the complexity of routing in PCNs is increased by two distinct features, which are not found in conventional ad-hoc networks. The first distinct feature is the *feasibility constraint*, which

means different balance requirements on different channels. It is resulted from that fees paid to the users along the route have to be sent together with the payment to the targeted recipient, during the payment transferring process. The second feature is referred to as the *timeliness constraint*, which cannot be found in computer ad hoc networks. The reason is that the tolerance on the cooperating time in payment forwarding process of each PCN user is different. The tolerance is judged by the number of hops to the recipient. Due to these differences between PCNs and computer ad hoc networks, PCNs cannot apply the algorithms for conventional routing problems directly.

2.4 System Model and Problem Formulation

In this section, we describe the network model and the payment model, and give a precise problem formulation.

2.4.1 Network Model

A PCN can be represented as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. Each node $v_i \in \mathcal{V}$ represents a user, who has a cryptocurrency account and establishes at least one payment channel with a peer user. Each edge $e = (v_i, v_j) \in \mathcal{E}$ represents a payment channel, where v_i is the *transferor* and v_j is the *transferee*. Each edge is associated with several attributes. First, each edge $(v_i, v_j) \in \mathcal{E}$ has a transaction fee $f_{i,j}$, denoting the amount of value charged by v_i for transferring a payment to v_j. For notational convenience, we let $f_{i,i} = 0$. Second, each edge $(v_i, v_j) \in \mathcal{E}$ has a channel balance $b_{i,j}$, denoting the amount of the remaining balance that v_i can transfer to v_j. Third, each edge $(v_i, v_j) \in \mathcal{E}$ also has an HTLC tolerance $\tau_{i,j}$, denoting the maximum time v_i would wait for the secret R provided by v_j. Note that we omit the transmission time in PCNs, because it is negligible in comparison with the transaction time on blockchain. For simplicity, we assume the set $\mathcal{E}$ only contains edges with positive balances at any time. An edge with zero balance is temporarily removed from the graph, until its balance gets recharged by new deposits or payment transactions on the opposite direction. In addition, we define an *(i, j) path* as a simple path from v_i to v_j.

We assume that each user only has local knowledge on all its incoming and outgoing edges, including their transaction fees, balances and HTLC tolerances. In general, each user cannot know the transaction fee, balance or HTLC tolerance of any remote edge, due to network asynchrony and dynamics.

2.4.2 Payment Model

A *payment request* is denoted by $R = (v_s, v_t, a)$, where v_s and v_t are the sender and recipient respectively, and a is the amount of the payment to be transferred. A payment request R is performed via a number of transactions through different channels, organized as an (s, t) path p denoted by a sequence

$v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$, where $v_0 = v_s$ and $v_L = v_t$. Here we abuse the notation $v_i \in p$ to represent that a node v_i is involved in a payment path p. We use a transaction fee function $F_p(l, m)$ to denote the total transaction fee from v_l to v_m on a path p, where $0 \leq l < m \leq L$:

$$F_p(l, m) = \begin{cases} \sum_{i=l+1}^{m-1} f_{i,i+1}, & v_l = v_s, \\ \sum_{i=l}^{m-1} f_{i,i+1}, & v_l \neq v_s. \end{cases} \tag{2.1}$$

For a path, all the payment channels are called *involved channels* (ICs), and all the users except the sender and the recipient along the path are called *intermediate users* (IUs).

2.4.3 Problem Formulation

To formally formulate our studied problem, we introduce the following necessary concepts.

Timeliness Constraint: A path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ satisfies timeliness constraint, if the payment request can be successfully fulfilled within the HTLC tolerance of each IC, *i.e.*, $\tau_{i,i+1} \geq L - i$, $\forall i \in [0, L-1]$. Timeliness guarantees the commitment of honest processing at any IC.

Feasibility Constraint: A path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ satisfies feasibility constraint, if the payment request can be successfully transferred through each IC. Specifically, the balance $b_{i,i+1}$ of $e = (v_i, v_{i+1})$ should be at least the payment amount a plus the accumulation of transaction fees paid to the IUs that follow v_i on the path, *i.e.*, $b_{i,i+1} \geq a + F_p(i+1, L)$, $\forall i \in [0, L-1]$.

Apparently, there may exist more than one payment path that can fulfill a given payment request. Thus, it is necessary for users to minimize the transaction fee, while still guaranteeing timeliness and feasibility constraints. Towards this goal, we consider the following optimization problem in this work:

Transaction Fee Minimization (TFM): Given a payment request $R = (v_s, v_t, a)$, find a timely and feasible path p to fulfill R, such that the total transaction fee is minimized.

In this work, we aim to design an optimal algorithm for the TFM problem in payment channel networks. Our objective is to find a path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ to fulfill the payment request $R = (v_s, v_t, a)$, subject to the timeliness and feasibility constraints.

2.5 CheaPay: An Optimal Algorithm for Routing in PCN

In this section, we design and analyze CheaPay, an optimal distributed algorithm for the TFM problem in PCNs.

2.5.1 Design Rationale

Before we formally describe the design of CheaPay, we introduce the following definition.

h-(i, j) TFM path: A path is an h-(i, j) TMF path, if it satisfies the following conditions: 1) the length of this path is no more than h; 2) this path satisfies both the timeliness and feasibility constraints; 3) this

path has the minimum transaction fee among all paths from v_i to v_j.

Our algorithm CheaPay is based on the distributed Bellman-Ford Algorithm [29, 30], in which each node v_i exchanges its h-(i,j) TFM paths with its transferors and transferees. CheaPay consists of three stages: initialization, updating and routing. In the initialization stage, the algorithm builds a local transaction fee table and a local routing table for each node v_i and sends the transaction fee table to all v_i's transferors. The local tables store the minimum transaction fees from v_i to all other nodes and the corresponding outgoing transferees. In the updating stage, the algorithm updates the local transaction fee table and routing table based on the transaction fee tables received from v_i's transferees. The basic idea is that an h-(i,j) TFM path must go through one of v_i's transferees. Then, the updated transaction fee table is sent to v_i's transferors. In the routing stage, the algorithm determines the $(|\mathcal{V}|-1)$-(s,t) TFM path after executing the updating procedure on each node for $|\mathcal{V}|-2$ times, where $|\mathcal{V}|$ is the total number of nodes. The $(|\mathcal{V}|-1)$-(s,t) TFM path is the final path to the payment request. The intuition is that for any $h \geq 0$, an h-(i,j) TFM path goes through no more than $|V|$ nodes and thus goes through no more than $|V|-1$ edges. We present the detailed algorithms in the following subsection. Note that when we say "Broadcast a message", we essentially mean one node sending a message to the other through a secure communication channel, rather than actually sending a payment through the payment channel.

2.5.2 Design of CheaPay

In this section, we describe the details of CheaPay, which is illustrated in Algorithms 1, 2 and 3.

Algorithm 1: CheaPay-Init

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a transaction fee table $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$.

```
1  h ← 1; N_i ← the set of v_i's transferee nodes;
2  for v_j ∈ V do
3  |   N^h(i,j) ← null;
4  |   if v_i = v_j then Γ^h(i,j) ← 0;
5  |   else if v_j ∈ N_i then
6  |   |   Γ^h(i,j) ← f_{i,j}; N^h(i,j) ← v_j;
7  |   |   if v_i = v_s then Γ^h(i,j) ← 0;
8  |   else Γ^h(i,j) ← ∞;
9  end
10 Broadcast {Γ^h(i,j)}_{v_j ∈ V} to v_i's transferors;
11 Request {Γ^h(k,j)}_{v_j ∈ V} from v_i's transferees;
12 return {Γ^h(i,j)}_{j ∈ V}
```

The initialization stage is shown in CheaPay-Init (Algorithm 1). CheaPay-Init builds two local tables for a node v_i. The first table is a transaction fee table $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$ that stores the minimum transaction fees corresponding to the h-(i,j) TFM paths, $\forall v_j \in \mathcal{V}$. Initially, h is set to 1. We use $\mathcal{N}_i$ to denote the set

of v_i's transferees. The minimum transaction fee from v_i to itself is always 0 (Line 4); the minimum transaction fee from v_i to its transferee v_k is initially set to $f_{i,k}$ (Line 6), which is the transaction fee through the payment channel (v_i, v_k); the minimum transaction fee from v_i to all the other nodes v_j is initially set to positive infinity ∞ (Line 8), which indicates that payment cannot be transferred from v_i to v_j through no more than $h = 1$ payment channel. Note that if the transferor v_i is the sender v_s, the transaction fee from v_i to v_j through 1 payment channel is set to 0 (Line 7), because no intermediate node is required. The second table is a routing table $\{\mathcal{N}_h(i,j)\}_{v_j \in \mathcal{V}}$ that stores v_i's outgoing transferees corresponding to the minimum transaction fees stored in $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$. We use $\mathcal{N}_h(i,j)$ to keep track of the h-(i,j) TFM path. CheaPay-Init outputs a transaction fee table and a routing table for a node v_i. We shall run Algorithm 1 to initialize the tables for each node $v_i \in \mathcal{V}$.

Algorithm 2: CheaPay-Update

Input: a node v_i, a table $\{\Gamma^h(k,j)\}_{v_j \in \mathcal{V}, v_k \in \mathcal{N}_i}$.
Output: an updated table $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$.

1 Request $\{\Gamma^h(k,j)\}_{v_k \in \mathcal{N}_i, v_j \in \mathcal{V}}$ from v_i's transferees;
2 **for** $v_j \in \mathcal{V}$ **do**
3 $\quad \Gamma^{cand}(i,j) \leftarrow \min_{v_k \in \mathcal{N}_i} \Gamma^h(k,j) + f_{i,k}$;
4 $\quad N^{cand}(i,j) \leftarrow \arg\min_{v_k \in \mathcal{N}_i} \Gamma^h(k,j) + f_{i,k}$;
5 $\quad$ **if** $\Gamma^{cand}(i,j) < \Gamma^h(i,j)$ *and* $b_{i,j} \geq \Gamma^{cand}(i,j) + a$ *and* $\tau_{i,j} \geq h + 1$ **then**
6 $\quad\quad \Gamma^{h+1}(i,j) \leftarrow \Gamma^{cand}(i,j)$; $N^{h+1}(i,j) \leftarrow N^{cand}(i,j)$;
7 $\quad$ **else**
8 $\quad\quad \Gamma^{h+1}(i,j) \leftarrow \Gamma^h(i,j)$; $N^{h+1}(i,j) \leftarrow N^h(i,j)$;
9 $\quad$ **end**
10 **end**
11 $h \leftarrow h + 1$;
12 Broadcast $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$ to v_i's transferors;
13 **return** $\{\Gamma^h(i,j)\}_{v_j \in \mathcal{V}}$

The updating stage is shown in CheaPay-Update (Algorithm 2). CheaPay-Update updates the transaction fee table and the routing table for a node v_i. The basic idea is that an h-(i,j) TFM path must go through one of v_i's transferees. Thus, we can obtain $\Gamma^{h+1}(i,j)$ based on $\Gamma^h(k,j)$, where $v_k \in \mathcal{N}_i$. First, CheaPay-Update requests $\{\Gamma^h(k,j)\}_{v_k \in \mathcal{N}_i, v_j \in \mathcal{V}}$ from all v_i's transferees. Then, CheaPay-Update calculates $\Gamma^{cand}(i,j)$, which is the candidate minimum transaction fee of the $(h+1)$-(i,j) TFM path (Line 3). Also, CheaPay-Update finds $\Gamma^{cand}(i,j)$'s corresponding outgoing transferee $\mathcal{N}_{cand}(i,j)$ (Line 4). if $\Gamma^{cand}(i,j)$ is smaller than $\Gamma^h(i,j)$, it indicates that using one more payment channel can decrease the transaction fee. We update the transaction fee table and routing table, if the new path satisfies both the timeliness constraint and feasibility constraint (Lines 5-6). If either of these two constraints is not satisfied, or $\Gamma^{cand}(i,j)$ is the same as $\Gamma^h(i,j)$, then $\Gamma^{h+1}(i,j)$ remains $\Gamma^h(i,j)$ and $N^{h+1}(i,j)$ remains $N^h(i,j)$ (Lines 7 to 9). After $\{\Gamma^{h+1}(i,j)\}_{v_j \in \mathcal{V}}$ is obtained, v_i sends the table to all its transferees. We shall run Algorithm 2

to update the tables for each node $v_i \in \mathcal{V}$.

Algorithm 3: CheaPay-Routing

```
    Input: a network G = (V, E), a payment request R = (v_s, v_t, a).
    Output: a path p_R.
 1  p_R ← ∅; for v_i ∈ V do Init(G, v_i);
 2  while h ≤ |V| − 1 do
 3  |  for v_i ∈ V do Update(v_i);
 4  end
 5  if Γ^h(s, t) < ∞ then
 6  |  v_i ← v_s; v_k ← N_{|V|−1}(i, j);
 7  |  while v_i ≠ null and v_k ≠ null do
 8  |  |  p_R ← p_R ∪ {v_i}; v_i ← v_k; v_k ← N_{|V|−1}(i, j);
 9  |  end
10  end
11  return p_R
```

The routing stage is shown in CheaPay-Routing (Algorithm 3). CheaPay-Routing outputs the $(|\mathcal{V}|-1)$-(s,t) TFM path. First, CheaPay-Routing runs CheaPay-Init to initialize each node $v_i \in \mathcal{V}$. Then, CheaPay-Routing calls CheaPay-Update to update the tables on each node $v_i \in \mathcal{V}$ for $|\mathcal{V}|-1$ iterations, because the solution path requires no more than $|\mathcal{V}|-1$ payment channels. If $\Gamma^{|V|-1}(s,t)$ is infinity, then there is no valid solution path. Otherwise, CheaPay-Routing keeps track of the outgoing transferees that are stored in the routing tables (Lines 7-9) and outputs the $(|\mathcal{V}|-1)$-(s,t) TFM path.

2.5.3 Analysis of CheaPay

The optimality of CheaPay is guaranteed by Theorem 1.

Theorem 1. *For any $h \leq |\mathcal{V}|-1$, $\Gamma^h(i,j)$ is the transaction fee of the h-(i,j) TFM path.*

Proof. We prove this theorem by induction.

Base case: $h = 1$.

- $v_i = v_j$. We have $\Gamma^1(i,j) = 0$, according to CheaPay-Init. Since it takes none to transfer a payment from v_i to itself, the statement holds.
- $v_j \in \mathcal{N}_i$ and $v_i = v_s$. We have $\Gamma^1(i,j) = 0$, according to CheaPay-Init. Because v_s is the sender, v_s does not charge the transaction fee. Thus, the statement holds.
- $v_j \in \mathcal{N}_i$ and $v_i \neq v_s$. We have $\Gamma^1(i,j) = f_{i,j}$, according to CheaPay-Init. Because the 1-(i,j) TFM path should go directly from v_i to v_j, the statement holds.
- $v_j \notin \mathcal{N}_i$. We have $\Gamma^1(i,j) = \infty$, according to CheaPay-Init. Because it is impossible to transfer from v_i to v_j through no more than 1 channel, the statement holds.

Inductive step: Assume that for any integer $1 \le m \le |\mathcal{V}| - 2$, $\Gamma^m(i,j)$ is the transaction fee of the m-(i,j) TFM path. We need to show that $\Gamma^{m+1}(i,j)$ is the transaction fee of the $(m+1)$-(i,j) TFM path.

Since the m-(i,j) TFM path must go through one of v_i's transferees, it suffices that either $\Gamma^h(i,j)$ or $\Gamma^{cand}(i,j) = \min_{v_k \in \mathcal{N}_i} \Gamma^m(k,j) + f_{i,k}$ is the transaction fee of the $(m+1)$-(i,j) TFM path.

- $\Gamma^m(i,j) \le \Gamma^{cand}(i,j)$. It indicates that using one more payment channel does not decrease the transaction fee. Therefore, the $(m+1)$-(i,j) TFM path remains the m-(i,j) TFM path. Because $\Gamma^m(i,j)$ is the transaction fee of the m-(i,j) TFM path, the statement holds.

- $\Gamma^m(i,j) > \Gamma^{cand}(i,j)$. It indicates that using one more payment channel decreases the transaction fee. But we still need to check if both the timeliness and feasibility constraints are satisfied. If they are satisfied, it suffices that $\Gamma^{m+1}(i,j)$ is the transaction fee of the $(m+1)$-(i,j) TFM path, the statement holds. ■

We now analyze the message complexity of CheaPay. First, CheaPay-Init builds two $O(|\mathcal{V}|)$ tables on v_i, where $|\mathcal{V}|$ is the total number of nodes. Second, CheaPay-Update sends messages to a node's transferors and request messages from the node's transferees. Therefore, CheaPay-Update exchanges $O(|\mathcal{E}|)$ messages on each node in each iteration. CheaPay-Routing's message complexity is dominated by CheaPay-Init and CheaPay-Update on each node, which is $O(|\mathcal{V}||\mathcal{E}|)$ in each iteration. In total, the overall message complexity of CheaPay is $O(|\mathcal{V}|^2|\mathcal{E}|)$.

2.6 Performance Evaluation

In this section, we evaluate the performance of CheaPay. As we surveyed in Section 2.2, there is no existing routing algorithm designed for transaction fee minimization subject to the timeliness and feasibility constraints in PCNs. Therefore, we demonstrate the effectiveness of CheaPay by comparing it to baseline algorithms.

2.6.1 Environment Setup

We use a real-world dataset sampled from the path-based transaction network Ripple [12, 22]. In particular, it is the complete network from November 2016 and all link modifications and transactions since its creation in January 2013. To evaluate the impact of the network size, we extract induced subgraphs consisting of $50, 100, \ldots, 250$ nodes. We assume that the transaction fees of all payment channels are distributed uniformly at random over $(0, 1]$. We compare CheaPay to the following baseline algorithms:

- *Cheapest:* It first finds a payment path that minimizes the total transaction fee, then checks if the path satisfies both the timeliness and feasibility constraints. If both constraints are satisfied, the

payment is accepted. Otherwise, the payment is rejected.

- *Widest:* It first finds a payment path that maximize the minimize channel balance on a path, then checks if the path satisfies both the timeliness and feasibility constraints. If both constraints are satisfied, the payment is accepted. Otherwise, the payment is rejected.

2.6.2 Performance Metrics

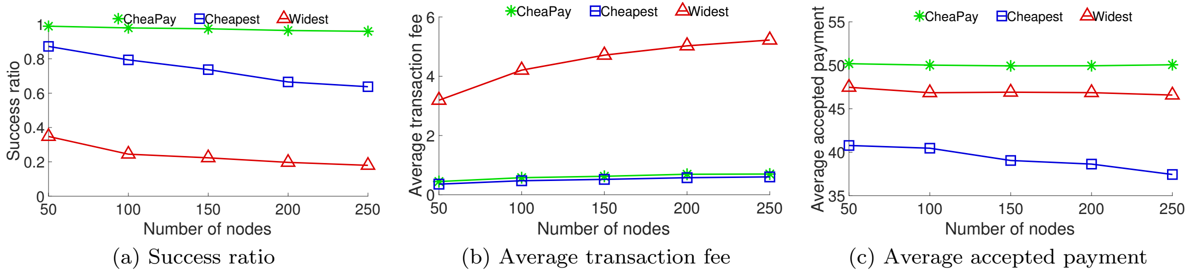
We use the following metrics for performance evaluation:

- *Success ratio:* The percentage of accepted payment requests.
- *Average transaction fee:* Average transaction fee over all accepted payment requests.
- *Average accepted payment:* Average payment over all accepted payment requests.

We do not evaluate the convergence speed of CheaPay, because it is a variant of the distributed Bellman-Ford Algorithm [29, 30] and thus has the same convergence speed.

2.6.3 Evaluation of CheaPay

Figure 2.3 shows success ratios, average transaction fees and average accepted payments of CheaPay, Cheapest and Widest.



(a) Success ratio (b) Average transaction fee (c) Average accepted payment

Figure 2.3 Impact of number of nodes on CheaPay, Cheapest and Widest.

Figure 2.3(a) shows the comparison of success ratios achieved by CheaPay, Cheapest and Widest. CheaPay outperforms the other algorithms in terms of the success ratio, due to its timeliness and feasibility guarantees. We can witness the growing gap between CheaPay and Cheapest, which indicates that focusing only on minimizing the total transaction fee without considering the timeliness and feasibility constrains decreases the success ratio significantly. Widest describes how well an algorithm can do without taking into account transaction fee minimization, timeliness or feasibility constraint. As expected, Widest gives the worst success ratio, because Widest only focuses on maximizing the minimum channel balance on a path, which possibly makes the path non-timely or infeasible. All algorithms have dropping success ratios

with more nodes. This is because although the number of nodes increases, the percentage of paths that satisfy both the timeliness and tolerance constraints decreases.

Figure 2.3(b) shows the comparison of the average transaction fees achieved by CheaPay, Cheapest and Widest. Cheapest gives a slightly lower average transaction fee than CheaPay, because Cheapest has a much lower success ratio and intends to accept paths with low transaction fees. Widest gives the highest average transaction fee, because Widest only focuses on maximizing the minimum channel balance on a path, but ignores minimizing the transaction fee.

Figure 2.3(c) shows the comparison of average accepted payments achieved by CheaPay, Cheapest and Widest. CheaPay outperforms the other algorithms in terms of the average accepted payment, due to its timeliness and feasibility guarantees. The average accepted payment of Widest drops with more nodes, because the minimum channel balances of paths decrease.

2.7 Conclusion

In this work, we studied the Transaction Fee Minimization (TFM) problem for routing in PCNs, while guaranteeing both the timeliness and feasibility constraints. We present an optimal distributed algorithm CheaPay for the TFM problem. Extensive simulations demonstrate that the proposed algorithm CheaPay achieves outstanding success ratio and average acceptance value compared to baseline algorithms.

CHAPTER 3
ROBUST PAYMENT ROUTING PROTOCOL WITH APPROXIMATION GUARANTEE IN PAYMENT CHANNEL NETWORKS

Among many proposals to improve the cryptocurrency scalability, one of the most promising and mature solutions is the payment channel network (PCN), which offers the off-chain settlement of transactions with minimal involvement of expensive blockchain operations. However, transaction failures may occur due to external attacks or unexpected conditions, e.g., an uncooperative user becoming unresponsive. In this work [31, 32], we present a distributed robust payment routing protocol RobustPay$^+$ to resist transaction failures, which achieves robustness, efficiency, distributedness and approximate optimization. Specifically, we investigate the problem of robust routing in PCNs from an optimization perspective, which is to find a pair of payment paths for a payment request, while minimizing the worst-case transaction fee, subject to the timeliness and feasibility constraints. We present a distributed 2-approximation algorithm for this problem. Moreover, we modify the original Hashed Time-lock Contract (HTLC) protocol and adapt it to the robust payment routing protocol to achieve robustness and efficiency. Extensive simulations demonstrate that RobustPay$^+$ significantly outperforms baseline algorithms in terms of the success ratio and the average accepted value.

3.1 Introduction

Over the past decade, the blockchain-based cryptocurrencies have risen to more than \$80 billion in peak capital, including Bitcoin [7], Ethereum [13], and Ripple [12]. These cryptocurrencies make use of the blockchain technology to achieve real-time total settlement [14] of different currencies and assets [33, 34], which is much cheaper than the current central bank system. Nevertheless, when attempting to scale up blockchains like Bitcoin and Ethereum, several concerns emerge. Firstly, every participant needs to know every single transaction to ensure a unique and synchronized global status. This leads to high overhead and demand for local storage. For instance, it could cost a Bitcoin user almost 20 GB (can increase to 60 GB after three years) of additional storage each year [15]. Further, blocks can only be added at a certain maximum rate determined by the necessary proof-of-work computations that need to be carried out by generating a number whose hash value that starts with a pre-decided number of zeros. Taking the Bitcoin blockchain as an example, the maximum number of transactions per second (tps) is only 7 [16], which is not comparable to over $47,000$ peak tps processed by Visa [17].

The payment channel network (PCN) was proposed to tackle the scalability issues [18]. PCNs can process instant and less valuable payments without involving blockchain transactions which are slow and

expensive. Only the initial and final balances of each channel are required to be registered. PCNs have been employed to develop Bitcoin's Lightning Network [18] and Ethereum's Raiden Network [19]. The uniqueness of PCNs is that PCNs can significantly lower the transaction fees than in blockchain. It is achieved by allowing a payment sent to the recipient through multiple hops between payment channels. In the payment transferring process, hosts of the channels on the route can charge fees accordingly. Therefore, the key motivation is to optimize the routing in PCNs and guarantee the success of a payment.

Several reported research works have investigated payment routing in PCNs [21–25, 35]. However, these efforts either emphasize on privacy [21, 22], or underestimate the importance of key realistic constraints such as the transaction fees [21–25, 35]. One misconception of the routing in PCNs is that PCNs may be treated as a conventional computer ad-hoc network. In fact, the complexity of routing in PCNs is increased by two distinct features, which are not found in conventional ad-hoc networks. The first distinct feature is the *feasibility constraint*, which is resulted from that fees paid to the users along the route have to be sent together with the payment to the targeted recipient. The second feature is referred to as the *timeliness constraint*, which is due to that the tolerance on the cooperating time in payment forwarding process of each PCN user is different. Due to these differences between PCNs and computer ad hoc networks, PCNs cannot apply the algorithms for conventional routing problems directly.

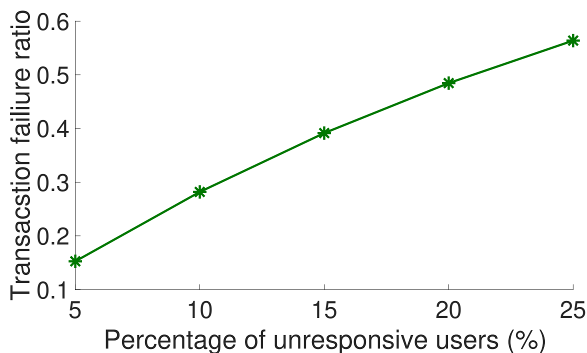


Figure 3.1 Impact of unresponsive users on the transactions

Recently, Zhang *et al.* [20] designed an optimal algorithm CheaPay to generate a single payment path that minimizes the transaction fee in PCNs and satisfies both the feasibility and timeliness constraints. However, CheaPay cannot resist transaction failures due to unexpected conditions or external attacks along the path. For example, some uncooperative intermediate users on the path may become unresponsive intentionally or unintentionally. Unresponsive users are a liability to channel participants who own funds in PCNs. Being unresponsive means that the channel funds may be on hold in the channel until the timeout period. A channel participant keeps funds in a channel either to make a payment for goods or service

(which now is delayed), or to earn transaction fees for being an intermediate node on a payment path to forward a transaction. Since the value of cryptocurrencies fluctuates dramatically, the time value of money (TVM) concept in financial management also applies to the blockchain-based cryptocurrencies, or might even contribute more according to the historic volatility. Thus, an unresponsive user makes the channel funds wasted during the timeout period. It is similar to a denial of service attack, which ties up cryptocurrencies rather than bandwidth. In order to evaluate the impact of the unresponsive users on the transactions, we conducted simulations on a real-world dataset from the Bitcoin Lightning Network [36]. The simulation result in Figure 3.1 shows that the transactions failure ratio increases by 15%, when 5% users are unresponsive. The failure ratio increases with unresponsive users. This indicates that unresponsive users are indeed an issue and need to be taken into consideration during payment routing in practice. Therefore, PCNs are expected to provide better *robustness* against transaction failures due to unresponsive users, *i.e.*, a payment routing protocol satisfies robustness, if it constructs two or more node-disjoint payment paths, where each payment path can fulfill the payment request. A payment is transferred on these payment paths simultaneously. If one path fulfills the payment first, the other path(s) will be invalidated.

In this work, we investigate the robust payment routing by constructing two node-disjoint payment paths for a payment request in PCNs. A robust payment routing has a number of distinct characteristics, thus it is expected to satisfy a set of desired properties. First, a robust payment routing protocol is expected to minimize the worst-case transaction fee, which is referred to as *optimization*, since it constructs more than one node-disjoint payment paths to resist transaction failures. As for NP-hard problems, we can only seek for *approximate optimization*. Secondly, a robust payment routing protocol should satisfy *efficiency*, i.e., to minimize the routing and payment latency incurred by transmitting a payment through multiple payment paths simultaneously. Finally, a robust payment routing should satisfy *distributedness*, as no central administrative operator exists in PCNs. Even if such an operator exists, it would not be trusted.

In face of these challenges, we propose RobustPay$^+$, a robust payment routing protocol that satisfies robustness, approximate optimization, efficiency and distributedness. Specifically, we investigate the problem of robust payment routing in PCNs from an optimization perspective. This problem is referred to as the *Maximum Transaction Fee Minimization (MTFM)* problem: minimizing the maximum transaction fee of a pair of node-disjoint paths to transfer a payment from the sender to the recipient, while guaranteeing that both the timeliness and feasibility constraints are satisfied for each involved payment channel on this pair of node-disjoint paths. Meanwhile, we design an HTLC mechanism providing more flexible choices and security to users and adapt it to the robust payment routing protocol. *The main contributions of this work are*:

- To the best of our knowledge, we are the first to consider the robust payment routing protocol, which provides resistance to transaction failures in PCNs.
- We investigate important design goals of payment routing in PCN, which are referred to as robustness, efficiency, distributedness and approximate optimization.
- We propose RobustPay$^+$, a distributed *Robust Pay*ment routing protocol against transaction failures in PCNs. RobustPay$^+$ consists of three stages: Payment Path Construction, HTLC Establishment and Payment Forwarding.
- We enhance the robustness for payment routing in PCNs by constructing two node-disjoint paths for a payment request and achieve approximate optimization by designing a distributed 2-approximation algorithm to minimize the worst-case transaction fee.
- We also modify the original HTLC protocol and adapt it to the robust payment routing to guarantee efficiency.
- Extensive simulations demonstrate that RobustPay$^+$ not only minimizes the worst-case transaction fee, but also achieves superior success ratio and average accepted payment value over baseline algorithms.

The remainder of the work is organized as follows. In Section 3.2, we provide a brief literature review of related work. In Section 3.3, we formally outline the design goals and give the problem formulation. In Section 3.4, we illustrate the robust payment routing protocol RobustPay$^+$, demonstrate the design of the routing algorithm and analyze the properties. In Section 3.5, we test and validate the performance of RobustPay$^+$ by comparing it to baseline algorithms. We summarize this work in Section 3.6.

3.2 Related Work

Up to now, there are only limited efforts on studying the routing problems in PCNs. Three years ago, one of the pioneering decentralized routing algorithms for PCNs, known as Flare [23]. In this algorithm, there is a routing table on each node, formed by the adjacent nodes that are close in hop distance and paths to a list of beacon nodes. The privacy-reserving routing problem was studied by Malavolta [21]. He developed SilenWhisper, which is a routing scheme based on Landmark Routing [26]. In SilenWhisper, the landmark that is passed by all paths may result in unnecessarily long paths. Furthermore, this approach can violate decentralization, which is the only intention of using the blockchain system.
SpeedyMurmurs [22] used an improved algorithm based on embedding-based path discovery [27]. In this way, the weakness of SilentWhisper could be overcome and improved in terms of success ratio, delay of

payment, overhead, length of path, and stabilization. Rohrer *et al.* [24] sketched the payment flow as multiple paths adding up together to make use of the available capacities in the network in an efficient way. Following Rohrer's path, Yu *et al.* [25] outlined a scattered algorithm, which improved the success ratio of payment and reduced the system overhead. However, all the previous studies either concentrate on the privacy [21, 22] or simplifying the problem without taking into account the hop-dependent constraints [21–25].

Recently, Bagaria *et al.* [37] devised a technique that constructs redundant payment paths free of counterparty risk. However, this solution was designed for multi-path routing schemes without considering optimal routing. Zhang *et al.* [20] proposed a distributed algorithm CheaPay to minimize the transaction fee of a payment path in PCNs while considering the timeliness and feasibility constraints. But CheaPay did not provide robustness to payment routing in PCNs, including responses to transaction failures due to unexpected conditions, e.g., an uncooperative user becoming unresponsive.

3.3 System Model and Problem Formulation

In this section, we outline the desired design goals and give a precise problem formulation. The network model and the payment model are the same as described in Section 2.4.

3.3.1 Design Goals

Under the blockchain, network and payment models specified above, we derive a set of desirable design goals that a payment routing protocol should satisfy, which are elaborated below.

- *Robustness*: A payment routing protocol satisfies robustness, if it generates two or more node-disjoint payment paths, where each of them can fulfill the payment request individually. In PCNs, a node may become unresponsive due to external attacks, unexpected conditions or uncooperative behaviors, which leads to transaction failures. If the routing protocol generates only a single path for a payment request, it fails when a node on this path becomes unresponsive. In order to resist such transaction failures, it is necessary to generate more than one node-disjoint payment paths to transfer a payment, where no common intermediate user is shared on both paths. The payment is forwarded on these payment paths simultaneously. If one path fulfills the payment first, the other path(s) will be invalidated.

- *Approximate Optimization*: A payment routing protocol satisfies optimization, if it minimizes the worst-case transaction fee for a payment request. Since a robust payment routing protocol constructs two or more payment paths, where each payment path can fulfill the payment request, it is necessary to minimize the maximum transaction fee of these node-disjoint payment paths. However, this

optimization problem could be NP-hard due to the distinct characteristics in PCNs. Sometimes we can only strive to achieve approximate optimization. We will discuss the NP-hardness of this problem in Section 3.3.2.

- *Efficiency*: A payment routing protocol satisfies efficiency, if it minimizes the routing and payment latency incurred by transmitting a payment through more than one path simultaneously. In the robust payment routing, only one payment path will be used to fulfill the payment request, and the other payment path(s) will be invalidated. Thus, it is necessary to guarantee that this payment path introduces the minimum latency.

- *Distributedness*: A payment routing protocol satisfies distributedness, if it dose not rely on a centralized trusted party. Centralized routing is subject to a single point of failures upon external attacks and hence cannot be trusted by users. Instead, users need to communicate with each other and conduct local computations to find routes for payments.

3.3.2 MTFM Problem Formulation

Following the desirable design goals that are outlined above, we consider the *Maximum Transaction Fee Minimization (MTFM)* problem for routing in PCNs. To formally formulate our studied problem, we introduce the following necessary concepts.

Timeliness Constraint: A payment path $p = v_0 \to v_1 \to \ldots \to v_L$ satisfies timeliness constraint, if the payment request can be successfully fulfilled within the HTLC tolerance of each IC, *i.e.*, $\tau_{i,i+1} \geq L - i$, $\forall i \in [0, L-1]$. Timeliness guarantees the commitment of honest processing at any IC.

Feasibility Constraint: A payment path $p = v_0 \to v_1 \to \ldots \to v_L$ satisfies feasibility constraint, if the payment request can be successfully transferred through each IC. Specifically, the balance $b_{i,i+1}$ of $e = (v_i, v_{i+1})$ should be at least the payment amount a plus the accumulation of transaction fees paid to the IUs that follow v_i on the path, *i.e.*, $b_{i,i+1} \geq a + F_p(i+1, L)$, $\forall i \in [0, L-1]$.

Apparently, a single payment path cannot guarantee *robustness*, since some uncooperative IUs on the path may decide to become unresponsive or suffer from external attacks. In order to avoid such a transaction failure, we can establish a pair of node-disjoint payment paths, where either payment path can fulfill the given payment request. Since either payment path can be used to transfer a payment, it is necessary to guarantee *optimization*, which is to minimize the maximum transaction fee of the pair of node-disjoint payment paths, while still guaranteeing timeliness and feasibility constraints. Towards this goal, we consider the following optimization problem in PCNs:

Maximum Transaction Fee Minimization (MTFM): Given a payment request $R = (v_s, v_t, a)$, find a pair of timely and feasible node-disjoint payment paths, either of which can fulfill R, such that the maximum transaction fee of these two payment paths is minimized.

In the following, we prove the NP-hardness of the MTFM problem.

Theorem 2. *The MTFM problem is NP-hard.*

Proof. We prove the MTFM problem is NP-hard by reduction from the Min-Max 2-path problem, which has been proved to be NP-complete in [38].

The Min-Max 2-path problem is defined as follows: *Given a directed graph $G = (\mathcal{V}, \mathcal{E})$ with a source s and a destination t, where each edge $(v_i, v_j) \in \mathcal{E}$ has a non-negative length, find two node-disjoint paths from s to t, such that the maximum of their lengths is minimized.*

In the Min-Max 2-path problem, each edge is associated with a length; in MTFM, each edge is associated with three attributes: a transaction fee, a channel balance and an HTLC tolerance. Thus, given an arbitrary instance of the Min-Max 2-path problem, we set the transaction fee of each edge to the length, set the the channel balance of each edge to ∞ and set the the HTLC tolerance of each edge to ∞. This converts the instance of the Min-Max 2-path problem to one of MTFM. It is clear that this instance of MTFM has a solution, if and only if the instance of the Min-Max 2-path problem has a solution. Thus, the MTFM problem is NP-hard. ■

Since the MTFM problem is proven to be NP-hard, we can only strive to achieve an approximation solution for the MTFM problem.

3.4 RobustPay$^+$: A Distributed Robust Payment Routing Protocol in PCNs

In this section, we present the design of RobustPay$^+$. We first provide the high-level overview and intuition behind RobustPay$^+$ and then follow the design goals that are outlined in Section 3.3.1 to provide a detailed protocol description.

3.4.1 Design Rationale

A payment *transaction failure* occurs, if there are external attacks or unexpected conditions along the payment path, e.g., an uncooperative IU decides to become unresponsive by not providing the preimage of H to its transferor within the HTLC tolerance. Therefore, it is necessary to design a payment routing protocol that satisfies robustness, approximate optimization, efficiency and distributedness. we propose a distributed robust payment routing protocol RobustPay$^+$ against transaction failures for payment transmissions in PCNs.

RobustPay$^+$ achieves robustness, approximate optimization and distributedness by designing a distributed algorithm derived from the Suurballe's Algorithm [39] for the min-sum 2-path problem. Meanwhile, RobustPay$^+$ achieves efficiency by applying and modifying the HTLC mechanism introduced in [18].

3.4.2 Design Challenges

To apply the Suurballe's Algorithm that was originally designed for centralized systems, we need to address several challenges. The first one is to transform the Suurballe's Algorithm into a distributed algorithm, where each node only has local knowledge. Second, the timeliness and feasibility constraints need to be taken into consideration.

Another challenge comes from the establishment of HTLCs, since the HTLC mechanism was originally designed to guarantee the off-chain security almost the same as the original blockchain, when routing on a single payment path. To apply the HTLC mechanism in a robust payment routing protocol, we need to modify the current HTLC protocol carefully to satisfy efficiency as well as off-chain security.

We address these challenges in the following subsections and outline how the users are expected to proceed in order to execute a payment request in RobustPay$^+$.

3.4.3 Payment Path Construction

The first stage is to construct two payment paths for a payment request, such that either payment path can fulfill the payment request, and there is no *intermediate user* (IU) shared on both payment paths. Such two payment paths are referred to as a pair of node-disjoint payment paths. If a transaction failure occurs on a payment path due to unexpected conditions, the other payment path can still work to fulfill the transaction. In this section, we design and analyze RobustPay$^+$, a distributed approximation algorithm that determines a pair of node-disjoint payment paths.

3.4.3.1 Design Rationale

Before we formally describe the design of RobustPay$^+$, we introduce the main notations in Table 3.1 and necessary definitions in the following.

(i, j)-TFM path [20]: A payment path is an (i, j)-TFM path, if it satisfies the following conditions: 1) this payment path satisfies both the timeliness and feasibility constraints; 2) this payment path has the minimum transaction fee among all payment paths from v_i to v_j.

h-(i, j) TFM path: [20] A payment path is an h-(i, j) TFM path, if it satisfies the following conditions: 1) the length of this payment path is no more than h; 2) this payment path satisfies both the timeliness

and feasibility constraints; 3) this payment path has the minimum transaction fee among all payment paths from v_i to v_j.

(i, j)-MTFM path pair: A pair of payment paths is an (i, j)-MTFM path pair, if it satisfies the following conditions:

1. these two payment paths satisfy both the timeliness and feasibility constraints;
2. these two payment paths do not share any common intermediate user (IU);
3. the maximum transaction fee of these two payment paths is the minimum among all pairs of payment paths from v_i to v_j.

Table 3.1 Main notations

Notation	Meaning
$f_{i,j}$	the amount of transaction fee of channel (v_i, v_j)
$b_{i,j}$	the amount of remaining balance of channel (v_i, v_j)
$\tau_{i,j}$	the HTLC tolerance of channel (v_i, v_j)
$\mathcal{IN}_i$	the set of v_i's transferees
$\mathcal{OUT}_i$	the set of v_i's transferors
$\Gamma^h(i,j)$	the minimum transaction fee corresponding to the h-(i,j) TFM path
$\mathcal{N}_h(i,j)$	v_i's outgoing transferees corresponding to $\Gamma^h(i,j)$
$v_{i'}$	the auxiliary node of v_i

Our algorithm RobustPay$^+$ is based on the distributed Suurballe's Algorithm [39], which was originally designed to minimize the total cost of two disjoint paths in a centralized system. Specifically, we design RobustPay$^+$ by implementing CheaPay [20], which generate a single (i, j)-TFM path in PCNs. RobustPay$^+$ algorithm consists of five stages: initialization, first routing, graph transformation, second routing and pair generation. In the initialization stage, the algorithm splits a node v_i into two nodes v_i and $v_{i'}$ by creating an auxiliary edge $(v_i, v_{i'})$ and reassigning all outgoing edges on v_i as outgoing edges on $v_{i'}$, because the Suurballe's Algorithm [39] was originally designed for finding a pair of edge-disjoint payment paths. In the first routing stage, the algorithm gives an (s, t)-TFM path by revoking CheaPay [20], which is an optimal distributed algorithm that minimizes the transaction fee of a payment path in PCNs while considering the timeliness and feasibility constraints. Note that this path is not one the two payment paths for final payment transaction. In the graph transformation stage, the algorithm transforms G to a residual graph. In the second routing stage, the algorithm outputs an (s, t)-TFM path in the residual graph, similarly to the first routing stage. In the pair generation stage, the algorithm generates a pair of node-disjoint payment paths by discarding the reversed edges in the second payment path from both payment paths and reconstructing the remaining edges.

Currently, source routing is utilized in the Lightning Network [36], where the sender node is responsible for calculating the entire path, from the sender to the recipient. To do so, the sender node needs to download a snapshot of the PCN topology to learn each channel's transaction fee and HTLC tolerance. Because the channel balance information is not public due to privacy concerns and varies over time due to dynamics, the transactions may fail as the actual balances on the channels may not satisfy the payment. Therefore, RobustPay$^+$ does not adopt source routing. Instead, each node makes distributed routing decisions based on a distributed Bellman-Ford style algorithm. In this setting, a node checks the balance availability with its neighbors during the stage of payment path construction.

RobustPay$^+$ establishes HTLCs after the payment path construction stage. Thus, collaterals are not locked during path construction. Indeed, it is possible that transactions may fail due to dynamic balance change in the PCN. Although the main goal of this work is to resist transaction failures due to unresponsive nodes, RobustPay$^+$ can also mitigate transaction failures caused by dynamic balance change. This is because RobustPay$^+$ establishes a pair of node-disjoint paths that are guaranteed to be edge-disjoint. Even if there are edge failures along one path due to the dynamic change between the payment path construction stage and the payment forwarding stage, the other path can still be used for the payment transaction.

3.4.3.2 Design of RobustPay$^+$

In this section, we describe the details of RobustPay$^+$, which is illustrated in Algorithms 4, 5 and 6.

The initialization stage is shown in RobustPay$^+$-Init (Algorithm 4). RobustPay$^+$-Init splits a node v_i into two nodes by creating an auxiliary node $v_{i'}$ and reassigning all outgoing edges on v_i as outgoing edges on $v_{i'}$. We use $\mathcal{IN}_i$ and $\mathcal{OUT}_i$ to denote the set of v_i's ingoing transferors and the set of v_i's outgoing transferees, respectively (Lines 1 and 2). Thus, v_i is $v_{i'}$'s transferor, and $v_{i'}$ is v_i's transferee (Lines 3 and 4). An auxiliary edge $(v_i, v_{i'})$ connects v_i and $v_{i'}$. Because there is no constraint to transfer a payment from v_i to its auxiliary node $v_{i'}$, we set $f_{i,i'} = 0$, $\tau_{i,i'} = \infty$, and $b_{i,i'} = \infty$ (Line 5). In order to reassign all outgoing edges on v_i as outgoing edges on $v_{i'}$, RobustPay$^+$-Init adds edges to connect $v_{i'}$ and v_i's transferees (Lines 6 to 11) and removes edges that connect v_i and v_i's transferees. We shall run Algorithm 4 to initialize the node splitting for each node $v_i \in \mathcal{V}$. Tables Table 3.2 and Table 3.3 show the example of routing tables that are stored on v_i before and after node splitting.

The first routing stage revokes CheaPay [20] to find an (s, t)-TFM path p_R^1 in G. Note this path is not one of the two payment paths for the final payment transaction. A transaction fee table $\{\Gamma(i, j)\}_{v_j \in \mathcal{V}}$ is generated on each node v_i, which stores the transaction fees of the (i, j)-TFM paths. Therefore, RobustPay$^+$ can reuse the transaction fee table $\{\Gamma(s, i)\}_{v_i \in \mathcal{V}}$ for graph transformation in the next stage.

Table 3.2 An Example of Routing Table before Node Splitting

i	j				
A	B	C	D	E	F
$\Gamma^h(i,j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i,j)$	A	-	-	A	-
number of hops	1	-	-	1	-
feasible	✓	-	-	✓	-

Table 3.3 An Example of Routing Tables After Node Splitting

i	j				
A	B	C	D	E	F
$\Gamma^h(i,j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i,j)$	A'	-	-	A'	-
number of hops	2	-	-	2	-
feasible	✓	-	-	✓	-
i'	j				
A'	B	C	D	E	F
$\Gamma^h(i',j)$	0	∞	∞	0	∞
$\mathcal{N}_h(i',j)$	A'	-	-	A'	-
number of hops	1	-	-	1	-
feasible	✓	-	-	✓	-

The graph transformation stage is shown in RobustPay$^+$-Trans (Algorithm 5). RobustPay$^+$-Trans transforms the original graph to a residual graph by reusing the transaction fee table $\{\Gamma(s,i)\}_{v_i \in \mathcal{V}}$ obtained from the previous stage. First, v_i requests $\Gamma(s,i)$ from its transferor along the (s,i)-TFM path (Lines 1 to 3). Then, v_i modifies the transaction fee of each edge (i',j) by replacing $f_{i,j}$ by $\tilde{f}_{i',j} = f_{i',j} - \Gamma(s,j) + \Gamma(s,i')$ (Line 5). Also, if the modified transaction fee of an edge on path p_R^1 is 0, RobustPay$^+$-Trans reverses the direction of this edge (Lines 6 to 8).

The final stage is shown in RobustPay$^+$-Output (Algorithm 6). RobustPay$^+$-Output outputs the (s,t)-MTFM path pair. First, RobustPay$^+$-Output runs RobustPay$^+$-Init to initialize the auxiliary node $v_{i'}$ on each node $v_i \in \mathcal{V}$. Second, RobustPay$^+$-Output revokes CheaPay to generate an (s,t)-TFM path in G. Third, RobustPay$^+$-Output runs RobustPay$^+$-Trans on each node $v_i \in \mathcal{V}$ to transform the original graph to a residual graph G_v. Then, RobustPay$^+$-Output revokes CheaPay again to generate an (s,t)-TFM path in G_v. Finally, RobustPay$^+$-Output outputs the (s,t)-MTFM path pair by discarding the reversed edges in the second payment path from both payment paths and reconstructing the remaining edges.

Algorithm 4: RobustPay$^+$-Init

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a node v_i and an auxiliary node $v_{i'}$.

```
1  IN_i ← the set of v_i's transferor nodes;
2  OUT_i ← the set of v_i's transferee nodes;
3  Create an auxiliary node v_{i'};
4  OUT_i ← OUT_i ∪ {v_{i'}}; OUT_{i'} ← ∅; IN_{i'} ← {v_i};
5  f_{i,i'} ← 0; τ_{i,i'} ← ∞; b_{i,i'} ← ∞;
6  for v_j ∈ OUT_i do
7  |   f_{i',j} ← f_{i,j}; f_{i,j} ← ∞;
8  |   τ_{i',j} ← τ_{i,j}; τ_{i,j} ← 0;
9  |   b_{i',j} ← b_{i,j}; b_{i,j} ← 0;
10 |   OUT_i ← OUT_i\{v_j}; OUT_{i'} ← OUT_{i'} ∪ {v_j};
11 end
12 return v_{i'}
```

Algorithm 5: RobustPay$^+$-Trans

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a residual graph G_t.

```
1  for v_k ∈ IN_i do
2  |   Request Γ(s, i) along the (s, i)-TFM path;
3  end
4  for v_j ∈ OUT_{i'} do
5  |   f_{i',j} ← f_{i',j} − Γ(s, j) + Γ(s, i');
6  |   if f̃_{i',j} == 0 and (v_{i'}, v_j) ∈ p_R^1 then
7  |   |   f_{j,i'} ← 0; f_{i',j} ← ∞;
8  |   end
9  end
10 return G_v
```

Algorithm 6: RobustPay$^+$-Output

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a payment request $R = (v_s, v_t, a)$.
Output: a payment path p_R.

```
1  p_R^1 ← ∅; p_R^2 ← ∅;
2  for v_i ∈ V do Init(G, v_i);
3  p_R^1 ← CheaPay(G, R);
4  for v_i ∈ V do Trans(G, v_i);
5  p_R^2 ← CheaPay(G, R);
6  for (v_i, v_j) ∈ p_R^2 do
7  |   if (v_j, v_i) ∈ p_R^1 then
8  |   |   p_R^1 ← p_R^1\{(v_j, v_i)}; p_R^2 ← p_R^2\{(v_i, v_j)};
9  |   end
10 end
11 Reconstruct the remaining edges of p_R^1, p_R^2;
12 return p_R^1, p_R^2
```

3.4.3.3 Analysis of RobustPay$^+$

In this subsection, we analyze the approximation ratio of RobustPay$^+$ as follows.

Theorem 3. RobustPay$^+$ outputs a 2-approximation solution to the MTFM problem.

Proof. We first prove that RobustPay$^+$ outputs an optimal solution to the following Min-Sum Transaction Fee (MSTF) problem: *Given a payment request $R = (v_s, v_t, a)$ and a PCN $G = (\mathcal{V}, \mathcal{E})$, find a pair of timely and feasible node-disjoint paths, either of which can fulfill R, such that the summation transaction fee of these two paths is minimized.*

Because RobustPay$^+$ is designed based on a distributed version of the Suurballe's Algorithm [39], RobustPay$^+$ outputs an optimal solution to the MSTF problem.

Let $\{p_1^*, p_2^*\}$ be the pair of paths in an optimal solution to the MTFM problem and $OPT = \max\{F_{p_1^*}, F_{p_2^*}\}$. Let $\{p_1, p_2\}$ be the pair of paths generated by RobustPay$^+$. Thus, we can get

$$\begin{aligned} 2\max\{F_{p_1^*}, F_{p_2^*}\} &\geq F_{p_1^*} + F_{p_2^*} \\ &\geq F_{p_1} + F_{p_2} \\ &\geq \max\{F_{p_1}, F_{p_2}\}, \\ 2OPT \geq 2\max\{F_{p_1^*}, F_{p_2^*}\} &\geq \max\{F_{p_1}, F_{p_2}\}. \end{aligned}$$

Therefore, RobustPay$^+$ outputs a 2-approximation solution to the MTFM problem. ■

We now analyze the message complexity of RobustPay$^+$. First, RobustPay$^+$-Init builds $O(|\mathcal{V}|)$ auxiliary nodes, where $|\mathcal{V}|$ is the total number of nodes. Second, RobustPay$^+$ calls CheaPay once, whose message complexity is $O(|\mathcal{V}|^2|\mathcal{E}|)$. Then RobustPay$^+$-Trans sends messages along each (s, i)-TFM path. Therefore, RobustPay$^+$-Trans exchanges $O(|\mathcal{E}|)$ messages on each node. RobustPay$^+$-Output's message complexity is dominated by CheaPay and RobustPay$^+$-Trans on each node, which is $O(|\mathcal{V}|^2|\mathcal{E}|)$. In total, the overall message complexity of RobustPay$^+$ is $O(|\mathcal{V}|^2|\mathcal{E}|)$.

3.4.4 HTLC Establishment

A Hashed Time-Locked Contract (HTLC) is a script that permits a designated party (the transferee) to spend funds by disclosing the preimage of a hash. It also permits a second party (the transferor) to spend the funds after a timeout is reached, in a refund situation. The original HTLC introduced in [18] was designed for payment routing in a single payment path. In the HTLC, the on hold payment is refunded to the transferor, only if the transferee does not provide the preimage of H within the HTLC tolerance. However, the HTLC does not provide the transferee with flexible choices to cancel a transaction before the

expiration. Even if the transferee decides to cancel a transaction, it can only wait until the expiration of the HTLC.

To adapt the HTLC protocol to RobustPay$^+$, we modify the original HTLC to provide more flexible choices as follows: If the transferee does not provide the preimage of H within the HTLC tolerance, or if the transferee cancels the transaction before the preimage of H is provided, the on hold payment in the HTLC is refunded to the transferor. A potential problem is that the recipient may claim the payment on both paths. In order to prevent this double-claim issue, we improve the HTLC by using two separate secrets on two payment paths, inspired by Boomerang [37]. However, Boomerang cannot prevent the sender from reverting both payments, since it only guarantees that the sender can revert all payments, if the recipient claims more than one payment. We cannot directly apply Boomerang and thus modify it to prevent the double-reverting issue, such that only one of the payment paths is randomly selected to be reversible. With this modification, the sender can revert the payment on the reversible path by providing the secret of the irreversible path, if the recipient claims the payment twice by producing secrets on both paths.

The script of the modified HTLC takes the following form, and the modification of the HTLC is highlighted in Table 3.4.

Table 3.4 Script of the modified HTLC

```
OP_IF
    OP_IF
        [HASHOP] ⟨digest⟩ OP_DROP OP_DUP
          OP_HASH160 ⟨buyer pubkey hash⟩
    OP_ELSE
        ⟨num⟩ [TIMEOUTOP] OP_EQUALVERIFY
          OP_DUP OP_HASH160 ⟨seller pubkey hash⟩
    OP_ENDIF
OP_NOTIF
    [CANCELOP] ⟨digest⟩ OP_DROP OP_DUP
    OP_HASH160 ⟨seller pubkey hash⟩
OP_ELSE
    ⟨num⟩ [TIMEOUTOP] OP_DROP OP_DUP
    OP_HASH160 ⟨buyer pubkey hash⟩
OP_ENDIF
OP_EQUALVERIFY
OP_CHECKSIG
```

A simple illustration of the modified HTLC is shown in Figure 3.2. Such a modification on HTLC can provide flexible choices for PCN users and security for senders. We formally give the following security guarantee:

Theorem 4. RobustPay$^+$ guarantees that the recipient cannot claim the payment twice.

Proof. Let $\{p_1, p_2\}$ the pair of node-disjoint payment paths generated by RobustPay$^+$. Let R_1 and R_2 be the preimages for the recipient to claim the payments on p_1 and p_2, respectively. Assume that the recipient claims the payments on both p_1 and p_2 by providing R_1 and R_2. This indicates the sender knows both R_1 and R_2. By providing R_1 on p_2, the sender can revert the payment transaction on p_2. Similarly, by providing R_2 on p_1, the sender can revert the payment transaction on p_1. Thus, it guarantees that the recipient cannot claim the payment twice. ■

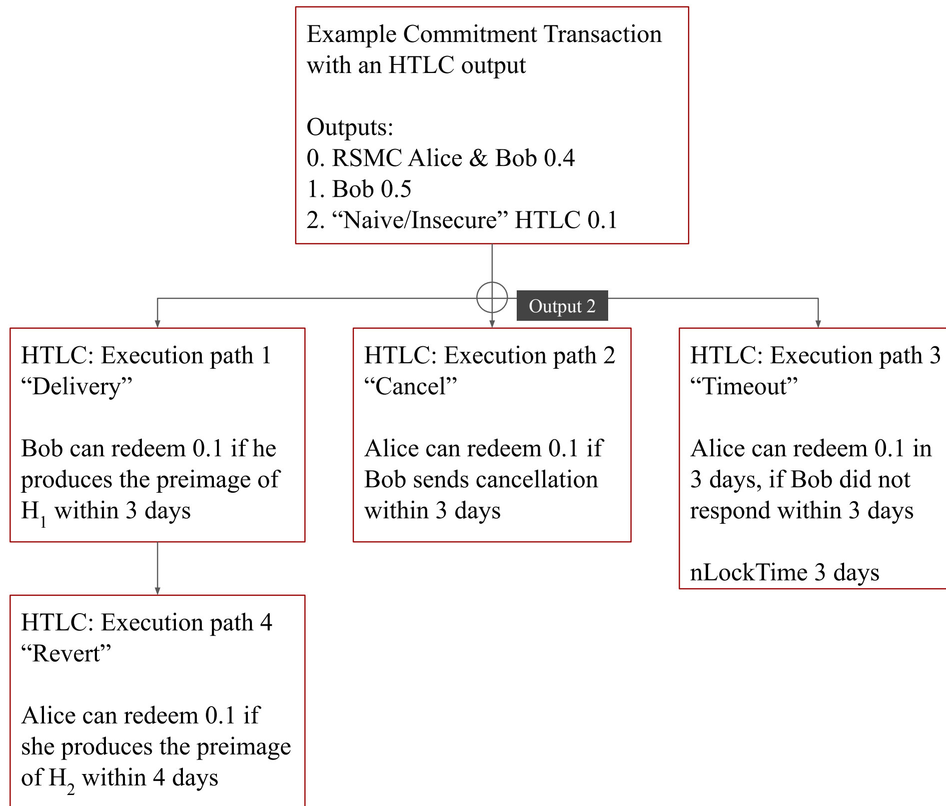


Figure 3.2 Modified hashed time-lock contract (HTLC). Alice sends a payment of 0.1 to the Bob via B and C with an HTLC tolerance of 3. Note that there are two possible spends from an HTLC output. If Bob can produce the preimage of H_1 within 3 days, Bob can redeem path 1. If Alice can produce the preimage of H_2 after Bob produces the preimage of H within 4 days, Alice can redeem path 4. If Alice sends cancellation before Bob can produce the preimage of H_1 within 3 days, Alice can redeem path 2. After 3 days, Alice is able to redeem path 3, if there is no response from Bob.

3.4.5 Payment Forwarding

After the payment path construction and the HTLC establishment processes, the sender can forward the payment to the recipient via the constructed payment paths. Once one of the two payment paths successfully transfers the payment to the recipient, the other payment path should be invalidated. If the

recipient tries to claim the payment on the second path, the sender can revert the payment on the first path. A simple illustration of the payment forwarding is shown in Figure 3.3. Two node-disjoint payment paths have been constructed in the previous stage, where A is the sender and D is the recipient. An HTLC has been created on each IC on both payment paths. Since D receives the HTLC from G on the lower (red) payment path earlier, D provides the preimage of H to C and receives the payment from C. Therefore, the upper (green) payment path is invalidated. D can choose to cancel the transaction from C to D. The on hold payment in the HTLC between C and D is refunded to C. So are the rest on hold payments in the HTLCs on the ICs along the upper (green) payment path. If the D tries to claim the payment on the upper (green) path, the A can revert the payment on the lower (red) path by providing the preimage of H, which indicates that the payment has be claimed. This guarantees that the payment cannot be double claimed.

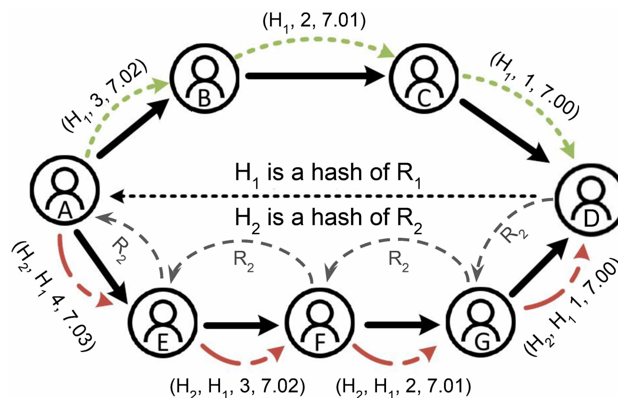


Figure 3.3 Payment Forwarding in RobustPay$^+$. The sender A sends a payment of 7 to the recipient D. Two node-disjoint payment paths have been constructed. One payment path is $A \rightarrow B \rightarrow C \rightarrow D$; another payment path is $A \rightarrow E \rightarrow F \rightarrow G \rightarrow D$. HTLCs are established on both payment paths simultaneously, from the sender A to the recipient D, sequentially. The upper (green) path is not reversible, and the lower (red) path is reversible. The recipient D provides the preimage of H to G on the lower (red) payment path and refunds C on the upper (green) payment path.

3.5 Performance Evaluation

In this section, we evaluate the performance of RobustPay$^+$. As we surveyed in Section 3.2, there is no existing payment routing protocol that satisfies robustness, optimization, efficiency or distributedness in payment channel networks. Therefore, we demonstrate the performance of RobustPay$^+$ by comparing it to CheaPay [20] and baseline algorithms.

3.5.1 Environment Setup

We implemented and modified a simulator for PCNs [20] to model the transaction arrivals and settlements. Transactions are serial and routed according to the routing algorithms, if the timeliness and

feasibility constraints are satisfied on the generated payment paths. The locked funds are unavailable for use by any node on the payment path. When a payment transaction is settled, these funds are released. The simulator supports payment transactions through a queue of pending payments. The queue is periodically polled to check if the transactions can progress further. The HTLC tolerance parameter is specified on each channel independently by the transferor on this channel, which represents the maximum time that the transferor is willing to wait for the preimage to confirm a transaction. Because the preimage is sent backwards from the recipient, the HTLC timeout parameter indicates the maximum distance (the number of hops) from a node to the recipient. Thus, the HTLC timeout parameter is at channel level rather than at a source-destination pair level. Since the HTLC timeout information is public in the Lightning Network, we use the real data in the simulation.

We obtained a real-world PCN topology from the Bitcoin Lightning Network [36]. In particular, we crawled a snapshot topology of the Lightning Network on July 14, 2020 [40]. To crawl the Lightning Network, we ran the Bitcoin Core daemon (bitcoind) [41], built a c-lightning [42] node on mainnet, and connected it to an existing Lightning node, which is the Bitstamp's Lightning Network node [43]. The network consists of $5,622$ nodes and $65,628$ channels. We use a real-world transaction dataset sampled from the path-based transaction network Ripple [12, 22]. In particular, it contains the complete network and transactions from November 2016 and all link modifications and transactions since its creation in January 2013. To evaluate the impact of the network size, we extract induced subgraphs consisting of $50, 100, \ldots, 250$ nodes. We assume that the transaction fees of all payment channels are distributed uniformly at random over (0, 1]. We compare RobustPay$^+$ to the following algorithms:

- *CheaPay* [20]: It finds a single payment path that minimizes the total transaction fee, while satisfying both the timeliness and feasibility constraints.
- *Cheapest:* First, it finds a payment path that minimizes the total transaction fee. Second, it finds a node-disjoint path that minimizes the total transaction fee by removing the intermediate node in the first path. Then, it checks if both paths satisfy both the timeliness and feasibility constraints. If both constraints are satisfied, the payment is accepted. Otherwise, the payment is rejected.
- *Widest:* First, it finds a payment path that maximizes the minimize channel balance on a path. Second, it finds a node-disjoint path that maximizes the minimize channel balance on a path by removing the intermediate node in the first path. Then, it checks if both paths satisfy both the timeliness and feasibility constraints. If both constraints are satisfied, the payment is accepted. Otherwise, the payment is rejected.

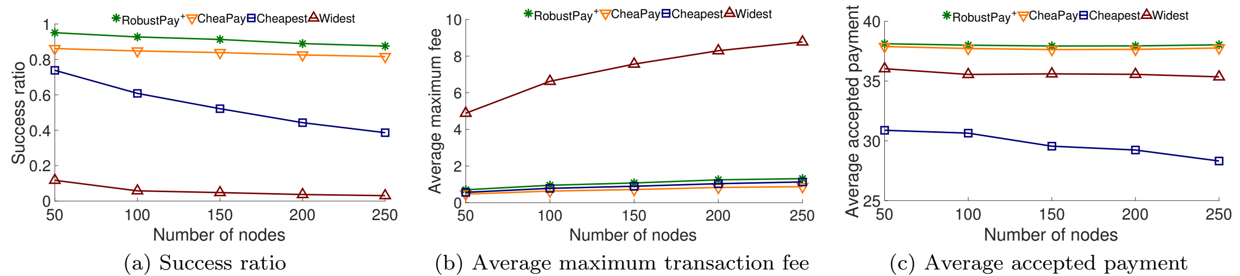
3.5.2 Performance Metrics

We use the following metrics for performance evaluation:

- *Success ratio:* The percentage of accepted payment requests.
- *Average maximum transaction fee:* Average maximum transaction fee of the pair of paths over all accepted payment requests.
- *Average accepted payment:* Average payment over all accepted payment requests.

3.5.3 Evaluation of RobustPay$^+$

Figure 3.4 shows success ratios, average maximum transaction fees and average accepted payments of RobustPay$^+$, CheaPay, Cheapest and Widest.



(a) Success ratio (b) Average maximum transaction fee (c) Average accepted payment

Figure 3.4 Impact of number of nodes on RobustPay$^+$, CheaPay, Cheapest and Widest.

Figure 3.4(a) shows the comparison of success ratios achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. RobustPay$^+$ outperforms the other algorithms in terms of the success ratio, due to its robustness, timeliness, and feasibility guarantees. CheaPay gives a lower success ratio than RobustPay$^+$, because CheaPay generates a single path and does not provide robustness. We can witness the growing gap between RobustPay$^+$ and Cheapest, which indicates that focusing only on minimizing the total transaction fee without considering the timeliness and feasibility constrains decreases the success ratio significantly. Widest describes how well an algorithm can do without taking into account the minimization of maximum transaction fee, timeliness or feasibility constraint. As expected, Widest gives the worst success ratio, because Widest only focuses on maximizing the minimum channel balance on a path, which possibly makes the path non-timely or infeasible. All algorithms have dropping success ratios with more nodes. This is because although the number of nodes increases, the percentage of paths that satisfy both the timeliness and tolerance constraints decreases.

Figure 3.4(b) shows the comparison of the average maximum transaction fees achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. Cheapest gives a slightly lower average maximum transaction fee than

RobustPay$^+$, because Cheapest has a much lower success ratio and intends to accept paths with low transaction fees. We can witness the gap between RobustPay$^+$ and CheaPay, which indicates that RobustPay$^+$ sacrifices a little bit of transaction fee minimization for robustness. Widest gives the highest average maximum transaction fee, because Widest only focuses on maximizing the minimum channel balance on a path, but ignores minimizing the transaction fee.

Figure 3.4(c) shows the comparison of average accepted payments achieved by RobustPay$^+$, CheaPay, Cheapest and Widest. RobustPay$^+$ outperforms the other algorithms in terms of the average accepted payment, due to its robustness, timeliness, and feasibility guarantees. CheaPay gives a slightly lower average accepted payment than RobustPay$^+$, because CheaPay does not provide robustness. The average accepted payment of Widest drops with more nodes, because the minimum channel balances of paths decrease.

We also evaluate the convergence speed of RobustPay$^+$. The result is shown in Figure 3.5. We can observe that the average number of messages increases with the the number of nodes. Since RobustPay$^+$ implements a variant of the distributed Suurballe's Algorithm [39], which is based on the distributed Bellman-Ford Algorithm [29, 30], RobustPay$^+$ has the growing trend of message complexity similar to that of Bellman-Ford.

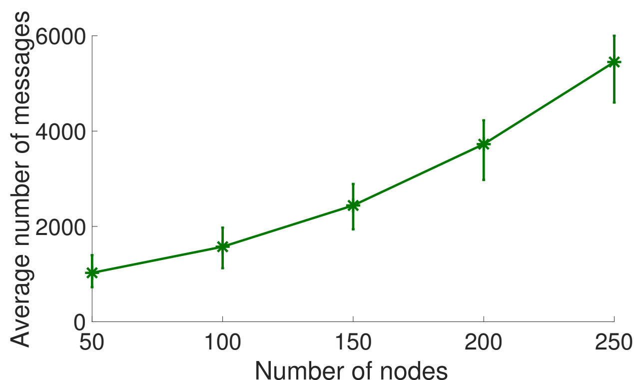


Figure 3.5 Message complexity of RobustPay$^+$

3.6 Conclusion

In this work, we investigated the robust payment routing protocol to resist payment transaction failures in PCNs. We first suggested a set of crucial design goals for payment routing, which are referred to as robustness, efficiency, distributedness and approximate optimization. Following these design goals, we presented a distributed robust payment routing protocol RobustPay$^+$ consisting of three stages: Payment Path Construction, HTLC Establishment and Payment Forwarding. For Payment Path Construction,

RobustPay^{+} achieved robustness by constructing two payment paths, where either payment path can fulfill the payment request. To guarantee approximate optimization, we formulated the Maximum Transaction Fee Minimization (MTFM) problem and presented a distributed 2-approximation algorithm RobustPay^{+}. Moreover, we modified the original HTLC protocol to provide efficiency and robustness and adapted it to the robust payment routing protocol. Extensive simulations demonstrated that RobustPay^{+} achieved outstanding success ratio and average acceptance value compared to baseline algorithms.

CHAPTER 4
COUNTER-COLLUSION SMART CONTRACTS FOR WATCHTOWERS IN PAYMENT CHANNEL NETWORKS

Payment channel networks (PCNs) are proposed to improve the cryptocurrency scalability by settling off-chain transactions. However, PCN introduces an undesirable assumption that a channel participant must stay online and be synchronized with the blockchain to defend against frauds. To alleviate this issue, watchtowers have been introduced, such that a hiring party can employ a watchtower to monitor the channel for fraud. However, a watchtower might profit from colluding with a cheating counterparty and fail to perform this job. Existing solutions either focus on heavy cryptographic techniques or require a large collateral. In this work [67], we leverage smart contracts through economic approaches to counter collusions for watchtowers in PCNs. This brings distrust between the watchtower and the counterparty, so that rational parties do not collude or cheat. We provide detailed analyses on the contracts and rigorously prove that the contracts are effective to counter collusions with minimal on-chain operations. In particular, a watchtower only needs to lock a small collateral, which incentivizes participation of watchtowers and users. We also provide an implementation of the contracts in Solidity and execute them on Ethereum to demonstrate the scalability and efficiency of the contracts.

4.1 Introduction

The past decade has seen a blooming of cryptocurrencies [14], e.g., Bitcoin [7] and Ethereum [13]. However, cryptocurrencies cannot scale for wide-spread use, due to high overhead and storage requirement [15]. For example, Bitcoin can only process up to 7 transactions per second (tps) [16], compared to over $47,000$ peak tps handled by Visa [17].

Payment channel networks (PCNs), e.g., Bitcoin's Lightning Network [18] and Ethereum's Raiden Network [19], have been proposed to tackle the scalability issues [18]. PCNs can process instant and less valuable payments without involving slow and expensive blockchain transactions [20, 25, 31]. However, an essential assumption in PCN is that channel funds can be secure only if the channel participant stays online. A channel participant risks losing payments if it goes offline, since the other channel participant on this channel can request to close the channel by publishing an outdated fraud channel state proof (CSP). Meanwhile, opening and closing channels are expensive transactions on the blockchain. Thus, the channel participants want to avoid performing these operations frequently. Watchtowers have long been considered crucial for squashing fraud in PCNs [68]. The concept of watchtower was first proposed in the Lightning whitepaper [68]. Recently, Bitcoin Wallet Electrum has released its watchtower implementation [69].

Lightning Labs [70] and Blockstream [71] also announced their plans on the implementations in the Lightning Network Daemon (lnd) [47] and c-lightning [42], which are the main Lightning clients.

Watchtowers [72–75] are monitoring services that are always online and able to monitor the blockchain as shown in Figure 4.1. A hiring party (*i.e.,* a channel participant) can hire a watchtower and go offline. The watchtower can be paid either for each detected fraud or simply for monitoring. Since the PCN protocol punishes detected frauds, channel participants rarely cheat. Hence, it is more reasonable to pay a watchtower based on its actual cost of watching, which is proportional to the amount of stored CSP data. On the other hand, a watchtower needs to prove to the hiring party that it has been indeed performing the channel monitoring job. In our construction, the hiring party sends the latest CSP to the watchtower after each transaction. The watchtower, in return, needs to provide the hiring party with a proof that it has been monitoring the blockchain and stored this CSP. This proof-scheme makes a payer-transaction scheme palatable to channel participants. However, a rational watchtower might profit from colluding with a counterparty (*i.e.*, the other channel participant on this monitored channel). For example, the counterparty can bribe the watchtower and request to close the channel by publishing a fraud CSP. As a result, the rational watchtower would fail to monitor the channel and not respond with the latest CSP.

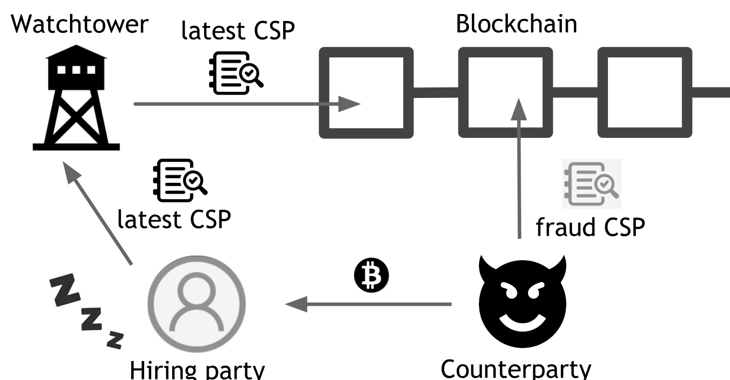


Figure 4.1 Illustration of a watchtower. The watchtower monitors the channel for the offline hiring party. When the counterparty requests to close the channel with a fraud CSP, the watchtowers responds with the latest CSP.

In this chapter, we tackle this collusion problem by addressing its root cause through economic approaches, instead of resisting collusions via heavy cryptographic methods. *The main contributions of this chapter are*:

- To the best of our knowledge, we are the first to study the collusion problem for watchtowers in PCNs from a game theatrical perspective, where a watchtower could profit from coordinating with the counterparty of the monitored channel and not performing the monitoring job.

- To tackle this collusion problem, we design three smart contracts, Watchtower contract, Collusion contract, and Betrayal contract, for scenarios where a hiring party outsources a channel monitoring job to a watchtower.
- We provide detailed analyses of the smart contracts. Specifically, we rigorously prove that there is a unique sequential equilibrium, *i.e.*, the rational parties will never collude or cheat with the existence of all the three contracts, even if they are allowed. Thus, the rational parties will never execute the Collusion or Betrayal contracts and are involved in minimal on-chain operations. In addition, the Watchtower contract is compilable with the PCN protocol and does not require on-chain operations.
- We provide a proof-of-concept implementation of the smart contracts in Solidity and execute them on Ethereum. It demonstrates that even if the contracts happen to be executed, the financial cost is very low.

The remainder of the chapter is organized as follows. In Section 4.2, we provide a brief literature review of the related work. In Section 4.3, we present the background in PCNs and relevant concepts in game theory. In Section 4.4, we formally describe the adversary model and give the necessary assumptions. In Section 4.5, we design the Watchtower contract and conduct detailed game theoretical analysis. In Section 4.6, we design the Collusion contract and conduct detailed game theoretical analysis. In Sections 4.7, we design the Betrayal contract and conduct detailed game theoretical analysis. In Sections 4.8, we conduct detailed game theoretical analysis of the full game induced by all the three contracts. In Section 4.9, we implement the smart contracts in Solidity and execute them on Ethereum. In Section 4.10, we conclude this chapter.

4.2 Related Work

The hazard of execution fork attacks against offline parties has raised a lot of attentions in the off-chain scaling community [68–71], who have proposed several mitigations thus far. Existing approaches enable the hiring parties (i.e., channel participants) to employ watchtowers [23] to help defend against execution forks on their behalf. Dryja *et al.* proposed Monitor [76], which requires $O(N)$ storage, where N is the number of off-chain transactions that have occurred within the channel. Osuntokunn *et al.* designed Watchtower [72], which improves Monitor's efficiency, but cannot be deployed without consensus rule changes in the Bitcoin network. Both proposals suffer from the drawback that if the hired watchtower fails, there is little recourse for the hiring party, since the protocols do not provide evidence about the employment.

Pisa [73] and Outpost [74] provide the hiring parties with publicly verifiable cryptographic evidence in case the watchtower fails, which can be used to penalize the watchtower. However, penalty does not

guarantee honest behaviors, i.e., responding upon fraud is not the watchtower's dominant strategy in these approaches. Cerberus [75] presented a watchtower extension of the Bitcon Lightning [36] channels, which needs to lock a large collateral more than the amount of the channel funds to resist against bribing. In this work, we address the collusion problem through economic approaches to avoid heavy cryptographic solutions and guarantee security against collusion with a small reasonable amount of deposit from watchtowers.

4.3 Background and Game Model

In this section, we introduce the background in payment channel networks and present relevant concepts in game theory.

4.3.1 Payment Channel Network (PCN)

To overcome the scalability issue, payment channels networks (PCNs) have been proposed to eliminate the need to commit each transaction on the blockchain [23]. A payment channel is protected by multi-signature smart contracts, which ensure validity, nonequivocality and non-repudiation. Two parties open a payment channel by each depositing a certain amount into a joint account and adding this opening transaction (*topen*) to the blockchain. Once the channel is opened, both parties exchange signed commitment transactions (*ctx*s) between each other. Now each signed *ctx* between them is essentially a distribution of *topen*'s funds agreed upon by both parties. When the channel closes, a closing transaction will be broadcast to the blockchain and will send funds to each party according to the latest *ctx* (referred to as *latest_ctx*).

If one party goes offline permanently, the counterparty can sign and broadcast *latest_ctx* to the blockchain to close the channel unilaterally. This brings distrust between the two parties. Because a dishonest party can broadcast a more favorable outdated fraud *ctx* (referred to as *previous_ctx*), since each party could store a stream of *previous_ctx*s backtracking to the channel opening. PCNs resolve this potential cheating by allowing *latest_ctx* to be exchanged with a revocation key that can allocate the entire channel funds to the victim party. If one party requests to close the channel unilaterally, it has to wait for a timelock, which gives the other party a time window to detect a fraud *previous_ctx* and raise a dispute on the blockchain. The dispute will be settled by the miners on the blockchain. Thus, an essential assumption is that a party has to be online at least once every timeblock to construct the corresponding *jtx* and broadcast it.

4.3.2 Watchtower

Watchtowers are service providers that are always online and able to monitor the blockchain. To go offline, a hiring party (*i.e.*, a channel participant) outsources the channel monitoring job to a watchtower and gives the latest *channel state proof* (CSP), a [ctx_txid_prefix, $ejtx$] pair, to the watchtower. The watchtower is expected to respond with the latest CSP, when the counterparty (*i.e.*, the other channel participant on the monitored channel) requests to close the channel unilaterally by publishing a fraud CSP (*i.e.*, an outdated $previous_ctx$).

The hiring party gives the latest CSP to the watchtower, when a new channel update is agreed upon. The watchtower stores a CSP map, where the keys are ctx_txid_prefixs and the values are $ejtx$s. Then, it watches the blockchain for any transaction whose $txid_prefix$ matches any of the keys in this map. If the watchtower finds a match, it extracts the $txid_suffix$ from the blockchain $txid$, and uses this ctx_txid to decrypt the corresponding $ejtx$ from its map to get the raw jtx, which is already signed by the corresponding hiring party of that channel. The watchtower then broadcasts this jtx to penalize the corresponding counterparty.

4.3.3 Smart Contract

Smart contracts are machinery that can be enforced on the blockchain [77]. In other words, a smart contract is a piece of program stored and executed on the blockchain. Smart contracts capture the logic of contractual clauses between parties and are executed when certain events are triggers. In addition, a smart contract can maintain funds and store the code that decides the flow of the funds in the contract account. Smart contracts are executed by the consensus peers, and the correctness of execution is guaranteed by the consensus protocol. Ideally, smart contracts can be considered to be executed by a global machine that will faithfully execute every instruction.

4.3.4 Games and Strategies

In this work, we design games in extensive form with imperfect information. Imperfect information means that the players has partial or no knowledge of the actions taken by others; while perfect information has a strong assumption that players knows every action of others. Blockchain is anonymous and global, and one can maintain several accounts to perform the actions. Therefore, we model the games as imperfect-information, which are more realistic and allows a wider scope of analysis. Informally speaking, an imperfect-information game in extensive form is a tree in terms of graph theory, where each node represents one player's choice, each edge represents a possible action, and the leaves represent final outcomes over which each player has a utility function. Formally, a *finite imperfect-information game* is

defined as follows [78].

Definition 1 (Finite imperfect-information game). *A finite imperfect-information game (in extensive form) is a tuple* $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{V}, \mathcal{T}, \chi, \rho, \sigma, u, \mathcal{I})$*, where:*

- $\mathcal{N}$ *is a set of* n *players.*
- $\mathcal{A}$ *is a single set of actions.*
- $\mathcal{V}$ *is a set of nonterminal choice nodes.*
- $\mathcal{T}$ *is a set of terminal nodes, disjoint from* $\mathcal{V}$*.*
- χ*:* $\mathcal{V} \to 2^{\mathcal{A}}$ *is the action function, which assigns to each choice node a set of possible actions.*
- ρ*:* $\mathcal{V} \to \mathcal{N}$ *is the player function, which assigns to each nonterminal node a player* $i \in \mathcal{N}$ *who chooses an action at that node.*
- σ*:* $\mathcal{V} \times \mathcal{A} \to \mathcal{V} \cup \mathcal{T}$ *is the successor function, which maps a choice node and an action to a new choice node or terminal node such that* $\forall v_i, v_j \in \mathcal{V}$ *and* $a_i, a_j \in \mathcal{A}$*, if* $\sigma(v_i, a_i) = \sigma(v_j, a_j)$ *then* $v_i = v_j$ *and* $a_i = a_j$*.*
- $u = (u_1, \ldots, u_n)$ *is a vector of utility functions, where* $u_i : \mathcal{T} \to \mathbb{R}$ *is player* i*'s utility function on* $\mathcal{T}$*.*
- $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_n)$*, where* $\mathcal{I}_i = (\mathcal{I}_{i,1}, \ldots, \mathcal{I}_{i,k_i})$ *is an equivalence relation on* $\{v \in \mathcal{V} : \rho(v) = i\}$ *(i.e., a partition of* i*'s choice nodes) subject to* $\chi(v) = \chi(v')$ *and* $\rho(v) = \rho(v')$*, whenever there is a* j *such that* $v, v' \in I_{i,j}$*. Intuitively, if two choice nodes are in the same information set* $\mathcal{I}_{i,j}$*, player* i *cannot differentiate them.*

A *strategy* of a player determines the action that the player will take in the game. In an imperfect-information game, the player i cannot differentiate two choice nodes v and v' in the same information set $\mathcal{I}_{i,j} \in \mathcal{I}_i$. In other words, the set of possible actions assigned to nodes in the same information set is the same, *i.e.*, $\forall v, v' \in \mathcal{I}_{i,j}$, $\chi(v) = \chi(v')$. Therefore, player i selects one of the available actions at each $\mathcal{I}_{i,j}$. Formally, a strategy in an imperfect-information game is defined as follows.

Definition 2 (Strategy). *Let* $\mathcal{G} = (\mathcal{N}, \mathcal{A}, \mathcal{V}, \mathcal{T}, \chi, \rho, \sigma, u, \mathcal{I})$ *be an imperfect-information extensive-form game. A strategy of a player* i *is a tuple of actions* $s_i = (a_{i,1}, \ldots, a_{i,k_i})$*, where* $a_{i,j} \in \chi(v), v \in \mathcal{I}_{i,j} \in \mathcal{I}_i$ *and* $\forall v, v' \in \mathcal{I}_{i,j}, \chi(v) = \chi(v')$*.*

Definition 3 (Strategy profile). *A strategy profile of a game is a tuple of all players' strategies* $s = (s_1, \ldots, s_n)$*, where* s_i *is one of the player* i*'s strategies.*

Definition 4 (Sequential equilibrium). *A strategy profile s is a sequential equilibrium of an extensive-form game $\mathcal{G}$, if there exist probability distributions $\mu(\mathcal{I}_{i,j})$, $\forall \mathcal{I}_{i,j} \in \mathcal{I}_i$ such that:*

1. *$(s, \mu) = \lim_{n \to \infty} (s_n, \mu_n)$ for some sequence $(s_1, \mu_1), (s_2, \mu_2), \ldots$, where s_n is fully mixed, and μ_n is consistent with s_n.*

2. *For any information set $\mathcal{I}_{i,j} \in \mathcal{I}_i$, and any strategy $s'_i \neq s_i$, we have $u_i(s|\mathcal{I}_{i,j}, \mu(\mathcal{I}_{i,j})) \geq u_i((s', s_{-i})|\mathcal{I}_{i,j}, \mu(\mathcal{I}_{i,j}))$.*

An example of game tree is shown in Figure 4.2. In this game, there are 2 players $\mathcal{N} = \{1, 2\}$ and a set of actions $\mathcal{A} = \{A, \ldots, H\}$. The circle nodes denote the nonterminal choice nodes $\mathcal{V}$. The rectangle nodes denote the terminal choice nodes $\mathcal{T}$. The utilities (payoffs) of players 1 and 2 are u_1 and u_2 at the bottom of the terminal nodes. An information set is denoted by the nodes within the dashed rectangles. There are 4 information sets: $\mathcal{I}_{1,1} = \{v_1\}, \mathcal{I}_{2,1} = \{v_2, v_3\}, \mathcal{I}_{2,1} = \{v_4, v_5\}$, and $\mathcal{I}_{2,2} = \{v_6, v_7\}$. The sequential equilibrium is a strategy profile $s = (s_1, s_2)$, where $s_1 = (C, G, G)$ and $s_2 = (D)$. The strategy s_1 indicates that player 1 plays C at $\mathcal{I}_{1,1}$, plays D at $\mathcal{I}_{1,2}$, and plays D at $\mathcal{I}_{1,3}$.

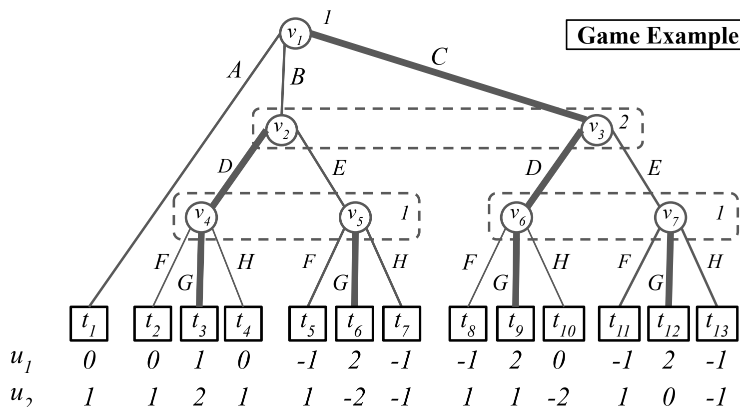


Figure 4.2 Example of game tree. Bold edges indicate the actions in the unique sequential equilibrium.

4.3.5 Monetary Variables

Below are the monetary variables that will be used in the contracts (listed in alphabetic order). They are all non-negative.

- b: the bribe that the counterparty agrees to pay the watchtower in the collusion agreement (Collusion contract).
- c: the watchtower's cost for monitoring the channel.

- f: the verification fee paid to the miners for settling a dispute on the blockchain.
- d_w: the amount that a watchtower agrees to deposit to get the channel monitoring job.
- d_c: the fund that the counterparty owns on the channel.
- d_t : the amount that both the counterparty and the watchtower agree to deposit in the collusion agreement (Collusion contract).
- w_h: the amount that the hiring party agrees to pay to a watchtower for monitoring the payment channel.
- w_c: the extra amount that the counterparty can earn by publishing a fraud channel state proof (CSP).

The following relations hold for obvious reasons:

- $w_h > c$: the watchtower does not accept under-paid jobs.
- $w_c > b$: the counterparty does not pay more bribe than its gain from the collusion.

The monetary variables w_h and d_w can be set by the hiring party in the Watchtower contract; the monetary variables b and d_t can be set by the counterparty in the Collusion contract.

4.4 System Model and Assumptions

In this section, we introduce the necessary assumptions in the system.

4.4.1 Cryptographic Assumptions

We make the typical cryptographic assumptions. We assume that there are secure communication channels, cryptographically secure hash functions, signatures, and encryption schemes. In addition, all parties (watchtowers, channel participants, external adversaries, etc) are computationally bounded.

4.4.2 Blockchain Assumptions

We assume a perfect blockchain, where both persistence and liveness hold [79]. Specifically, we assume that if a valid transaction is propagated in the blockchain, it cannot be censored and will be included in the "permanent" part of the blockchain immediately. Additionally, we assume that any channel participant can go offline (intentionally or unintentionally) for a (long) time period of up to T_{off}. Furthermore, we assume that watchtowers are resilient against DoS attacks and always online. This assumption is realistic, since watchtowers are required to deposit to participate in the system and thus will invest in the anti-DoS protection [75].

4.4.3 Adversary Model

We consider an honest hiring party who outsources the job of monitoring the payment channel. For the hiring party, the goal is to get its channel monitored while minimizing the cost. The watchtower is unreliable and could respond with a false channel state proof (CSP) for the outsourced monitoring job. Note that in this work, we do not distinguish intentional and unintentional faults, because it is difficult to collect evidence. We assume the watchtower and the counterparty are physically isolated, because the watchtowers are usually maintained by the leading nodes in PCNs. We also assume each party is an individually rational adversary. Being rational means that a party always acts in a way that maximizes its payoff and is capable of thinking through all possible outcomes and choosing strategies that will result in the best possible outcome.

4.5 The Watchtower Contract

In this section, we present the Watchtower contract and analyze the game induced by the Watchtower contract.

4.5.1 The Contract

The Watchtower contract is an outsourcing contract signed by a hiring party (H) and a watchtower (W). The contract allows H to employ W to monitor the channel. W should watch the channel state and respond with the latest channel state proof (CSP) p timely, if the counterparty (C) requests to close the channel unilaterally. At a high level, it aims to incentivize honest behaviors by asking W to deposit beforehand. If the watchtower behaves honestly, the deposit will be refunded; if the watchtower cheats and is detected, the deposit will be taken by H. The contract is presented below:

1. The contract is between H and W.
2. H agrees to provide the latest CSP p to W and deposit w_h to the contract for W to monitor its channel.
3. W agrees to monitor the channel and deposit d_w to contract.
4. If either H or W fails to sign the contract before the deadline T_1, the contract will terminate, and any w_h and/or d_w held by the contract will be refunded.
5. When C requests to close the channel at T_2:
   (a) If W sends p before $T_2 + \delta$, then $w_h + d_w$ is paid to W.
   (b) Otherwise, $w_h + d_w$ is paid to H.

6. When C does not request to close the channel by T_3:

   (a) If W provides p to H to prove that it has been indeed monitoring the channel, then $w_h + d_w$ is paid to W.

   (b) Otherwise, $w_h + d_w$ is paid to H.

The Watchtower functionality that allows W to deposit has recently be implemented as an extension called Cerberus [80] of the Bitcoin Lightning Network [36]. Thus, it does not require on-chain operations that cost transaction and execution fees. Cerberus counters collusions by locking a large deposit from W. In this work, we provide a contract-based solution, such that only a small deposit is required from W to counter collusions.

4.5.2 The Game and Analysis

The game included by the Watchtower contract is shown in Game 1 in Figure 4.3. In this game, the players are $\mathcal{N} = \{C, W\}$. Although the contract involves H, H can be eliminated from the game, because it has only one deterministic strategy, *i.e.*, outsourcing the monitoring job to W. W and C can communicate and send a fraud CSP $p' \neq p$ to close the channel. The action set is $\mathcal{A} = \{p, p', other\}$. The first two actions represent that the player sends p or p' before the deadline. The last means any other actions the player may take. Game 1 has 2 information sets: $I_{c,1} = \{v_1\}$, and $I_{w,1} = \{v_2, v_3, v_4\}$. The game tree captures $\mathcal{V}, \mathcal{T}, \chi, \rho$, and σ. The utilities (payoffs) are listed below the terminal nodes.

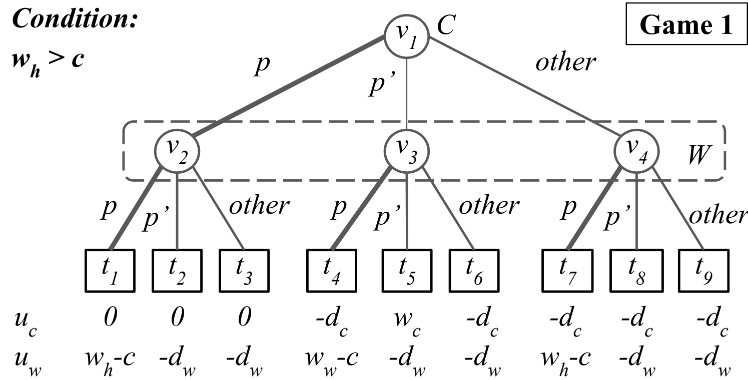


Figure 4.3 Game induced by the Watchtower contract

Next, we analyze Game 1 and show that if $w_h > c$, both players will always send p and Game 1 will always terminate at t_1. Formally, we have:

Theorem 5. If $w_h > c$, Game 1 has a unique sequential equilibrium $(s_c^1, s_w^1) = ((p), (p))$. According to the equilibrium, C will send the latest CSP p, and W will respond with p.

Proof. We prove this by showing that the only reachable outcome of Game 1 is t_1. The intuition is that sending p always leads to the highest payoff for both players. At W's choice node v_2, W's payoff is $w_h - c$, $-d_w$, and $-d_w$, if the game ends at t_1, t_2, and t_3, respectively. Since $w_h - c > -d_w$, W will send p at v_2 to reach t_1. Similarly, W will send p at v_3 and v_4. Hence, W will always send p, no matter what C's action is. Inferring this, C knows that the only reachable nodes are t_1, t_4 and t_7. Since t_1 has a higher payoff than t_4 and t_7 for C, C will always send p. Thus, Game 1 has a unique sequential equilibrium that both players will send p. ■

4.6 The Collusion Contract

The Watchtower contract works, because there is no trust between the counterparty (C) and the watchtower (W) to collude. However, C and W can make credible and enforceable promises to collude by both signing a collusion contract. In this section, we present the Collusion contract and analyze the game induced by both the Watchtower and Collusion contracts.

4.6.1 The Contract

The Collusion contract is a collusion contract signed by C and W. The contract allows C to bribe W for collusion, *i.e.*, W should coordinate with C, when C requests to close this channel unilaterally with a fraud channel state proof (CSP) p'. At a high level, it aims to incentivize C and W to send a fraud CSP by redistributing the profit between them and punishing those who deviate from the collusion. The trust between C and W is built by asking both C and W to deposit beforehand. Either C or W, who deviates from collusion, will be punished by losing the deposit. The contract is presented below:

1. The contract is between C and W.
2. C agrees to provide p' and deposit $d_t + b$ to the contract.
3. W agrees to collude with C and deposit d_t to the contract.
4. If either C or W fails to sign the contract before the deadline T_2, the contract will terminate, and any b and/or d_t held by the contract will be refunded.
5. When C requests to close the channel by sending p' at T_2:
   (a) If W sends p', d_t is refunded to C; $d_t + b$ is paid to W.
   (b) Otherwise, $2d_t + b$ is paid to C.
6. When C sends a CSP other than p' at T_2:

(a) If W sends p', $2d_t + b$ is paid to W.

(b) Otherwise, $d_t + b$ is refunded to C; d_t is refunded to W.

7. When C does not request to close the channel by T_3: $d_t + b$ is refunded to C, and d_t is refunded to W.

The Collusion contract must be signed before T_2, so that C and W can trust each other to collude without any risk. C needs to provide p' before T_2. In Clause $5(b)(ii)$, when both C and W deviate from collusion, neither of them is punished.

In this contract, C agrees to pay a bribe b to motivate W to collude. Both C and W lock a deposit d_t to ensure that 1) the deviating player always gets a lower payoff than not deviating from the collusion; 2) the player following the collusion always gets a higher payoff than not following the collusion.

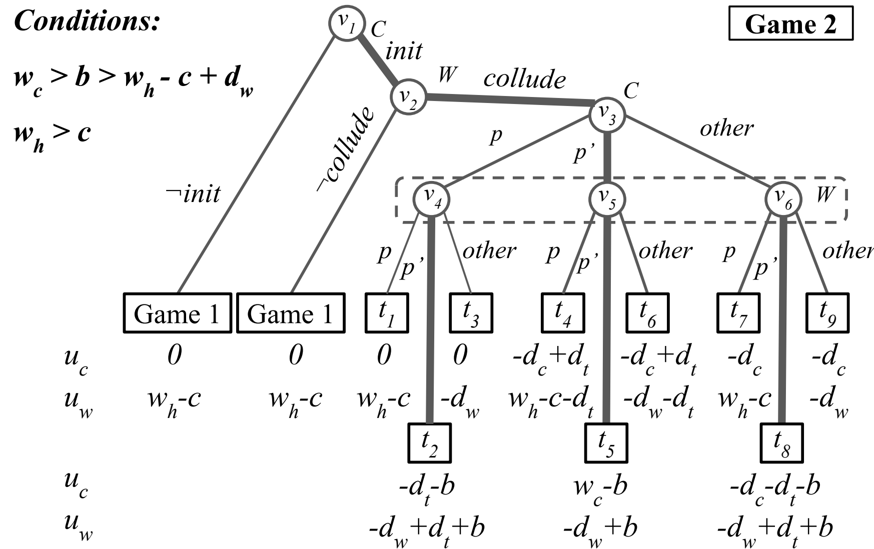


Figure 4.4 Game induced by the Watchtower contract and the Collusion contract

4.6.2 The Game and Analysis

The game induced by the Watchtower and Collusion contracts is shown in Game 2 in Figure 4.4. In this game, the players are $\mathcal{N} = \{C, W\}$. The action set is $\mathcal{A} = \{\neg init, init, \neg collude, collude, p, p', other\}$. The first two actions represent that C has the option of to initiate the collusion or not. The followed two actions represent that W has the option of to collude with C or not. Game 2 has 4 information sets, *i.e.*, $\mathcal{I}_{c,1} = \{v_1\}$, $\mathcal{I}_{c,2} = \{v_3\}$ (belonging to C), and $\mathcal{I}_{w,1} = \{v_2\}$, $\mathcal{I}_{w,2} = \{v_4, v_5, v_6\}$ (belonging to W). The payoffs (utilities) are listed below the terminal nodes.

If C plays $\neg init$ or W plays $\neg collude$, they will end up with playing Game 1 (in Figure 4.3), because the only activated contract is the Watchtower contract. We will not show the analysis of these two

branches here, as it is exactly the same as in Section 4.5.2 (subject to relabeling of nodes). Because only one terminal node t_1 is reachable in Game 1, we can replace each branch with a single terminal node *Game* 1, and it has the same payoffs as t_1 in Game 1. If the Collusion contract in initiated and signed, the payoffs are decided jointly by both the Watchtower and Collusion contracts.

Next, we analyze Game 2 and show that if $w_c > b > w_h - c + d_w$ and $w_h > c$, both players will collude and send p' and Game 2 will always terminate at t_5. Formally, we have:

Theorem 6. *If $w_c > b > w_h - c + d_w$ and $w_h > c$, Game 2 has a unique sequential equilibrium (s_c^2, s_w^2):*

$$\begin{cases} s_c^2 = & (init, s_c^1, s_c^1, p'), \\ s_w^2 = & (s_w^1, collude, s_w^1, p'). \end{cases}$$

According to the equilibrium, C will initiate the collusion, W will agree to collude, C will send the fraud CSP p', and W will respond with p'.

Proof. We prove this by showing that the only reachable outcome of Game 2 is t_5. The intuition is that colluding and sending p' always leads to the highest payoff for both players. At W's choice node v_4, the payoff of W is $w_h - c$, $-d_w + d_t + b$, and $-d_w$, if the game ends at t_1, t_2, and t_3, respectively. Since $b > w_h - c + d_w$, W will play p' at v_4 to reach t_2. Similarly, W will play p' at v_5 and v_6. Hence, W will always play p', no matter what C's action is. Inferring this, C knows that the only reachable nodes are t_2, t_5 and t_8. Since t_5 has the highest payoff for C, C will always play p' to reach t_5. Inferring this, W will always *collude* at v_2, because t_5 has a higher payoff than *Game* 1 for W. Inferring this, C will always *init* the collusion at v_1. Thus, Game 2 has a unique sequential equilibrium that both players will collude and send p'. ■

4.7 The Betrayal Contract

The Collusion contract works, because it brings trust between the counterparty (C) and the watchtower (W). However, the hiring party (H) can create distrust between C and W by incentivizing W to betray the collusion and behave honestly. In this section, we present the Betrayal contract and analyze the sub-game induced by the Watchtower and Betrayal contracts.

4.7.1 Challenge

The main challenge to address the collusion problem is to get out of the counter/counter-back loop. H could always counter the Collusion contract by providing a higher reward w_h to W, which makes the collusion less profitable and changes the equilibrium. However, C could always counter-back by providing a higher bribe b, so that the collusion becomes more profitable again. This counter/counter-back loop could fall into the endless competition between C and H until one runs out of funds. The Betrayal contract

focuses on the root cause and tackles the collusion problem by bringing distrust between C and W. Specifically, H offers W the total penalty from C, if W reports the collusion and respond correctly. The goal of the Betrayal contract is to incentivize W to report the collusion, but not necessarily to betray the collusion.

4.7.2 The Contract

The Betrayal contract is a report contract signed by H and W. The contract allows W to report collusion and respond upon fraud. W could report the collusion to H and respond with a channel state proof (CSP) p^*, when C tries to bribes W to collude and close this channel unilaterally. At a high level, it tries to incentivize W to report the collusion by paying C's deposit d_c to W. The contract is presented below:

1. The contract is signed between H and W, both of whom have signed the Watchtower contract.
2. H agrees to deposit d_c for W to report the collusion.
3. W agrees to deposit f and report the collusion.
4. If either H or W fails to sign the contract before T_2, the contract will terminate, and any d_c or/and f held by the contract will be refunded.
5. If W reports a collusion before T_2:
   (a) If C sends a fraud CSP at T_2,
      i. If W responds with $p^* = p$, $d_c + f$ is paid to W.
      ii. Otherwise, d_c is paid to H.
   (b) Otherwise, f is paid to H.
6. If W does not report a collusion before T_2, then d_c is refunded to H, and f is refunded to W.

Note that W does not have to sign the Collusion contract to report a collusion. W can misreport a collusion, unintentionally or intentionally. W can respond with $p*$ anonymously from another address maintained by W, when C requests to close the channel. By providing the evidence to H in the Betrayal contract, W can be prove that it has performed the channel monitoring job.

4.7.3 The Sub-game and Analysis

Before analyzing the full game induced by the Watchtower, Collusion, and Betrayal contracts, we first present and analyze a sub-game induced by the Watchtower and Betrayal contracts. The sub-game illustrates the situation when the Collusion contract is not activated, because either the collusion coalition is not initiated by C or is rejected by W.

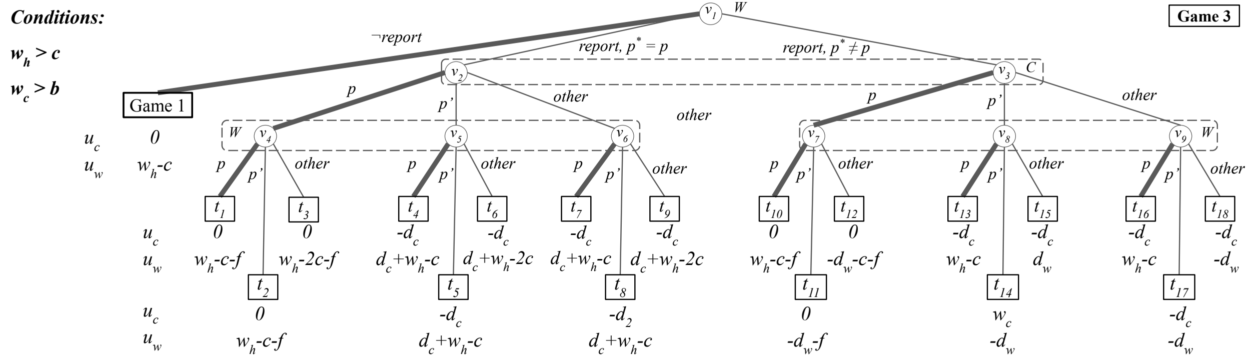


Figure 4.5 Sub-game induced by the Watchtower contract and the Betrayal contract

The sub-game included by the Watchtower and Betrayal contracts is shown in Game 3 in Figure 4.5. In this game, the players are $\mathcal{N} = \{C, W\}$. The action set is $\mathcal{A} = \{\neg report, report\ p* = p, report\ p* \neq p, p, p', other\}$. The first action means that the player has the option of not to report the collusion. The followed two actions represent that the player can report the collusion and send p^* before the deadline. Game 3 has 4 information sets, *i.e.*, $\mathcal{I}_{c,1} = \{v_2, v_3\}$ (belonging to C), $\mathcal{I}_{w,1} = \{v_1\}$, $\mathcal{I}_{w,2} = \{v_4, v_5, v_6\}$, and $\mathcal{I}_{w,3} = \{v_7, v_8, v_9\}$ (belonging to W). The payoffs (utilities) are listed below the terminal nodes.

If W chooses not to report the collusion, the only activated contract is the Watchtower contract and the branch is exactly the same as the game tree of Game 1. We will not show the analysis of this branch here, as it is exactly the same as in Section 4.5.2 (subject to relabeling of nodes). Because only one terminal node t_1 is reachable in Game 1, we can replace this branch with a single terminal node *Game* 1, and it has the same payoffs as t_1 in Game 1. If W decides to report, W can report the collusion and then send either $p^* = p$ (branch to v_2) or $p^* \neq p$ (branch to v_3). In both cases, the payoffs are decided jointly by the Watchtower and Betrayal contracts.

Next, we analyze Game 3 and show that if $w_h > c$ and $w_c > b$, W will not report collusion and both players will send p. Game 3 will terminate at t_1 in Game 1. Formally, we have:

Theorem 7. *If $w_h > c$ and $w_c > b$, then Game 3 has a unique sequential equilibrium (s_c^3, s_w^3), where*

$$\begin{cases} s_c^3 = & (p), \\ s_w^3 = & (\neg report, p, p, p). \end{cases}$$

According to this equilibrium, W will not report a collusion, C will send the latest CSP p, and W will respond with p.

Proof. We prove this by showing that the only reachable outcome of Game 3 is t_1 in Game 1. The intuition is that not reporting collusion and sending p always leads to the highest payoff for each player. At W's choice node v_7, the payoff of W is $w_h - c - f$, $-d_w - f$, and $-d_w - c - f$, if the game ends at t_{11}, t_{12}, and t_{13}, respectively. Since $w_h - c - f > -d_w - f$, W will play p at v_7 to reach t_{10}. Similarly, W will play p at v_8 and v_9. Hence, W will always play p, no matter what C's action is. Inferring this, C knows that the only reachable nodes are t_{10}, t_{13} and t_{16}. Since t_{10} has the highest payoff for C, C will play p at v_3 to reach t_{10}. Similarly, W will always play p at v_4, v_5, and v_6, no matter what C's action is; C will always play p at v_2 to reach t_1. Inferring this, W will always $\neg report$ at v_1, because *Game* 1 has a higher payoff than t_1 and t_{10} for W. Thus, Game 3 has a unique sequential equilibrium that W will not report collusion and both players will send p'. ■

4.8 The Full Game Induced by All Contracts

In this section, we present and analyze the full game induced by the Watchtower, Collusion contract, and Betrayal contracts.

4.8.1 The Game and Analysis

The full game included by the Watchtower, Collusion, and Betrayal contracts is shown in Game 4 in Figure 4.6. In this game, the players are $\mathcal{N} = \{C, W\}$. The action set is $\mathcal{A} = \{init, \neg init, collude, \neg collude, \neg report, report\ p* = p, report\ p* \neq p, p, p', other\}$. The first two action represent that the player has the option of to initiate the collusion or not. The followed two actions represent that the player has the option of to collude or not. Game 4 has 7 information sets, *i.e.*, $\mathcal{I}_{c,1} = \{v_1\}$, $\mathcal{I}_{c,2} = \{v_4, v_5, v_6\}$ (belonging to C), and $\mathcal{I}_{w,1} = \{v_2\}$, $\mathcal{I}_{w,2} = \{v_3\}$, $\mathcal{I}_{w,3} = \{v_7, v_8, v_9\}$, $\mathcal{I}_{w,4} = \{v_{10}, v_{11}, v_{12}\}$, $\mathcal{I}_{w,5} = \{v_{13}, v_{14}, v_{15}\}$ (belonging to W). The payoffs (utilities) are listed below the terminal nodes.

In this game, if the collusion is not initiated by C or is rejected by W, C and W will end up with playing Game 3, because there is no activated Collusion contract. If W agrees to collude with C and does not report to H, they will enter a subtree rooted at v_4, where the outcome is the same as Game 2, because there is no activated Betrayal contract. Otherwise, if W agrees to collude with C but also reports to H, they will enter subtrees rooted v_5 and v_6. The payoffs are decided jointly by the Watchtower, Collusion,

and Betrayal contracts.

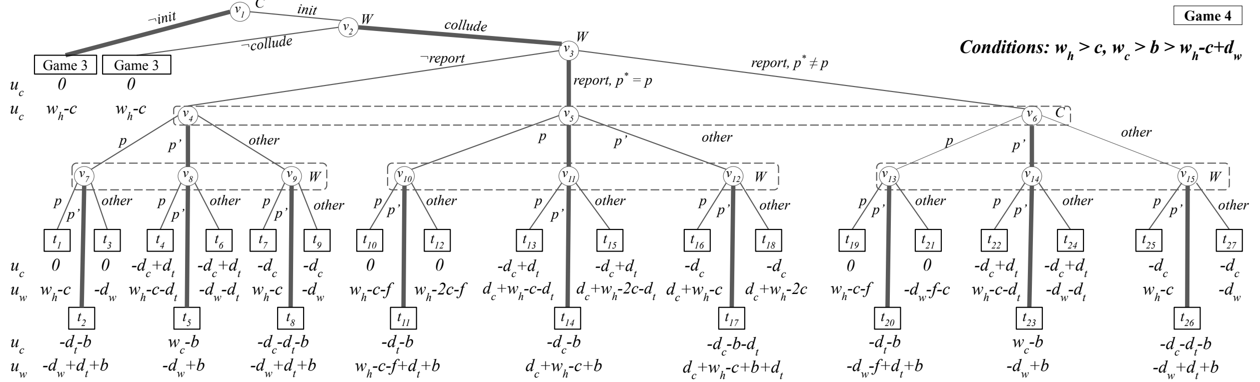


Figure 4.6 Full game induced by the Watchtower contract, the Collusion contract, and the Betrayal contract

Next, we analyze Game 3 and show that if $w_h > c$ and $w_c > b > w_h - c + d_w$, both players will not initiate or agree to collude, and they will always send p. Formally, we have:

Theorem 8. If $w_c > b > w_h - c + d_w$ and $w_h > c$, Game 4 has a unique sequential equilibrium (s_c^4, s_w^4):

$$\begin{cases} s_c^4 = & (\neg init, s_c^3, s_c^3, p') \\ s_w^4 = & (s_w^3, collude, s_w^3, report\ p^* = p', p', p', p'). \end{cases}$$

According to this, C will not initiate the collusion, W will not report a collusion, C will send p, and W will respond with p.

Proof. We prove it by backward induction. Let the strategy profile in the sequential equilibrium be $s = (s_c^4, s_w^4)$, where

$$\begin{cases} s_c^4 = & ([\phi_1(\neg init), \phi_2(init)], s_c^3, s_c^3, \\ & [\phi_3(p), \phi_4(p'), \phi_5(other)]) \\ s_w^4 = & (s_w^3, [\psi_1(\neg collude), \psi_2(collude)], s_w^3, \\ & [\psi_3(\neg report), \psi_4(report\ p^* = p), \psi_5(report\ p^* \neq p)], \\ & [\psi_6(p), \psi_7(p'), \psi_8(other)], [\psi_9(p), \psi_{10}(p'), \psi_{11}(other)], \\ & [\psi_{12}(p), \psi_{13}(p'), \psi_{14}(other)]), \end{cases} \tag{4.1}$$

where ϕ_i and ψ_j are probabilities that satisfy the following: $0 \leq \phi_i \leq 1, \phi_1 + \phi_2 = 1, \phi_3 + \phi_4 + \phi_5 = 1$, $0 \leq \psi_j \leq 1, \psi_1 + \psi_2 = 1, \psi_3 + \psi_4 + \psi_5 = 1, \psi_6 + \psi_7 + \psi_8 = 1, \psi_9 + \psi_{10} + \psi_{11} = 1, \psi_{12} + \psi_{13} + \psi_{14} = 1$.

At the information set $\mathcal{I}_{w,3} = \{v_7, v_8, v_9\}$, W tries to maximize its expected payoff:

$$u_w(s; \mathcal{I}_{w,3}) = \phi_3 u_w(s; v_7) + \phi_4 u_w(s; v_8) + \phi_5 u_w(s; v_9), \tag{4.2}$$

where

$$\begin{cases} u_w(s; v_7) &= \psi_6 u_w(t_1) + \psi_7 u_w(t_2) + \psi_8 u_w(t_3), \\ u_w(s; v_8) &= \psi_6 u_w(t_4) + \psi_7 u_w(t_5) + \psi_8 u_w(t_6), \\ u_w(s; v_9) &= \psi_6 u_w(t_7) + \psi_7 u_w(t_8) + \psi_8 u_w(t_9). \end{cases}$$

It can be inferred that $\psi_6 = 0, \psi_7 = 1, \psi_8 = 0$, because the following holds when $w_h > c$ and $w_c > b > w_h - c + d_w$: $u_w(t_2) > u_w(t_1) > u_w(t_3)$; $u_w(t_5) > u_w(t_4) > u_w(t_6)$; $u_w(t_8) > u_w(t_7) > u_w(t_9)$. Thus, when $\psi_6 = 0, \psi_7 = 1, \psi_8 = 0$, it maximizes the expected payoff for W at $\mathcal{I}_{w,3}$. Therefore, we have $u_w(s; v_7) = u_w(t_2)$, $u_w(s; v_8) = u_w(t_5)$ and $u_w(s; v_9) = u_w(t_8)$. Substituting the above into Eq. (4.2), we have $u_w(s; \mathcal{I}_{w,3}) = \phi_3 u_w(t_2) + \phi_4 u_w(t_5) + \phi_5 u_w(t_8)$.

Similarly, at $\mathcal{I}_{w,4} = \{v_{10}, v_{11}, v_{12}\}$, when $\psi_9 = 0, \psi_{10} = 1, \psi_{11} = 0$, it maximizes the expected payoff for W. Also, at $\mathcal{I}_{w,5} = \{v_{13}, v_{14}, v_{15}\}$, when $\psi_{12} = 0, \psi_{13} = 1, \psi_{14} = 0$, it maximizes the expected payoff for W. Therefore, we have

$$\begin{cases} u_w(s; \mathcal{I}_{w,3}) & = \phi_3 u_w(t_2) + \phi_4 u_w(t_5) + \phi_5 u_w(t_8), \\ u_w(s; \mathcal{I}_{w,4}) & = \phi_3 u_w(t_{11}) + \phi_4 u_w(t_{14}) + \phi_5 u_w(t_{17}), \\ u_w(s; \mathcal{I}_{w,5}) & = \phi_3 u_w(t_{20}) + \phi_4 u_w(t_{23}) + \phi_5 u_w(t_{26}). \end{cases}$$

Moving backward, at $\mathcal{I}_{c,2} = \{v_4, v_5, v_6\}$, C tries to maximize its expected payoff, which is $u_c(s; \mathcal{I}_{c,2}) = \psi_3 u_c(s; v_4) + \psi_4 u_c(s; v_5) + \psi_5 u_c(s; v_6)$, where

$$\begin{cases} u_c(s; v_4) & = \phi_3(\psi_6 u_c(t_1) + \psi_7 u_c(t_2) + \psi_8 u_c(t_3)) \\ & \quad +\phi_4(\psi_9 u_c(t_4) + \psi_{10} u_c(t_5) + \psi_{11} u_c(t_6)) \\ & \quad +\phi_5(\psi_{12} u_c(t_7) + \psi_{13} u_c(t_8) + \psi_{14} u_c(t_9)), \\ u_c(s; v_5) & = \phi_3(\psi_6 u_c(t_{10}) + \psi_7 u_c(t_{11}) + \psi_8 u_c(t_{12})) \\ & \quad +\phi_4(\psi_9 u_c(t_{13}) + \psi_{10} u_c(t_{14}) + \psi_{11} u_c(t_{15})) \\ & \quad +\phi_5(\psi_{12} u_c(t_{16}) + \psi_{13} u_c(t_{17}) + \psi_{14} u_c(t_{18})), \\ u_c(s; v_6) & = \phi_3(\psi_6 u_c(t_{19}) + \psi_7 u_c(t_{20}) + \psi_8 u_c(t_{21})) \\ & \quad +\phi_4(\psi_9 u_c(t_{22}) + \psi_{10} u_c(t_{23}) + \psi_{11} u_c(t_{24})) \\ & \quad +\phi_5(\psi_{12} u_c(t_{25}) + \psi_{13} u_c(t_{26}) + \psi_{14} u_c(t_{27})). \end{cases}$$

By substituting $\psi_6 = 0, \psi_7 = 1, \psi_8 = 0$; $\psi_9 = 0, \psi_{10} = 1, \psi_{11} = 0$ and $\psi_{12} = 0, \psi_{13} = 1, \psi_{14} = 0$, we have

$$\begin{cases} u_c(s; v_4) & = \phi_3 u_c(t_2) + \phi_4 u_c(t_5) + \phi_5 u_c(t_8), \\ u_c(s; v_5) & = \phi_3 u_c(t_{11}) + \phi_4 u_c(t_{14}) + \phi_5 u_c(t_{17}), \\ u_c(s; v_6) & = \phi_3 u_c(t_{20}) + \phi_4 u_c(t_{23}) + \phi_5 u_c(t_{26}). \end{cases}$$

Because $u_c(t_5) > u_c(t_2) > u_c(t_8)$, it maximizes $u_c(s; v_4)$, when $\phi_3 = 0, \phi_4 = 1, \phi_5 = 0$. Similarly, it maximizes $u_c(s; v_5)$ and $u_c(s; v_6)$, when $\phi_3 = 0, \phi_4 = 1, \phi_5 = 0$. Thus, C will always choose $\phi_3 = 0, \phi_4 = 1, \phi_5 = 0$ to maximize $u_c(s; \mathcal{I}_{c,2})$. Thus, we have $u_c(s; v_4) = u_c(t_5)$, $u_c(s; v_5) = u_c(t_{14})$ and $u_c(s; v_6) = u_c(t_{23})$.

Moving backward, at $\mathcal{I}_{w,2} = \{v_3\}$, W tries to maximize its expected payoff, which is

$$u_w(s; \mathcal{I}_{w,2}) = \psi_3 u_w(t_5) + \psi_4 u_w(t_{14}) + \psi_5 u_w(t_{23}). \tag{4.3}$$

Because $u_w(t_{14}) > u_w(t_5) = u_w(t_{23})$, W will always choose $\psi_3 = 0, \psi_4 = 1, \psi_5 = 0$ to maximize $u_w(s; v_3)$. Substituting the above into Eq. (4.3), we have $u_w(s; \mathcal{I}_{w,2}) = u_w(t_{14})$.

Moving backward, at $\mathcal{I}_{w,1} = \{v_2\}$, W tries to maximize its expected payoff:
$u_w(s; \mathcal{I}_{w,1}) = u_w(s; v_2) = \psi_1 u_w(Game\ 3) + \psi_2 u_w(t_{14})$. Because $u_w(t_{14}) > u_w(Game\ 3)$, W will always choose $\psi_1 = 0, \psi_2 = 1$ to maximize $u_w(s; v_2)$. Thus, we have $u_w(s; \mathcal{I}_{w,1}) = u_w(t_{14})$.

Moving backward, at $\mathcal{I}_{c,1} = \{v_1\}$, C tries to maximize its expected payoff, which is
$u_c(s; \mathcal{I}_{c,1}) = u_c(s; v_1) = \phi_1 u_c(Game\ 3) + \phi_2 u_c(t_{14})$. Because $u_c(Game\ 3) > u_c(t_{14})$, W will always choose $\phi_1 = 1, \phi_2 = 0$ to maximize $u_c(s; v_1)$. Therefore, $u_c(s; \mathcal{I}_{c,1}) = u_c(s; v_1) = u_c(Game\ 3)$.

Since Game 3 terminates at t_1 in Game 1, Game 4 terminates at t_1 in Game 1. Substituting ϕ_i and ψ_j into Eq. (4.1), we have:

$$\begin{cases} s_c^4 = & (\neg init, s_c^3, s_c^3, p') \\ s_w^4 = & (s_w^3, collude, s_w^3, report\ p^* = p', p', p', p'). \end{cases}$$

Thus, the unique sequential equilibrium has been proven. ■

4.8.2 Insights on Contract Design

According to Theorem 8, two conditions ($w_c > b > w_h - c + d_w$ and $w_h > c$) must be satisfied to have the unique sequential equilibrium. When $b \leq w_h - c + d_w$, the collusion will not happen, because there would be no motivation for W to collude, according to the analysis on the Collusion contract. In addition, it is obvious that $w_c > b$, since C would not pay W more bribe than its gain from the collusion. Also, $w_h > c$ is set that by H in the Watchtower contract, because W does not accept unpaid jobs. Therefore, these three contracts are effective to counter collusion, as long as $w_c > w_h - c + d_w$. The monetary variables w_h and d_w can be set by H in the Watchtower contract. Therefore, as long as H sets $d_w < w_c$, we have $w_c > w_h - c + d_w$. H can infer w_c based on the transaction history of this channel.

It can also be observed that the watchtower is required to lock a small and reasonable collateral $d_w < w_c$. Since the value of cryptocurrencies fluctuates dramatically, the time value of money (TVM) concept in financial management also applies to cryptocurrencies, or might even contribute more according to the historic volatility. Thus, a small collateral can incentivize the watchtowers to participate and guarantee security in PCNs.

Based on the analysis of Game 4, the rational watchtower and counterparty will not collude or cheat with the existence of all the three contracts. Thus, the rational parties will never execute the Collusion or Betrayal contracts, even if allowed. This indicates that the rational parties are involved in minimal operations on the blockchain. In addition, the Watchtower contract [80] is compilable with the PCN protocol and does not require on-chain storage or execution.

4.9 Implementation

We provide a proof-of-concept implementation of the proposed smart contracts to evaluate the scalability and efficiency. In the following, we present a high-level overview of the experiments. Because the Watchtower contract has been embedded in the Bitcoin Lightning [36] protocol, our experiment involves two contracts: the Collusion and Betrayal contracts. We implement the contracts in Solidity and execute them on Ethereum with Geth [81]. The contracts are loosely coupled with the actual channel monitoring jobs as an external service. The actual channel monitoring jobs can be treated as blackboxes, and the contracts do not need to know their internal details.

4.9.1 Scalability

As we surveyed in Section 4.2, Cerberus [75, 80] has provided an implementation of the watchtower functionality, which is compilable with the Lightning Network [36] protocol. Specifically, it allows the hiring party to employ a watchtower and enable the watchtower to lock and reclaim a deposit, which has the similar logic steps as our Watchtower contract except for Cerberus's requirement of a large deposit. This extension of the Lightning Network has shown that watchtowers can scale well in PCNs, *i.e.*, the number of on-chain transactions is constant and independent of the number of transactions in an off-chain channel, similarly to the current Lightning protocol.

Table 4.1 Cost of using the smart contracts on Ethereum. We have approximated the cost in USD ($) using the conversion rate of 1 ether = $156.89 and the gas price of 2 Gwei which reflects the real world costs as of April 2020.

Contract	Operation	Cost(Gas)	Cost($)
Collusion	Init	1,637,844	0.5139
	Create	225,583	0.0708
	Collude	62,733	0.0197
	Distribute	95,242	0.0299
Betrayal	Init	1,672,264	0.5247
	Create	137,546	0.0432
	Betray	68,725	0.0216
	Distribute	544,476	0.1708

4.9.2 Overhead and Financial Cost

According to the analysis and discussion in Section 4.8, The rational parties will never execute the Collusion or Betrayal contracts. The Watchtower contract is compilable with the PCN protocol, and does not require executions on the blockchain. Thus, we evaluate the cost of executing the Collusion and Betrayal contracts on Ethereum, in case that a party happens to sign and execute the contracts. The results are presented in Table 4.1. The cost is in the amount of gas consumed by each function, and the

converted monetary value in US dollar ($). The gas price was 2×10^{-9} ether (2 Gwei) and the exchange rate was 1 ether = $158.52 as of April 2020. The total cost of the Collusion contract is about 2 million gas ($0.63). The total cost of the Betrayal contract is about 2.4 million gas ($0.76).

The financial cost for running the smart contracts is roughly related to the computational and storage complexity of the function. For example, the cost of the $Init$ operation is dramatically higher than the other operations. The fundamental reason is that the cost of data storage on the blockchain is very expensive. Thus, the execution cost can be significantly reduced by recycling the contracts. Specifically, a hiring party may outsource the channel monitoring job to a watchtower from time to time. Thus, it is not cost-effective or necessary to initiate a new smart every time. In the smart contracts, there is a $reset()$ function that can be called by the contract owner after the contract has concluded. This function can clean up the contract and reset it to the initial state, which allows the reuse of contracts.

4.10 Conclusion

In this work, we studied the collusion problem for watchtowers in PCNs, where a hiring party outsources the payment channel monitoring job to a watchtower, but the watchtower could benefit from coordinating with the counterparty and not performing the job. To address this problem, we proposed a smart contract based solution by leveraging economic approaches. Specifically, we designed three smart contracts, the Watchtower, Collusion, and Betrayal contracts, to bring distrust between parties and guarantee security in PCNs. We conducted detailed analyses of the contracts and rigorously proved that there is a unique sequential equilibrium, where the rational parties will not collude or cheat and are involved in minimal on-chain operations. In particular, a watchtower only needs to lock a small deposit, which incentivizes the participation of watchtowers and users. Moreover, a proof-of-concept implementation of the smart contracts was provided and executed on Ethereum. The performance demonstrated that the contracts are gas-efficient.

CHAPTER 5
BALANCE-AWARE THROUGHPUT MAXIMIZING ROUTING IN PAYMENT CHANNEL NETWORKS

Payment channel networks (PCNs) have been proposed to tackle the scalability issues in blockchain-based cryptocurrencies, which offer the off-chain settlement of transactions. However, the balance depletion problem caused by unidirectional transactions may jeopardize the payments in PCNs. Existing works address this problem through sending artificial payments for the sole purpose of rebalancing the channels. In this work, we take advantage of a unique property of routing in PCNs to mitigate this channel depletion problem. Specifically, we design BAR, a distributed *B*alance-*A*ware *R*outing protocol. We prove that BAR maximizes the transaction throughput, subject to the conservation, timeliness, and feasibility constraints. Moreover, we modify the original Hashed Time-Lock Contract protocol to adapt it to BAR, such that BAR achieves efficiency and atomicity. Extensive simulations demonstrate that BAR outperforms a state-of-the-art algorithm Spider [44] in terms of success ratio and transaction throughput.

5.1 Introduction

Over the past decade, the blockchain-based cryptocurrencies have risen to more than $800 billion in peak capital, e.g., Bitcoin [7] and Ethereum [13]. Nevertheless, blockchain-based crytocurrencies cannot be widely applicable when scaling up. For example, the maximum number of transactions per second (tps) in Bitcoin is only 7 [16], which is not comparable to over $47,000$ peak tps processed by Visa [17].

Payment channel networks (PCNs), e.g., Bitcoin Lightning Network [18] and Ethereum Raiden Network [19], have been proposed to tackle the scalability issues [18]. PCNs can process instant payments without slow and expensive blockchain transactions. A payment is routed through multiple intermediate channels (ICs). Therefore, it is necessary to optimize routing in PCNs. Several existing works have investigated payment routing in PCNs [20–25, 31]. However, these efforts underestimate the importance of the key realistic constraints that are not common in the conventional ad-hoc networks. The first feature is the *feasibility constraint*, which means the balance requirements on the ICs. Due to privacy concerns, the channel balance information is not public, which may lead to transaction failures. The second feature is referred to as the *timeliness constraint*, because the tolerance on each user's cooperating time varies. The tolerance is judged by the number of hops to the recipient (more details in Section 2.3). One more special feature is that the payment transactions in opposite directions can cancel each other. The fundamental reason is that the data packets in the ad-hoc network are differentiated, while the payments in PCNs do

not. This can significantly improve the performance of payment routing. However, it is not supported in the current PCN protocol. An example illustrates this special feature in Figure 5.1.

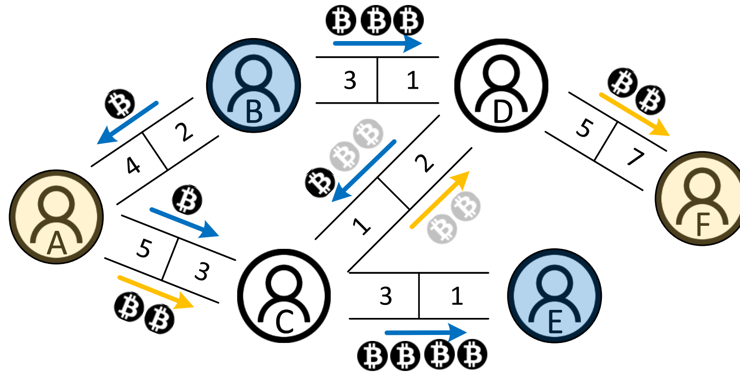


Figure 5.1 Example showing the special feature of routing in PCNs. A sends 2₿ to F; B sends 4₿ to E. Two values between two nodes represent the fund distribution in the channel. The channel $C \leftrightarrow D$ is involved in both payments. Although the bottleneck of (C, D) is 1₿, it can fulfill both payment requests.

Another uniqueness of PCNs is that payment routing can cause the rebalancing issue. During the payment routing, an intermediate user gains funds from its transferor and forwards funds to its transferee. Consequently, its balance decreases on the channel with its transferee, while its balance increases on the channel with its transferor. Although the total of its funds remains, its funds are redistributed. This may result in undesired balance depletion of certain channels. In the current PCN protocol, one has to close and reopen channels, which requires slow and expensive on-chain operations. Even if it supports additional deposits into an existing channel in the future, it still requires expensive blockchain transactions. Since payment routing redistributes the balances, we can take advantage of it to rebalance the channels. Recently, Khalil *et al.* proposed Revive [45], which rebalances channels independently of the payment routing and requires a centralized leader. Sivaraman *et al.* designed Spider [44], which achieves high throughput while maintaining the original channel balances. However, the rebalancing option is not provided to the channels. Thus, it is desirable to provide channel rebalancing, when designing payment routing algorithms in PCNs.

In this work, we investigate the balance-aware payment routing, which fulfills payments while rebalancing payment channels in PCNs. A balance-aware payment routing has a number of distinct characteristics. Thus, it is expected to satisfy a set of desired properties. First, a balance-aware routing protocol is expected to provide the rebalancing option to the channels, which is referred to as *channel balance awareness*, because payment routing could lead to undesired depletion on certain channels. Second, a balance-aware payment routing should satisfy *distributedness*, as no central administrative operator should exist or be trusted. Third, a balance-aware payment routing protocol should satisfy *efficiency*, *i.e.*, to minimize the latency incurred by transmitting multiple payments through one channel. Lastly, a

balance-aware payment routing should satisfy *atomicity*, such that a payment can be sent as several partial payments, and the recipient cannot claim the payment until it receives all the partial payments.

In face of these challenges, we propose BAR, a *B*alance-*A*ware payment *R*outing protocol that satisfies efficiency, distributedness, and atomicity. Specifically, we investigate the balance-aware payment routing in PCNs from an optimization perspective. This problem is referred to as the *Transaction Throughput Maximization* problem: maximize the total amount of fulfilled payments, while rebalancing the payment channels, such that the conservation, timeliness, and feasibility constraints are satisfied. Meanwhile, we modified the original HTLC mechanism to provide efficiency and adapt it to the Atomic Multi-Path Payments (AMP) [46] to achieve atomicity. *The main contributions of this work are*:

- To the best of our knowledge, we are the first to consider the balance-aware payment routing, which maximizes the throughput while rebalancing the channels in PCNs.
- We investigate important design goals of payment routing in PCN, which are referred to as channel balance awareness, distributedness, efficiency, atomicity, and optimality.
- We propose BAR, a distributed balance-aware routing protocol that maximizes the throughput in PCNs. BAR consists of three stages: Payment Flow Construction, HTLC Establishment and Payment Forwarding.
- We achieve channel balance awareness and optimality by designing a distributed algorithm to maximize the transaction throughput while rebalancing the channels.
- We also modify the original HTLC protocol to provide efficiency and adapt it to the balance-aware payment routing to guarantee atomicity.
- Extensive simulations demonstrate that BAR not only maximize the transaction throughput, but also achieves superior success ratio and transaction throughput over the state-of-art algorithms Spider [44] and LND [47].

The remainder of the chapter is organized as follows. In Section 5.2, we provide a brief literature review of related work. In Section 5.3, we formally describe the system model, outline the design goals and give the problem formulation. In Section 5.4, we illustrate the balance-aware payment routing protocol BAR, demonstrate its design, and analyze its properties. In Section 5.5, we test and validate the performance of BAR by comparing it to the state-of-the-art algorithms. We summarize this chapter in Section 5.6.

5.2 Related Work

The payment routing in PCNs has received increasing attentions. In 2016, Prihodko *et al.* [23] developed Flare, where each node stores a routing table formed by the adjacent nodes and paths to a list of beacon nodes. Malavolta *et al.* [21] studied the privacy-reserving routing and developed SilenWhisper based on Landmark Routing [26]. SpeedyMurmurs [22] used an improved algorithm based on embedding-based path discovery [27]. Rohrer *et al.* [24] sketched the payment flow as multiple paths adding up together to make use of the available capacities efficiently. Along this line, Yu *et al.* [25] outlined a scattered algorithm, which improved the success ratio and reduced the system overhead. Zhang *et al.* [20] proposed a distributed algorithm CheaPay to minimize the transaction fee. Wang *et al.* [48] developed a dynamic routing algorithm Flash, which differentiates elephant payments from mice payments. However, all the previous studies ignore the rebalancing issue.

In 2017, Khalil *et al.* proposed Revive [45] for channel rebalancing, which requires a centralized leader and works in network topologies with cyclic graphs. Later, Subramanian *et al.* [49] designed a decentralized technique of rebalancing in acyclic PCNs. But both of them treat rebalancing as a separate task independent of the payment routing. Recently, Sivaraman *et al.* [44] developed Spider, a payment routing algorithm that maximizes the throughput while maintaining the original channel balances. However, Spider does not provide the rebalancing option to the channels. Whereas, we propose a distributed balance-aware payment routing protocol to fulfill the payments while rebalancing the channels in PCNs. Moreover, the balance-aware payment routing problem is modeled and studied from an optimization perspective.

5.3 System Model and Problem Formulation

In this section, we describe the network model and the payment model and give a precise problem formulation.

5.3.1 Network Model

A PCN can be represented as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges [25, 44, 45, 49] . Each node $v_i \in \mathcal{V}$ represents a user, who has a cryptocurrency account and establishes at least one payment channel with a peer user. Each edge $e = (v_i, v_j) \in \mathcal{E}$ represents a payment channel, where v_i is the *transferor* and v_j is the *transferee*. Each node v_i is associated with an HTLC tolerance τ_i, denoting the maximum time that v_i would wait for the preimage R provided by its transferee. Each edge is associated with several attributes. First, each edge $(v_i, v_j) \in \mathcal{E}$ has a channel balance $b_{i,j}$, denoting the amount of the remaining funds that v_i owns on this channel. Second, each edge $(v_i, v_j) \in \mathcal{E}$ has a rebalancing flag $r_{i,j}$, which is a binary value denoting whether this channel desires rebalancing.

Third, each edge $(v_i, v_j) \in \mathcal{E}$ has a rebalancing parameter $\delta_{i,j}$, denoting the maximum amount of funds that v_i is willing to transfer to v_j. We assume that $\delta_{i,j} \leq b_{i,j}$, because v_i cannot transfer more than the funds that it owns on this channel. The rationale behind this attribute is that v_i may want to limit the payment amount to avoid a depleted channel. On the other hand, the channel (v_j, v_i) is rebalanced, when v_i transfers a payment to v_j. However, the more v_j gains on the channel (v_j, v_i), the more v_j loses on its other channels. Because the total funds that an intermediate user owns on all of its channels remain, but the distribution of the funds changes after the payment routing. Note that we omit the transmission time in PCNs, because it is negligible in comparison with the transaction time on blockchain.

We assume that each user only has local knowledge on its incoming and outgoing channels, including their HTLC tolerances, balances, rebalancing flags, and rebalancing parameters. In general, each user cannot know the above information of any remote edge, due to privacy concerns and dynamics.

5.3.2 Payment Model

A *payment request* is denoted by $q = (s, t, a)$, where v_s and v_t are the sender and recipient respectively, and a is the amount of the payment. A payment request q is performed via an (s, t) *payment flow* defined as follows.

Definition 5 (Payment Flow). *Given a PCN $G = (\mathcal{V}, \mathcal{E})$ and a payment request $q = (s, t, a)$, an (s, t) payment flow is a function $f^{s,t} : \mathcal{E} \to \mathbb{R}^{\geq 0}$, such that for any $(v_i, v_j) \in \mathcal{E}$, $f^{s,t}_{i,j}$ represents the amount of forwarded transaction on the channel (v_i, v_j) to fulfill q.*

Here we abuse the notation $f^{s,t} \in f$ to represent that an (s, t) payment flow $f^{s,t}$ is involved in the overall payment flow f in G. For a payment flow, all the involved payment channels are called *involved channels* (ICs), and all the involved users except the sender and the recipient along the payment path are called *intermediate users* (IUs).

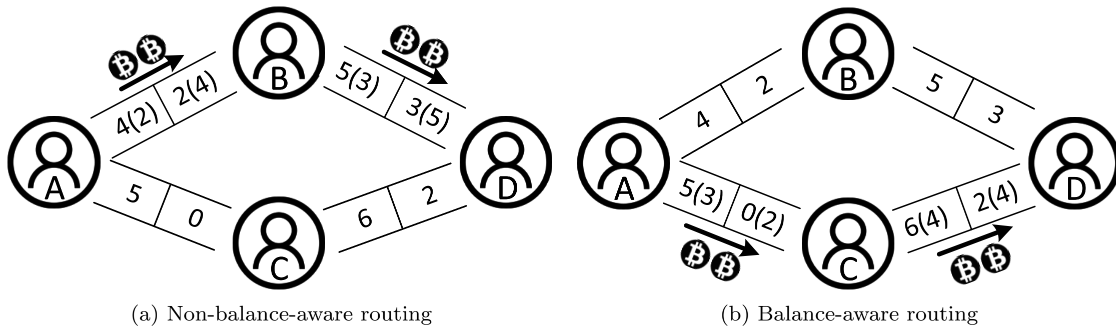
5.3.3 Design Goals

Under the network and payment models specified above, we derive a set of desirable design goals that a payment routing protocol should satisfy, which are elaborated below.

- *Channel Balance Awareness*: A payment routing protocol satisfies channel balance awareness, if it provides the rebalancing option to the payment channels. When an IU forwards a payment, its balance decreases on the channel with its transferee, while its balance increases on the channel with its transferor. Although its total funds remain, the funds are redistributed. This may result in undesired balance depletion of certain channels. In the current PCN protocol, one has to close and

reopen channels, which requires slow and expensive on-chain operations. Even if it supports additional deposits into an existing channel in the future, it still requires expensive blockchain transactions. Since payment routing can redistribute the channel balances, we can take advantage of it to rebalance the channels off chain. Thus, it is desirable to provide channel rebalancing, when designing payment routing algorithms in PCNs.

An example illustrates the importance of channel balance awareness in Figure 5.2. The sender A sends 2Ƀ to D. There exist two payment paths to fulfill this payment: $A \rightarrow B \rightarrow D$ and $A \rightarrow C \rightarrow D$. It can be observed that channel (A, C) desires rebalancing. In the non-balance-aware routing (in Figure 5.2(a)), the algorithm randomly picks a path and does not rebalance (A, C). In the balance-aware routing (in Figure 5.2(b)), (A, C) has a higher priority to be involved in routing, which rebalances (A, C).



(a) Non-balance-aware routing

(b) Balance-aware routing

Figure 5.2 Illustration for channel balance awareness. Two values between two nodes represent the fund distribution between two users before routing. The values in the brackets represent the fund distribution after routing.

- *Distributedness*: A payment routing protocol satisfies distributedness, if it dose not rely on a centralized party. Centralized routing is subject to a single point of failures upon external attacks, and hence cannot be trusted. Instead, each user runs the same local algorithm based on its local data and communication with each other.

- *Efficiency:* A payment routing protocol satisfies efficiency, if it minimizes the routing and payment latency incurred by transmitting multiple payments through a payment channel simultaneously. In the balance-aware payment routing, a payment channel can be involved in several payment requests at the same time. Thus, it is necessary to guarantee that the transactions introduce the minimum latency.

- *Atomicity:* A payment routing protocol satisfies atomicity for security purposes, if a sender can split and send a large payment into several partial payments, and a recipient cannot claim a payment until it receives all of the partial payments.

5.3.4 Problem Formulation

To formally formulate our studied problem, we introduce the following necessary concepts.

- *Conservation constraint*: An (s,t) payment flow satisfies conservation constraint, if $\forall v_i \in \mathcal{V} \backslash \{v_s, v_t\}$,

  $$\sum_{(v_i,v_j)\in\mathcal{E}} f_{i,j}^{s,t} = \sum_{(v_k,v_i)\in\mathcal{E}} f_{k,i}^{s,t}.$$

  Conservation guarantees that no IC gains or loses any funds by accepting or forwarding the payment transactions.

- *Feasibility Constraint*: An (s,t) payment flow satisfies feasibility constraint, if $\forall v_i \in \mathcal{V} \backslash \{v_s, v_t\}$,

  $$f_{i,j}^{s,t} \leq \delta_{i,j}.$$

  Feasibility guarantees that the payment transaction can be successfully transferred through each IC without violating the rebalancing parameter.

- *Timeliness constraint*: An (s,t) payment flow satisfies timeliness constraint, if $\forall (v_i, v_{i+1})$ along an (s,t) payment path $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ where $v_0 = v_s$ and $v_L = v_t$.,

  $$\tau_i \geq L - i, \forall i \in [0, L-1].$$

  Timeliness ensures that the payment transaction can be successfully fulfilled within the HTLC tolerance of each IC, which guarantees the commitment of honest processing at any IC.

Because payment routing occupies the funds in the ICs and consumes liquidity in a PCN, it is necessary to maximize the total amount of the payment transactions, while rebalancing the payment channels, such that the conservation, feasibility, and timeliness constraints are guaranteed. Towards this goal, we consider the following optimization problem in PCNs:

Transaction Throughput Maximization: Given a payment request list

$$\mathcal{Q} = \{(v_{s_1}, v_{t_1}, a_1), \ldots, (v_{s_n}, v_{t_n}, a_n)\},$$

find a conservative, feasible, and timely payment flow, such that the transaction throughput $\sum_{k=0}^{n} \sum_{(v_j,v_{t_i})\in\mathcal{E}} f_{j,t_i}^{s_k,t_k}$ is maximized.

5.4 BAR: A Distributed Balance-aware Routing Protocol in PCNs

In this section, we design and analyze BAR. We first provide the high-level overview and intuition behind BAR, and then follow the design goals that are outlined in Section 5.3.3 to provide a detailed protocol description.

5.4.1 Design Rationale

The rebalancing issue occurs due to the uniqueness of PCNs. When an intermediate user routes a payment, its funds are redistributed between its incoming and computing channels. Although its total funds remain, this may result in undesired depletion of certain channels. In the current PCN protocol, a depleted channel has to be closed and reopened with expensive on-chain operations. Thus, it is necessary to provide channel rebalancing for payment routing in PCNs.

We propose BAR, a distributed balance-aware routing protocol that maximizes the transaction throughput while rebalancing payment channels in PCNs. BAR achieves channel balance awareness and distributedness by designing a distributed algorithm derived from the Goldberg-Tarjan algorithm [50] and the Dinic's algorithm [51]. Meanwhile, BAR achieves efficiency by modifying the original HTLC mechanism [18] and ensures atomicity by adapting it to Atomic Multi-Path Payments (AMP) [46].

5.4.2 Design Challenges

To apply the Goldberg-Tarjan algorithm [50] and the Dinic's algorithm [51] that were originally designed for centralized systems, as well as to adapt HTLC to the balance-aware routing, we need to address several challenges as follows.

1. To transform a centralized algorithm into a distributed algorithm, each node should execute the same local algorithm based on its local data and exchange messages with its transferors and transferees over secure communication channels.

2. To satisfy the timeliness constraint, the distance (the number of edges) from a node to the recipient should be labeled to guarantee that it does not exceed the node's HTLC tolerance.

3. On the one hand, the Goldberg-Tarjan algorithm is suitable for the distributed PCNs, because each node makes decisions locally based on the knowledge about its transferors and transferees. The Goldberg-Tarjan algorithm makes use of a height on each node to push flows to downhill transferors. However, a node's height does not always reflect its distance, because the excess flows can only be pushed back to the transferor by lifting the height of a transferee. On the other hand, the Dinic's algorithm introduces the concepts of level and blocking flow, where a node's level represents the

distance from the node to the recipient. However, the Dinic's algorithm finds blocking flows by depth-first-search, which is not suitable for the distributed PCNs. Hence, to apply the Goldberg-Tarjan algorithm and the Dinic's algorithm in PCNs, we need to modify the current algorithms carefully to satisfy distributedness as well as optimality.

4. The payment transactions in opposite directions cancel each other in PCNs, which is a unique characteristics that does not exist in the conventional ad-hoc network. The fundamental reason is that the data packets transmitted in the ad-hoc network differentiate in opposite directions, while the payments in PCNs do not. An example illustrates this unique characteristics in Figure 5.1, where A sends 2Ƀ to F, and B sends 4Ƀ to E. The payment channel $C \leftrightarrow D$ is involved in both payment requests. Although the bottleneck of (C, D) is 1Ƀ, it can fulfill both payment requests. Thus, it can significantly improve the performance of payment routing, if we can take advantage of this special feature.

   However, the current PCN protocol does not support this, because each HTLC is associated with a transaction to guarantee the off-chain security almost the same as the original blockchain. Although the cancellation of multiple transactions can result in one transaction, the HTLCs are constructed and executed separately. To enable the transaction cancellation that involves multiple transactions and HTLCs, we need to modify the current HTLC protocol efficiency and atomicity as well as off-chain security.

We address these challenges in the following subsections and outline how the users are expected to execute payment requests in BAR.

5.4.3 Payment Flow Construction

The first stage is to construct payment flows for the payment requests, such that the total amount of the fulfilled payment requests are maximized. In this section, we design and analyze BAR, a distributed balance-aware algorithm that determines a payment flow, while rebalancing payment channels.

5.4.3.1 Design Rationale

Before we formally describe the design of BAR, we introduce the main notations in Table 6.1 and necessary definitions in the following.

Excess: The excess x_i is the difference between the total amount of v_i's gained transactions and the total amount of v_i's forwarded transactions, *i.e.*, $x_i = \sum_{v_k \in \mathcal{I}_i} f_{k,i} - \sum_{v_j \in \mathcal{O}_i} f_{i,j}$.

Height: The height h_i^t is an integer that labels the distance (number of edges) from v_i to the recipient v_t.

Uphill transferor: A node v_j is v_i's uphill transferor, if $b_{j,i} > 0$ and v_j's height is 1 unit higher than v_i's height, i.e., $h_i^t = h_j^t - 1$.

Downhill transferee: A node v_j is v_i's downhill transferee, if $b_{i,j} > 0$ and v_j's height is 1 unit lower than v_i's height, i.e., $h_i^t = h_j^t + 1$.

Table 5.1 Main notations

Notation	Meaning
$r_{i,j}$	the rebalancing flag of channel (v_i, v_j)
$\delta_{i,j}$	the rebalancing parameter of channel (v_i, v_j)
$b_{i,j}$	the amount of remaining balance of channel (v_i, v_j)
$f_{i,j}$	the amount of the forwarded payment on channel (v_i, v_j)
$\mathcal{I}_i$	the set of v_i's transferees
$\mathcal{O}_i$	the set of v_i's transferors
τ_i	the HTLC tolerance of v_i
x_i	the excess of v_i
h_i^t	the relative height of v_i to the recipient v_t

Our algorithm BAR is based on the Goldberg-Tarjan algorithm [50] and the Dinic's algorithm [51], which were originally designed to solve the maximum flow problem in centralized systems. In the Goldberg-Tarjan algorithm, each node v_i finds its augmenting paths based on its local knowledge about its transferors and transferees. In the Dinic's algorithm, each node v_i finds its augmenting paths based on node heights in a depth-first-search way. BAR combines these two algorithms by leveraging the concept of node heights and allowing each nodes to make local decisions by exchanging messages with its transferors and transferees.

BAR consists of four stages: initialization, label, push, and request. In the initialization stage, the algorithm initializes the excess and rebalancing flags for each node v_i. In the label stage, each node v_i labels its height of by exchanging messages with all its transferors and transferees. In the push stage, each node v_i first pushes its excess to its transferees who are 1 unit lower until its excess turns 0 or there is no valid transferee. Then v_i pushes its remaining excess back to its transferors who are 1 unit higher until its excess turns 0. In the request stage, the algorithm generates a payment flow to fulfill the payment requests. We present the detailed algorithms in the following subsection. Note that by "Broadcast a message", we essentially mean one node sending a message to the other through a secure communication channel, rather than sending a payment through the payment channel.

5.4.3.2 Design of BAR

In this section, we describe the details of BAR, which are illustrated in Algorithms 7, 8, 9, and 10.

Algorithm 7: BAR-Init

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i
Output: v_i

```
1  x_i ← 0 ; // Initialize v_i's excess to 0
2  I_i ← the set of v_i's transferor nodes;
3  O_i ← the set of v_i's transferee nodes;
4  // Initialize rebalancing parameters
5  O_i^0 ← ∅; O_i^1 ← ∅; I_i^0 ← ∅; I_i^1 ← ∅;
6  for v_j ∈ O_i do
7  |   f_{i,j} ← 0;
8  |   if (v_i, v_j) desires rebalancing then
9  |   |   r_{i,j} ← 1; O_i^1 ← O_i^1 ∪ {v_j};
10 |   else
11 |   |   r_{i,j} ← 0; O_i^0 ← O_i^0 ∪ {v_j};
12 |   end
13 end
14 for v_j ∈ I_i do
15 |   f_{i,k} ← 0;
16 |   if (v_k, v_i) desires rebalancing then
17 |   |   r_{k,i} ← 1; I_i^1 ← I_i^1 ∪ {v_k};
18 |   else
19 |   |   r_{k,i} ← 0; I_i^0 ← I_i^0 ∪ {v_k};
20 |   end
21 end
22 return v_i
```

The initialization stage is shown in BAR-Init (Algorithm 7). BAR-Init initializes the excess and rebalancing flags for each node v_i. The excess of v_i is initialized to 0 (Line 1). We use $\mathcal{I}_i$ and $\mathcal{O}_i$ to denote the set of v_i's incoming transferors and the set of v_i's outgoing transferees, respectively (Lines 2 and 3). Then v_i checks if its transfrees desire rebalancing and mark them (Lines 6 to 13). Similarly, v_i sets if itself desires rebalancing as a transferee and mark the channels (Lines 14 to 21). We shall run Algorithm 7 to initialize each node $v_i \in \mathcal{V}$.

The label stage is shown in BAR-Label (Algorithm 8). BAR-Label labels the height of each node v_i in a breadth-first-search way. The height of a recipient node v_i is set to 0 and broadcast to all v_i's transferors and transferees (Lines 3 to 5). When v_i receives the height h_j^t from its transferee v_j, then v_i updates its height h_i^t, if this is the first time to set its height, *i.e.*, $h_i^t = -1$, or v_i's current height is greater than the new height, *i.e.*, $h_i^t > h_j^t + 1$ (Lines 6 to 11). If v_i's height h_i^t is higher than its HTLC tolerance τ_i, then v_i does not satisfy the timeliness constraint and is not valid for the payment routing. In this case, v_i's height is set to be infinity, which means v_i cannot reach the recipient within τ_i (Line 13). Once v_i has received the height messages from all its transfrees and finished updating its height, v_i broadcasts h_i^t to all v_i's transferors and transferees (Line 14). We shall run Algorithm 8 to label the height for each node $v_i \in \mathcal{V}$.

Algorithm 8: BAR-Label

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i
Output: the height h_i^t of node v_i

1 $h_i^t \leftarrow -1$; `// Initialize` v_i`'s height`
2 `// Initialize recipient` v_t`'s height`
3 **if** $v_i = v_t$ **then**
4     $h_i^t \leftarrow 0$; Broadcast h_i^t to all nodes in $\mathcal{I}_i \cup \mathcal{O}_i$;
5 **end**
6 **for** $v_j \in \mathcal{O}_i$ upon v_i receiving h_j^t **do**
7   `// if` v_i`'s height is firstly set or` v_i`'s new height is smaller`
8   **if** $\delta_{i,j} > 0$ **and** $(h_i^t = -1$ **or** $h_i^t > h_j^t + 1)$ **then**
9         $h_i^t \leftarrow h_j^t + 1$; `// Update` v_i`'s height`
10   **end**
11 **end**
12 `//` v_i `is unreachable`
13 **if** $h_i^t > \tau_i$ **or** $h_i^t = -1$ **then** $h_i^t \leftarrow \infty$;
14 Broadcast h_i^t to all nodes in $\mathcal{I}_i \cup \mathcal{O}_i$
15 **return** h_i^t

Algorithm 9: BAR-Push

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i, a sender v_s, a recipient v_t.
Output: a residual graph G, an (s,t) payment flow $\Delta^{s,t} : \mathcal{E} \rightarrow \mathbb{R}$.

1 **for** $v_j \in \mathcal{I}_i \cup \mathcal{O}_i$ upon v_i receiving $\Delta_{j,i}^{s,t}$ **do**
2   `//` v_i `is 1 unit higher/lower than` v_j
3   **if** $h_j^t = h_i^t + 1$ **or** $h_j^t = h_i^t - 1$ **then**
4   `// push flow from` v_j `to` v_i
5         $\delta_{i,j} \leftarrow \delta_{i,j} + \Delta_{i,j}^{s,t}$; $x_i \leftarrow x_i + \Delta_{i,j}^{s,t}$;
6   **end**
7 **end**
8 `// push excess to downhill transferees`
9 **for** $v_j \in \mathcal{O}_i^1 \cup \mathcal{O}_i^0$ **do**
10   **if** $e_i = 0$ **then** break;
11   **if** $h_i^t = h_j^t + 1$ **and** $\delta_{i,j} > 0$ **and** $e_i > 0$ **then**
12         $\Delta_{i,j}^{s,t} \leftarrow \min\{x_i, \delta_{i,j}\}$; $x_i \leftarrow x_i - \Delta_{i,j}^{s,t}$;
13         $\delta_{i,j^{s,t}} \leftarrow \delta_{i,j} - \Delta_{i,j}^{s,t}$; $f_{i,j} \leftarrow f_{i,j} + \Delta_{i,j}^{s,t}$;
14   Broadcast $\Delta_{i,j}^{s,t}$ to v_j;
15   **end**
16 **end**
17 `// push excess to uphill transferors`
18 **for** $v_k \in \mathcal{I}_i^0 \cup \mathcal{I}_i^1$ **do**
19   **if** $e_i = 0$ **then** break;
20   **if** $h_i^t = h_k^t - 1$ **and** $\delta_{i,k} > 0$ **and** $e_i > 0$ **then**
21         $\Delta_{i,k}^{s,t} \leftarrow \min\{x_i, \delta_{i,k}\}$; $x_i \leftarrow x_i - \Delta_{i,k}^{s,t}$;
22         $\delta_{i,k}^{s,t} \leftarrow \delta_{i,k} - \Delta_{i,k}^{s,t}$; $f_{k,i} \leftarrow f_{k,i} - \Delta_{i,k}^{s,t}$;
23   Broadcast $\Delta_{i,k}^{s,t}$ to v_k;
24   **end**
25 **end**
26 **return** $(G, \Delta^{s,t})$

The push stage is shown in BAR-Push (Algorithm 9). BAR-Push pushes the excess of each node v_i, except for the recipient, until the excess turns 0. When v_i's receives a flow $\Delta_{j,i}$ pushed from one of its transferrors or transferees v_j, then v_i updates its rebalancing parameter $\delta_{i,j}$ and its excess x_i (Lines 1 to 7). When v_i's excess is positive, it pushes it excess. First, BAR-Push pushes v_i's excess as much as possible to v_i's downhill transferees whose heights are $h_i^t - 1$ until the excess turns 0 or there is no valid transferee. The amount of the pushed flow $\Delta_{i,j}$ is broadcast to the corresponding transferee v_j (Lines 9 to 16). Note that the channels whose rebalancing flags are 1 are used to push the excess downhills in a higher priority. If there still exists excess on v_i, then BAR-Push pushes v_i's excess back to v_i's uphill transferors whose heights are $h_i^t + 1$ until the excess turns 0. Note that the channels whose rebalancing flags are 0 are used to push the excess back in a higher priority. The amount of the pushed back flow $\Delta_{i,j}$ is broadcast to the corresponding transferor v_k (Lines 18 to 25). We shall run Algorithm 9 to push the excess for each node $v_i \in \mathcal{V} \backslash \{v_t\}$.

Algorithm 10: BAR-Request

```
   Input: a network G = (V, E), a payment request q = (v_s, v_t, a)
   Output: f^{s,t}
 1 // add payment amount to sender's excess
 2 f^{s,t} ← 0; x_s ← x_s + a;
 3 // lable recipient's height
 4 h_t^t ← BAR-Label(G, v_t);
 5 // recipient has not received full payment
 6 while x_t < a do
 7 |  for v_i ∈ V\{v_t} do
 8 |  |  while x_i > 0 do
 9 |  |  |  (G, Δ^{s,t}) ← BAR-Push(G, v_i, v_s, v_t);
10 |  |  |  f^{s,t} ← f^{s,t} ∪ Δ^{s,t};
11 |  |  end
12 |  end
13 |  for v_i ∈ V do h_i^t ← BAR-Label(G, v_i);
14 end
15 return f^{s,t}
```

Algorithm 11: BAR-Output

```
   Input: a network G = (V, E), a payment request list Q.
   Output: a payment flow f.
 1 f ← 0 ;
 2 for v_i ∈ V do BAR-Init(G, v_i);
 3 for q ∈ Q in parallel do f^{s,t} ← BAR-Request(G, q);
 4 return f
```

The request stage is shown in BAR-Output (Algorithm 10). BAR-Output generates a payment flow that maximizes the transaction throughput upon the arrival of a payment request. First, BAR-Output

adds the payment amount a to the sender v_s's excess (Line 2). Then, BAR-Output calls BAR-Label to find payment flows and calls BAR-Push to update the heights for each node $v_i \in \mathcal{V}$, as long as the payment has not been fulfilled (Lines 4 to 14). Once the payment is fulfilled or the timeout is reached, BAR-Output terminates.

The output stage is shown in BAR-Request (Algorithm 11). BAR-Request generates a payment flow that maximize the transaction throughput. First, BAR-Request runs BAR-Init to initialize each node $v_i \in \mathcal{V}$. Then, BAR-Request calls BAR-Output to fulfill each payment request parallelly.

5.4.3.3 Analysis of BAR

Because BAR is a distributed algorithm based on Goldberg-Tarjan algorithm [50] and Dinic's algorithm [51], we have the following theorem.

Theorem 9. BAR outputs an optimal solution to the Transaction Throughput Maximization problem.

We now analyze the message complexity of BAR. First, BAR-Init initializes each node by exchanging $O(|\mathcal{E}|)$ messages, where $|\mathcal{E}|$ is the total number of edges. Second, BAR-Label labels the height of each node by exchanging $O(|\mathcal{E}|)$ messages. Then, BAR-Push exchanges $O(|\mathcal{V}||\mathcal{E}|)$ messages on each node, where $|\mathcal{V}|$ is the total number of nodes. BAR-Output's message complexity is dominated by BAR-Label and BAR-Push. The while loop in BAR-Request terminates within $|\mathcal{V}|$ iterations. In total, the overall message complexity of BAR is $O(|\mathcal{V}|^2|\mathcal{E}|)$.

5.4.4 HTLC Establishment

A Hashed Time-Locked Contract (HTLC) is a script that permits a designated party (the transferee) to spend funds by disclosing the preimage of a hash. It also permits a second party (the transferor) to spend the funds after a timeout is reached in a refund situation. The original HTLC [18] was designed for payment routing in a single payment path. Thus, each HTLC is associated with one payment transaction. As discussed in Section 5.4.2, a payment channel can be involved in multiple payment transactions simultaneously in a balance-aware routing. However, the original HTLCs are constructed separately and vary enormously in logic steps. Multiple payment requests that go through the same payment channel have to be fulfilled sequentially. This can result in long latency and thus discourage user participation in PCNs.

To adapt the HTLC protocol to BAR, we provide efficiency by modifying the original HTLC: Suppose there are two transactions q_1 and q_2 in opposite directions on a payment channel $A \leftrightarrow B$, and $a_1 \geq a_2$ without loss of generality, where a_i is the payment amount of q_i. Then q_1 can split a_1 into two partial payments, *i.e.,* a_2 and $a_1 - a_2$. To cancel a_2 in both directions, the modified HTLC holds a_2 from A and takes 2 hashes as the input, which are the hash H_1 of the preimage R_1 and the hash H_2 of the preimage

R_2. Specifically, the modified HTLC has 2 possible outputs depending on the preimages provided by both A and B. If B does not produce R_1 within T_1, then the HTLC terminates, and A gets the refund a_2 without requiring R_2 from path 2. If B produces R_1 within T_1 to prove that the recipient on path 1 has received a_2, then the HTLC extends for a time duration of T_2. Note that B can prove the correctness of R_1 by Zero-Knowledge Proof [52] without revealing R_1. If A produces R_2 within T_2, then A and B reveal and swap R_2 and R_1, and A gets a_2 refunded. If A does not produce R_2 within T_2, then B reveals R_1 and receives a_2. A simple illustration of the modified HTLC is shown in Figure 5.3. Such a modification on HTLC can guarantee efficiency and security for a balance-aware routing.

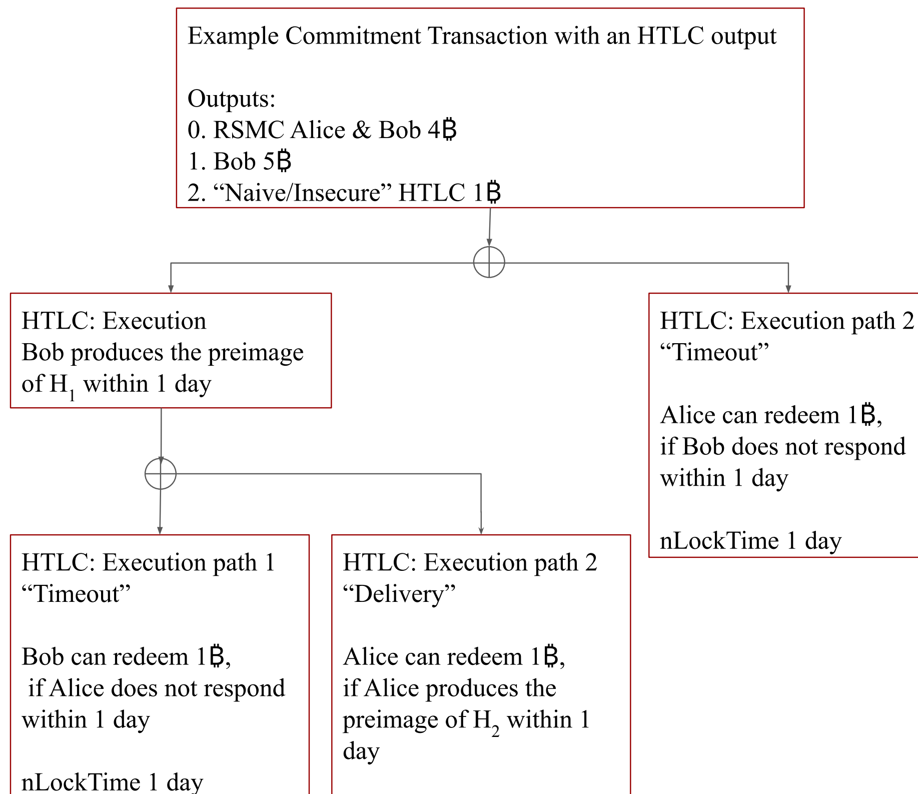


Figure 5.3 Modified hashed time-lock contract (HTLC). Alice sends 1₿ to Bob and Bob sends 1₿ to Alice via an HTLC tolerance of 2. Note that there are 3 possible spends from an HTLC output. If Bob does not produce the preimage of H_1 within 1 day, then the HTLC terminates. They can redeem path 2. If Bob produces the preimage of H_1 within 1 day, Bob does not need to reveal it. Then, if Alice produces the preimage of H_2 within another 1 day, they reveal and swap R_i and R_2 via fair exchange and redeem path 2. If Alice dose not produce the preimage of H_2 within another 1 day, Bob reveals the preimage of H_1 to Alice and redeem path 1.

An example compares the efficiency of the original HTLC and the modified HTLC in Figure 5.4. Two payment requests go through the same payment channel in opposite directions. When adopting the original HTLC (in Figure 5.4(a)), the payment request from C to D can only be fulfilled after the payment from A

to B is fulfilled. In total, it consumes the liquidity of up to

$5T \cdot 1Ƀ + 4T \cdot 1Ƀ + 3T \cdot 1 \ + 3T \cdot 1Ƀ + 2T \cdot 1Ƀ + 1T \cdot 1Ƀ = 18T \cdot 1Ƀ$ to fulfill both payment requests. When adopting the modified HTLC (in Figure 5.4(a)), both payment request can be fulfilled simultaneously. In total, it consumes the liquidity of up to $3T \cdot 1Ƀ + 2T \cdot 1Ƀ + 1T \cdot 1Ƀ + 3T \cdot 1Ƀ + 2T \cdot 1Ƀ = 6T \cdot 1Ƀ$ to fulfill both payment requests. Thus, the modified HTLC can significantly improve the efficiency of payment routing.

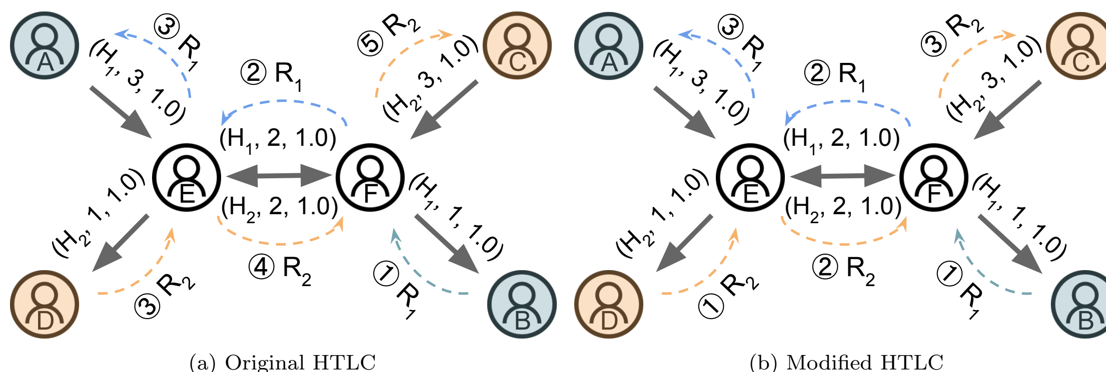


Figure 5.4 HTLC Establishment and Payment Forwarding in BAR. A sends a payment of 1 to B. C sends a payment of 1 to D. Circled numbers represent the sequence of the operations.

A large payment sometimes has to be splitted into several partial payments, to avoid transaction failures due to insufficient balances on the ICs. To guarantee atomicity, *i.e.,* the recipient cannot claim the payment until it receives all the partial payments, the Atomic Multi-Path Payments (AMP) protocol [46] can be embedded into the balance-aware routing by using secret sharing. Specifically, the sender derivates the based preimage R as partial preimages $r_1, \ldots, r_i, \ldots, r_n$. The hash h_i of r_i is associated with the ith partial payment. The recipient can reconstruct the base preimage to claim the total payment, only after it receives all the partial preimages.

5.4.5 Payment Forwarding

After the payment flow construction and the HTLC establishment processes, a sender can forward the payment to its recipient via the HTLCs. Multiple transactions that are involved in the same payment channel in opposite directions can cancel each other. A simple illustration of the payment forwarding is shown in Figure 5.4. Two edge-joint payment paths have been constructed in the previous stage, where A and C are the senders, and C and D are the corresponding recipients. An HTLC has been created on each IC on both payment paths. Since E requires the preimage of H_1 from F to release 1Ƀ on the blue payment path, and F requires the preimage of H_2 from E to release 1Ƀ on the orange payment path, E and F can reveal and swap their preimages through fair exchange. Therefore, the funds only need to be locked for

once, which makes the balance-aware routing more efficient.

5.5 Performance Evaluation

In this section, we evaluate the performance of BAR. As we surveyed in Section 5.2, there is no existing balance-aware payment routing protocol in payment channel networks. The main software that implements the Bitcoin Lightning Network [36] is the Lightning Network Daemon (LND) [47], which computes the shortest path to send a payment repeatedly, until success or timeout. The most related work is Spider [44], a state-of-the-art high throughout payment routing algorithm that maintains the original channel balances. Therefore, we evaluate the performance of BAR by comparing it to Spider and LND. LND is the baseline to show how well an algorithm can do without rebalancing and maintaining balances. Spider describes how well an algorithm can do without rebalancing.

5.5.1 Environment Setup

We obtained a real-world PCN topology from the Bitcoin Lightning Network [36]. In particular, we crawled a snapshot topology of the Lightning Network on July 14, 2020. To crawl the Lightning Network, we ran the Bitcoin Core daemon (bitcoind) [41], built a c-lightning [42] node on mainnet, and connected it to an existing Lightning node, which is the Bitstamp's Lightning Network node [43]. The network consists of $5,622$ nodes and $32,814$ channels. We use a real-world transaction dataset [53] that was processed [44]. In particular, it contains the credit card transactions that occurred in two days in September, 2013 in Europe, with the mean of €88, the median of €25, and the maximum of €$3,930$.

5.5.2 Performance Metrics

We use the following metrics for performance evaluation:

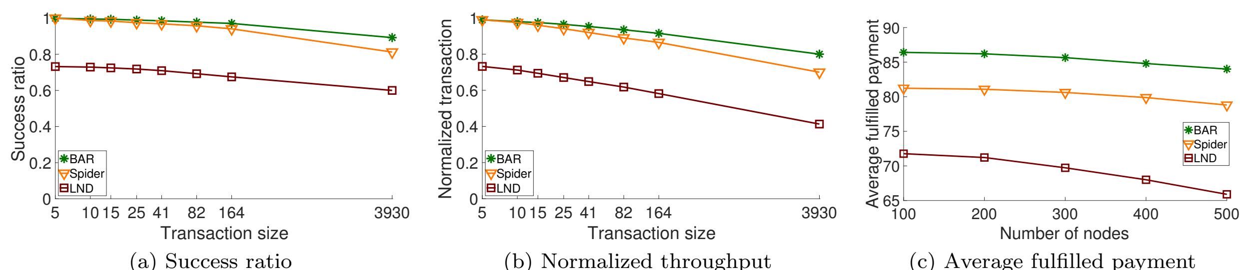
- *Success ratio:* The number of fulfilled payment requests over the number of all payment requests.
- *Normalized throughput:* The total amount of fulfilled payments over the total amount of all payment requests.
- *Average fulfilled payment:* The average amount of payment over all fulfilled payment requests.

5.5.3 Evaluation of BAR

Figure 5.5 shows success ratios, normalized throughputs, and average fulfilled payments of BAR, Spider, and LND.

Figure 5.5(a) shows the impact of transaction size on the success ratios achieved by BAR, Spider, and LND. BAR outperforms Spider and LND, because BAR rebalances payment channels and guarantees both

the feasibility and timeliness constraints. LND indicates that some large payments cannot be fulfilled without splitting the payments. All algorithms have dropping success ratios, when the transaction size grows. This is because the large payments need to be splitted into partial payments. If any one of the partial payments fails, the payment can not be fulfilled.



(a) Success ratio (b) Normalized throughput (c) Average fulfilled payment

Figure 5.5 Comparison between BAR and Spider

Figure 5.5(b) shows the impact of transaction size on the normalized throughputs achieved by BAR, Spider, and LND. Spider gives a slightly lower normalized throughput than BAR, because Spider forces the payment channels to maintain the original balances and adopts source routing. The normalized throughputs of all algorithms drop with larger transaction sizes. This is because a large payment needs to be splitted into partial payments, which has a high probability to fail one partial payment. LND does not split payments, which results in a low throughput.

Figure 5.5(c) shows the impact of the number of nodes on the average fulfilled payments. BAR outperforms Spider and LND, due to its rebalancing operation and guarantee on both the timeliness and feasibility. The average fulfilled payments of all schemes drop with more nodes, because the number of valid payment paths decreases.

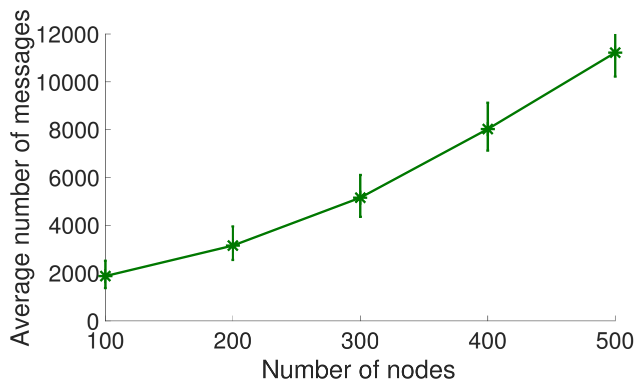


Figure 5.6 Message complexity of BAR

We also evaluate the convergence speed of BAR. The result is shown in Figure 5.6. We can observe that the average number of messages increases with the number of nodes. Since BAR implements a variant of the Goldberg-Tarjan algorithm [50] and the Dinic's algorithm [51], BAR has the growing trend of message complexity similar to that of the distributed Goldberg-Tarjan algorithm.

5.6 Conclusion

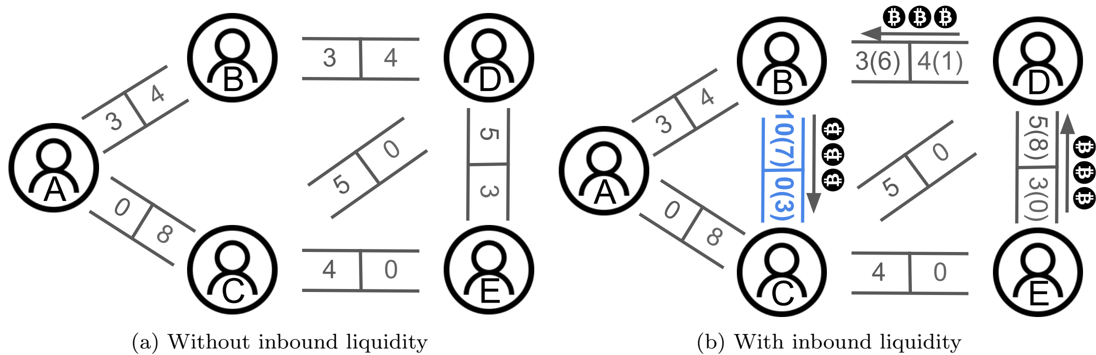
In this work, we investigated the balance-aware payment routing protocol to maximize the transaction throughput while rebalancing the payment channels in PCNs. We first suggested a set of crucial design goals for payment routing, which are referred to as channel balance awareness, distributedness, efficiency, and atomicity. Following these design goals, we presented a distributed balance-aware payment routing protocol BAR consisting of three stages: Payment Flow Construction, HTLC Establishment, and Payment Forwarding. BAR achieved channel balance awareness by providing the rebalancing option to the payment channels. To guarantee optimality, we formulated the Transaction Throughput Maximization problem and presented a distributed algorithm BAR. Moreover, we modified the original HTLC protocol to provide efficiency and adapted it to the balance-aware payment routing protocol to achieve atomicity. Extensive simulations demonstrated that BAR achieved outstanding success ratio and average acceptance value compared to a state-of-the-art algorithm Spider [44].

CHAPTER 6

AN INCENTIVE MECHANISM FOR CRYPTO CAPITAL COMMITMENT IN PAYMENT CHANNEL NETWORKS

Payment channel networks (PCNs) are proposed to improve the cryptocurrency scalability by settling off-chain transactions. However, a significant barrier is that a PCN user must solicit sufficient capital owned by the counterparty on its channel (*i.e.*, inbound liquidity) to receive payments. To alleviate this inbound liquidity problem, Channel Liquidity Marketplaces (CLMs), e.g., Bitcoin's Lightning Pool, have been introduced, such that participants can buy and sell inbound liquidity by trading crypto capital commitment in PCNs. Existing CLMs lack good incentive mechanisms that can attract more user participation. To fulfill this void, we design Cumulonimbus, an incentive mechanism for trading crypto capital commitment, which satisfies truthfulness, individual rationality, budget balance, and computational efficiency. Particularly, Cumulonimbus considers two unique features of crypto capital commitment, referred to as demand indivisibility and supply divisibility. Extensive simulations demonstrate that Cumulonimbus achieves higher satisfaction ratio, liquidity utilization, and social welfare compared with a state-of-the-art CLM mechanism Lightning Pool [54].

6.1 Introduction



(a) Without inbound liquidity (b) With inbound liquidity

Figure 6.1 Illustration for inbound liquidity. In the left, two values between two nodes represent the capital distribution. Although C owns 8₿ + 5₿ + 4₿ = 17₿, C's inbound liquidity from the counterparties (*i.e.,* A, C, and D) is 0. Thus, C cannot receive payments from any other user. In the right, C has an inbound liquidity of 10₿ from B by creating a payment channel. Now C can receive payments from A, B, D, and E. For example, C can receive 3₿ along $E \rightarrow D \rightarrow B \rightarrow C$. The values in the brackets represent the capital distribution after C receives 3₿ from E.

The past decade has seen a blooming of cryptocurrencies [14], *e.g.*, Bitcoin [7] and Ethereum [13]. However, cryptocurrencies cannot scale for wide-spread use, due to high overhead and storage

requirement [15]. Payment channel networks (PCNs), *e.g.*, Bitcoin's Lightning Network [18] and Ethereum's Raiden Network [19], have been proposed to tackle the scalability issues [18]. However, a significant barrier in PCNs is that a user can receive payments only if there is sufficient capital owned by the counterparty (*i.e.,* the other user on its channel), which is referred to as the *inbound liquidity* [54]. In Figure 6.1, an example illustrates the importance of inbound liquidity. In. Figure 6.1(a), C cannot receive payments from any other user, because C has no inbound liquidity from the counterparties (*i.e.*, A, D, and E) on it channels. In. Figure 6.1(b), C can receive payments from the other users after B provides an inbound liquidity of 10Ƀ by opening a channel to C. Therefore, inbound liquidity is essential as it directly determines the success of payments, which is the ultimate purpose of PCNs.

On the one hand, the *liquidity takers* who desire inbound liquidity need to convince others to open channels to them and to deposit crypto capital on their channels. On the other hand, the *liquidity makers* who own crypto capital seek to provide inbound liquidity for profit. To enable the liquidity takers and makers to publish their liquidity demand and supply information, researchers have developed various applications and systems for trading crypto capital commitment [54–58]. This new paradigm is commonly referred to as *channel liquidity marketplace* (CLM).

Unfortunately, the existing works only scratch the surface by either requiring voluntary participation without considering the design of incentive mechanisms or neglecting some critical properties of incentive mechanisms. When participating in a CLM, the liquidity makers must lock their capital in the channels for a certain time period. Since the value of cryptocurrencies fluctuates dramatically, the time value of money (TVM) concept in financial management also applies to cryptocurrencies, or might contribute more according to the historic volatility. Therefore, a liquidity maker would not be interested in participating in a CLM, unless it receives a satisfying reward to compensate its TVM consumption. Without adequate liquidity maker participation, it is impossible for the liquidity takers to solicit sufficient inbound liquidity, which is essential to the success of PCNs.

At first glance, crypto capital commitment seems to resemble the conventional capital commitment, *e.g.*, bond. However, there are two important differentiating features, which make the design of incentive mechanisms for crypto capital commitment more complex. The first feature is the *demand indivisibility*, due to the fact that a liquidity taker's demand should be either fully provided by one liquidity maker or none. This is because inbound liquidity separated on multiple channels does not help a liquidity taker receive sufficient large payments. The second feature is the *supply divisibility*, because a liquidity maker can split and deposit its crypto capital on multiple channels to fulfill multiple liquidity takers. Due to these reasons, incentive mechanisms for conventional goods or financial resources cannot be applied to crypto capital commitment.

In this chapter, we design an incentive mechanisms to motivate both the liquidity takers and makers to participate in CLMs, which allows them to buy and sell inbound liquidity by trading crypto capital commitment in PCNs. *The main contributions of this chapter are*:

- To the best of our knowledge, we are the first to design an incentive mechanism for trading crypto capital commitment with the consideration of demand indivisibility and supply divisibility.
- We design an incentive mechanism Cumulonimbus, which guarantees that a liquidity taker's demand is either fully supplied by a liquidity maker or none, but a liquidity maker's supply can fulfill multiple liquidity takers.
- We rigorously prove that Cumulonimbus satisfies truthfulness, individual rationality, budget balance, and computational efficiency.

The remainder of the chapter is organized as follows. In Section 6.2, we provide a brief literature review of the related work. In Section 6.3, we present the background in PCNs, formally describe the system model and incentive mechanism model, and give the necessary assumptions.

6.2 Related Work

Up to now, there are only limited efforts on the design of incentive mechanisms for crypto capital commitment in PCNs. Realizing the great potential benefit, some PCN power users have provided channel liquidity services, *e.g.*, Bitrefill's Thor [59], Yalls [60], LNBig [61], and ln2me [62]. These services promise to provide inbound liquidity by opening channels to the liquidity takers and deposit crypto capital on the channels. Nevertheless, there is only a single liquidity maker (the liquidity service provider itself) who determines and charges a posted price. The Celar Network [56] has designed a liquidity backing auction with a single liquidity maker, which utilizes VCG to determine the winning liquidity takers and payments. However, all these works do not involve adequate liquidity maker participation, which makes it impossible for the liquidity takers to solicit sufficient inbound liquidity,

ZmnSCPxj [58] has proposed a channel liquidity marketplace (CLM), where both the liquidity makers and takers can participate in trading crypto capital commitment. However, it focuses only on the protocol and smart contract design, instead of the incentive mechanism design. Recently, Lightning Lab [55] has released a CLM called Lightning Pool [54], which is implemented as a sealed-bid frequent batched uniform price double auction. Lightning Pool adopts a greedy algorithm to match the liquidity takers and makers, but does not consider truthfulness, individual rationality, or budget balance. Therefore, all the existing works either do not consider the design of incentive mechanisms or have neglected some critical properties of incentive mechanisms, In this chapter, we design an incentive mechanism to motivate both the liquidity

takers and makers to participate in CLMs to buy and sell inbound liquidity, while considering the unique features of crypto capital commitment, *i.e.,* the demand indivisibility and supply divisibility.

6.3 System Model and Problem Formulation

In this section, we present an overview of the channel liquidity marketplace system, describe the incentive mechanism model, and give the desired properties.

6.3.1 Background and System Overview

Cryptocurrencies have suffered from the large overhead of global consensus and security assurance, which largely limits their applications in real-world scenarios. To tackle the scalability issues, PCNs (*e.g.*, Bitcoin Lightning [36] and Ethereum Raiden [19]) have been proposed to offer off-chain settlement of transactions with minimal involvement of expensive blockchain operations. To send and receive payments in a PCN, a user must open a payment channel to another PCN user. Two users establish a payment channel by each depositing a certain amount of capital into a joint account and adding this transaction to the blockchain. Once the payment channel has been opened, a user is able to send payments, if it owns sufficient capital on its channel.

Inbound Liquidity. Unfortunately, it is not guaranteed that a user can receive payments by opening a payment channel. There has to be sufficient capital owned by the counterparty (*i.e.,* the other user on its channel). Such crypto capital commitment allocated by the counterparty is typically referred to as *inbound liquidity.* Because a user must convince other users to open channels to it and to allocate crypto capital commitment towards it, the *inbound liquidity problem* remains a significant barrier to the success of PCNs.

6.3.2 Channel Liquidity Marketplace

The adoption of *channel liquidity marketplace* (CLM) is one way to mitigate the inbound liquidity problem. A CLM, *e,g.*, Bitcoin Lightning Pool [55], is a marketplace that enables participants to buy and sell inbound liquidity by trading crypto capital commitment. In general, there are two roles of participants defined as follows.

Definition 6 (Liquidity Taker). *A Liquidity Taker is a participant who wants to receive payments and is willing to pay for inbound liquidity.*

Definition 7 (Liquidity Maker). *A Liquidity Maker is a participant who owns capital and is willing to sell inbound liquidity for profit.*

We consider a CLM consisting of an auctioneer, a set $\mathcal{T} = \{T_1, T_2, \ldots, T_i, \ldots, T_n\}$ of n liquidity takers and a set $\mathcal{M} = \{M_1, M_2, \ldots, M_j, \ldots, M_m\}$ of m liquidity makers. The non-trusted auctioneer designs

incentive mechanisms to achieve desired economic properties, determines the winning takers and makers, and calculate their payments. The inbound liquidity is assumed to be *homogeneous*, which means that there is no preference for the takers using inbound liquidity from different makers. Note that this model can be easily extended to the *heterogeneous* case by rating the inbound liquidity quality of each maker, such that a taker can set its preference to solicit inbound liquidity from certain makers. Since a liquidity taker needs inbound liquidity to receive payments, it cannot receive sufficient large payments, if the inbound liquidity is separated on multiple channels. Therefore, a liquidity taker's demand should be either fully provided by one liquidity maker or none, referred to as *demand indivisibility*. Meanwhile, a liquidity maker's supply can be provided to multiple liquidity takers, since it can split and deposit its capital on multiple channels, referred to as *supply divisibility*.

6.3.3 Incentive Mechanism Model

We aim to design an incentive mechanism that allows both liquidity takers and makers to buy and sell inbound liquidity by trading capital commitment, while considering demand indivisibility and supply divisibility. In the incentive mechanism, the makers are sellers, and the takers are buyers. Throughout the rest of this chapter, we use the terminology of maker and seller, taker and buyer interchangeably. Each buyer T_i requests an inbound liquidity demand of d_i units and holds a private valuation $v_i^b \geq 0$ for buying d_i units and a bid $b_i \geq 0$ as the maximum amount that it would pay for d_i units. Each seller M_j provides an inbound liquidity supply of s_j units and holds a private valuation $v_j^s \geq 0$ for selling one unit and an ask $a_j \geq 0$ as the minimum amount that it would sell one unit.

Table 6.1 Main notations

Notation	Meaning
$\mathcal{T}$	set of liquidity takers (buyers)
$\mathcal{M}$	set of liquidity makers (sellers)
$\mathcal{T}_w$	winning set of buyers
$\mathcal{M}_w$	winning set of sellers
d_i	buyer T_i's liquidity demand amount
b_i	buyer T_i's bid for d_i units of liquidity
v_i^b	buyer T_i's valuation for d_i units of liquidity
p_i^b	payment for buyer T_i for d_i units of liquidity
u_i^b	utility for buyer T_i
s_j	seller M_j's liquidity supply amount
a_j	seller M_j's ask for one unit of liquidity
v_j^s	seller M_j's valuation for one unit of liquidity
p_j^s	payment to seller M_j for one unit of liquidity
u_j^s	utility for seller M_j
x_{ij}	binary variable indicating if M_j sells d_i units to T_i

The incentive mechanism works as follows: after collecting the bids and asks privately from all buyers and sellers, the incentive mechanism decides the allocation for each buyer and seller. We use a binary variable x_{ij} to represent whether a buyer T_i's demand is fulfilled by a seller M_j, defined as:

$$x_{ij} = \begin{cases} 1, & \text{if } M_j \text{ sells } d_i \text{ units of liquidity to } T_i, \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

The incentive mechanism also computes the payment for each buyer and seller. A winning buyer T_i pays p_i^b for buying d_i units of inbound liquidity, and a winning seller M_j receives p_j^s for selling one unit of inbound liquidity. The total payment for seller M_j is $p_j^s \sum_{T_i \in \mathcal{T}} x_{ij} d_i$. Because a seller's sold liquidity cannot exceed its supply, we have

$$\sum_{T_i \in \mathcal{T}} x_{ij} d_i \leq s_j, \forall M_j \in \mathcal{M}. \tag{6.2}$$

Due to demand indivisibility, we have

$$\sum_{T_i \in \mathcal{T}} x_{ij} \leq 1, \forall T_i \in \mathcal{T}. \tag{6.3}$$

The utility of a buyer T_i is defined as follows:

$$u_i^b = \begin{cases} v_i^b - p_i^b, & \text{if } T_i \text{ wins,} \\ 0, & \text{otherwise.} \end{cases} \tag{6.4}$$

The utility of a seller M_j is defined as follows:

$$u_j^s = (p_j^s - v_j^s) \sum_{T_i \in \mathcal{T}} x_{ij} d_i, \tag{6.5}$$

6.3.4 Desired Properties

There are several desired properties for an incentive mechanism to satisfy:

- *Truthfulness*: an incentive mechanism is truthful if each buyer or seller obtains the highest utility by biding its true valuation of the resource.
- *Individual Rationality*: an incentive mechanism is individually rational if all buyers and sellers have non-negative utilities by revealing their true valuations.
- *Budget Balance*: an incentive mechanism is budget balanced if the auctioneer's profit is nonnegative, *i.e.*, the difference between the payments charged from buyers and paid to sellers is nonnegative.
- *Computational Efficiency*: an incentive mechanism is computationally efficient if it can be conducted within polynomial time.

6.4 An Incentive Mechanism for Crypto Capital Commitment in Channel Liquidity Marketplace

In this section, we design and analyze Cumulonimbus, an incentive mechanism for crypto capital commitment in payment channel networks. We first provide the high-level overview and intuition behind Cumulonimbus, and then follow the design goals that are outlined in Section 6.3.4 to provide a detailed incentive mechanism description.

6.5 Overview

Cumulonimbus consists of two stages: the winner selection stage and the pricing stage. The winner selection stage applies a linear-program-based mechanism, which introduces a virtual buyer to intensify the competition on the buyer side and guarantee budget balance. It first determines the winning buyers, and then determines the winning buyers. In the pricing stage, the incentive mechanism finds the VCG [63] payment for each winning buyer and seller to guarantee truthfulness. We present the detailed incentive mechanism in the following subsections.

6.6 Social Welfare Maximization

To motivate the participation of both buyers and sellers, we focus on maximizing the social welfare, *i.e.,* the summation of the payoff of the auctioneer and the utilities of all the buyers and sellers. If all the buyers and sellers bid truthfully, the maximal social welfare $W(\mathcal{T}, \mathcal{M})$ can be solved by the following integer program $P(\mathcal{T}, \mathcal{M})$:

$$\text{maximize } W(\mathcal{T}, \mathcal{M}) = \sum_{T_i \in \mathcal{T}} \sum_{M_i \in \mathcal{M}} (b_i - a_j d_i) x_{ij} \tag{6.6}$$

$$\text{s.t. } x_{ij} \in \{0, 1\}, \forall T_i \in \mathcal{T}, \forall M_j \in \mathcal{M}, \tag{6.7}$$

$$0 \leq \sum_{M_j \in \mathcal{M}} x_{ij} \leq 1, \forall T_i \in \mathcal{T}, \tag{6.8}$$

$$0 \leq \sum_{T_i \in \mathcal{T}} x_{ij} d_i \leq s_j, \forall M_j \in \mathcal{M}. \tag{6.9}$$

The first constraint represents that T_i either buys d_i units of liquidity from M_j or none. The second constraint represents demand indivisibility, such that T_i can buy liquidity from at most one seller. The third constraint represents supply divisibility, such that a seller's sold liquidity cannot exceed its supply. The social warfare maximization problem $P(\mathcal{T}, \mathcal{M})$ can be proven NP-hard by reducing the demand matching problem, which has been proved NP-hard [64].

Theorem 10. *The social warfare maximization problem $P(\mathcal{T}, \mathcal{M})$ is NP-hard.*

In order to design a computationally efficient mechanism, we can only resort $P(\mathcal{T},\mathcal{M})$ to its linear relaxation formulation $\hat{P}(\mathcal{T},\mathcal{M})$, where $\hat{x}_{i,j}$ is a fractional variable that represents the percentage of T_i's demand fulfilled by M_j:

$$\text{maximize } \hat{W}(\mathcal{T},\mathcal{M}) = \sum_{T_i\in\mathcal{T}}\sum_{M_i\in\mathcal{M}}(b_i - a_j d_i)\hat{x}_{ij} \tag{6.10}$$

$$\text{s.t. } 0 \le \hat{x}_{ij} \le 1, \forall T_i \in \mathcal{T}, \forall M_j \in \mathcal{M}, \tag{6.11}$$

$$0 \le \sum_{M_j\in\mathcal{M}} \hat{x}_{ij} \le 1, \forall T_i \in \mathcal{T}, \tag{6.12}$$

$$0 \le \sum_{T_i\in\mathcal{T}} \hat{x}_{ij} d_i \le s_j, \forall M_j \in \mathcal{M}. \tag{6.13}$$

6.7 Incentive Mechanism Design

In this section, we describe the details of Cumulonimbus, which are illustrated in Algorithms 12, 13, 14, and 15.

Algorithm 12: Cumulonimbus-Buyer Selection

Input: a buyer set $\mathcal{T}$, a seller set $\mathcal{M}$, and a virtual buyer T_q
Output: a winning buyer set $\mathcal{T}_w$

```
1 b_q ← ∞; d_q ← max{s_j | M_j ∈ M}; T_w ← ∅;
2 T_q ← the virtual buyer associated with b_q and d_q;
3 Solve linear program P̂(T ∪ {T_q}, M);
4 for T_i ∈ T do
5 |  for M_j ∈ M do
6 |  |  if x̂_ij = 1 then T_w ← T_w ∪ {T_i};
7 |  end
8 end
9 return T_w
```

Cumulonimbus-Buyer Selection (Algorithm 12) determines the winning buyers. In order to guarantee budget balance, Algorithm 12 introduces a virtual buyer to intensify the competition on the buyer side. The intuition behind the virtual buyer is to create imbalances between the supply availability and demand requirement [65]. Therefore, a virtual buyer T_q should have the largest demand and an unlimited bid, such that it can always occupy the largest liquidity supply. Specifically, we set T_q's bid as ∞ and set T_q's demand as $d_q = \max\{s_j|M_j \in \mathcal{M}\}$, which is the largest seller supply (Lines 1 to 2). Then, Algorithm 12 solves the linear program $\hat{P}(\mathcal{T}\cup\{T_q\},\mathcal{M})$ with the virtual buyer T_q, the buyer set $\mathcal{T}$, and the seller set $\mathcal{M}$ (Line 3). In the solution of $\hat{P}(\mathcal{T}\cup\{T_q\},\mathcal{M})$, if a buyer T_i's liquidity demand is fulfilled by exactly one seller, *i.e.*, $\exists M_j \in \mathcal{M}, \hat{x}_{ij} = 1$, then T_i wins (Line 6). Algorithm 12 generates the winning buyer set $\mathcal{T}_w$ by checking $\hat{x}_{i,j}$ for each $T_i \in \mathcal{T}$ and $M_j \in \mathcal{M}$.

Cumulonimbus-Seller Selection (Algorithm 13) determines the winning sellers, similarly as Algorithm 12. The main difference is that Algorithm 13 determines the winning sellers by solving the linear program $\hat{P}(\mathcal{T}_w, \mathcal{M})$ with the winning buyer set $\mathcal{T}_w$ and the seller set $\mathcal{M}$ (Line 1). In the solution of $\hat{P}(\mathcal{T}_w, \mathcal{M})$, if a seller M_j's liquidity supply can fully satisfy any buyer, *i.e.*, $\exists T_i \in \mathcal{T}, \hat{x}_{ij} = 1$, then M_j wins.

Cumulonimbus-Buyer Pricing (Algorithm 14) determines the payments for all the buyers. In order to guarantee truthfulness, Algorithm 14 computes the VCG [63] payment for each buyer. The intuition behind VCG is to charge each winning buyer how much its participation hurts the others. Thus, for each winning buyer $T_i \in \mathcal{T}_w$, Algorithm 14 solves the linear program $\hat{P}(\mathcal{T} \cup \{T_q\} \backslash \{T_i\}, \mathcal{M})$ with the virtual buyer T_q, the buyer set $\mathcal{T}$ excluding T_i, and the seller set $\mathcal{M}$ (Line 3). The difference between $\hat{W}(\mathcal{T} \cup \{T_q\} \backslash \{T_i\}, \mathcal{M})$ and $\hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M}) - b_i$ is the payment for a winning buyer T_i (Line 4). A losing buyer's payment is 0 (Line 1).

Algorithm 13: Cumulonimbus-Seller Selection

```
   Input: a winning buyer set T_w, and a seller set M
   Output: a winning seller set M_w
 1 Solve linear program P̂(T_w, M);
 2 M_w ← ∅; T_w ← ∅;
 3 for T_i ∈ T do
 4 |   for M_j ∈ M do
 5 |   |   if x̂_ij = 1 then
 6 |   |   |   M_w ← M_w ∪ {M_j}; T_w ← T_w ∪ {T_i};
 7 |   |   end
 8 |   end
 9 end
10 return M_w
```

Algorithm 14: Cumulonimbus-Buyer Pricing

```
   Input: a buyer set T, a seller set M, and a winning buyer set T_w
   Output: the payments for all buyers
 1 for T_i ∈ T do  p_i^b ← 0 ;
 2 for T_i ∈ T_w do
 3 |   Solve linear program P̂(T ∪ {T_q}\{T_i}, M);
 4 |   p_i^b ← b_i − Ŵ*(T ∪ {T_q}, M) + Ŵ*(T ∪ {T_q}\{T_i}, M);
 5 end
 6 return (p_1^b, ..., p_i^b, ..., p_n^b)
```

Cumulonimbus-Seller Pricing (Algorithm 15) computes the sellers' payments, similarly as Algorithm 14. The main difference is that, for each winning buyer $M_j \in \mathcal{M}_w$, it solves the linear program $\hat{P}(\mathcal{T}_w, \mathcal{M} \backslash \{M_j\})$ with the winning buyer set $\mathcal{T}_w$ and the seller set $\mathcal{M}$ excluding M_j (Line 3).

Algorithm 15: Cumulonimbus-Seller Pricing

Input: a winning buyer set $\mathcal{T}_w$, a seller set $\mathcal{M}$, and a winning buyer set $\mathcal{T}_w$

Output: the payments for all sellers

1 **for** $M_j \in \mathcal{M}$ **do** $p_i^s \leftarrow 0$;

2 **for** $M_j \in \mathcal{M}_w$ **do**

3 Solve linear program $\hat{P}(\mathcal{T}_w, \mathcal{M}\backslash\{M_j\})$;

4 $p_j^s \leftarrow \frac{a_j \sum_{T_i \in \mathcal{T}} d_i \hat{x}_{ij} + \hat{W}^*(\mathcal{T}_w, \mathcal{M}) - \hat{W}^*(\mathcal{T}_w, \mathcal{M}\backslash\{M_j\})}{\sum_{T_i \in \mathcal{T}} d_i \hat{x}_{ij}}$;

5 **end**

6 **return** $(p_1^s, \ldots, p_j^s, \ldots, p_n^m)$

6.8 Analysis

We prove that Cumulonimbus satisfies the desired properties introduced in Section 6.3.4.

Theorem 11. *Cumulonimbus satisfies truthfulness, individual rationality, budget balance, and computational efficiency for buyers and sellers.*

We prove Theorem 11 by the following lemmas.

Lemma 1. *Cumulonimbus is truthful for buyers.*

Proof. We prove that each buyer obtains the highest utility by biding its true valuation of the liquidity. Assume there exists a buyer T_i, whose utility is higher when bidding $b_i' \neq v_i^b$. Let ${u^*}_i^b$ and ${u'}_i^b$ denote T_i's utilities, and let $\hat{W}^*(\mathcal{T} \cup \{T_q\}, \mathcal{M})$ and $\hat{W}'(\mathcal{T} \cup \{T_q\}, \mathcal{M})$ denote the solutions of $\hat{P}(\mathcal{T} \cup \{T_q\}, \mathcal{M})$, when T_i bids v_i^b and b_i', respectively. Since $p_i^b = v_i^b - \hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M}) + \hat{W}(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M})$, we have $u_i^b = v_i^b - p_i^b = \hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M}) - \hat{W}(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M})$ according to Equation (6.4). Since ${u^*}_i^b < {u'}_i^b$, we have $\hat{W}^*(\mathcal{T} \cup \{T_q\}, \mathcal{M}) - \hat{W}^*(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M}) < \hat{W}'(\mathcal{T} \cup \{T_q\}, \mathcal{M}) - \hat{W}^*(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M})$. Thus, $\hat{W}^*(\mathcal{T} \cup \{T_q\}, \mathcal{M}) < \hat{W}'(\mathcal{T} \cup \{T_q\}, \mathcal{M})$. This contradicts that $\hat{W}^*(\mathcal{T} \cup \{T_q\}, \mathcal{M})$ is the optimal solution that maximizes $\hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M})$. Therefore, if a buyer bids the true valuation of the liquidity, its utility will not be less than that when it lies. ∎

Lemma 2. *Cumulonimbus is individually rational for buyers.*

Proof. Assume that each buyer T_i bids truthfully, *i.e.*, $b_i = v_i^b$. For each winning buyer T_i, its payment is $p_i^b = v_i^b - \hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M}) + \hat{W}(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M})$. According to Equation (6.4), we have $u_i^b = v_i^b - p_i^b = \hat{W}(\mathcal{T} \cup \{T_q\}, \mathcal{M}) \geq 0 - \hat{W}(\mathcal{T} \cup \{T_q\}\backslash\{T_i\}, \mathcal{M})$. Therefore, $u_i^b \geq 0$, for all winning buyers. For a losing buyer, $u_i^b = 0$. Thus, $u_i^b \geq 0$. Cumulonimbus is individually rational for all buyers. ∎

Lemma 3. *Cumulonimbus is truthful for sellers.*

Lemma 4. *Cumulonimbus is individually rational for sellers.*

Lemmas 3 and 4 can be proved similarly as Lemmas 1 and 2. It is trivial to prove the computational efficiency. We focus on proving budget balance in the following.

Lemma 5. *Cumulonimbus is budget balanced.*

Proof. We first consider the buyer side and calculate a lower bound on the payments charged from the buyers. Let $a_{[k]}$ denote the seller's ask bid for the k-th lowest liquidity unit. Thus, the lower bound of all buyer payments is $a_{[\sum_{T_i \in \mathcal{T}_w} d_i + d_q]} \sum_{T_i \in \mathcal{T}_w} d_i$. Then, we consider the seller side and calculate an upper bound on the payments paid to the sellers. Because the highest price is no more than $a_{[\sum_{T_i \in \mathcal{T}_w} d_i]}$, the upper bound is $a_{[\sum_{T_i \in \mathcal{T}_w} d_i]} \sum_{T_i \in \mathcal{T}_w} d_i$. Since $a_{[\sum_{T_i \in \mathcal{T}_w} d_i + d_q]} \geq a_{[\sum_{T_i \in \mathcal{T}_w} d_i]}$, the difference between the payments charged from buyers and paid to sellers is nonnegative. Thus, Cumulonimbus is budget balanced. ■

6.9 Performance Evaluation

In this section, we evaluate the performance of Cumulonimbus. As we surveyed in Section 6.2, there is no existing truthful incentive mechanism designed for crypto capital commitment. Therefore, we demonstrate the effectiveness of Cumulonimbus by comparing it to Lightning Pool [54], a state-of-the0art channel liquidity marketplace (CLM) mechanism. Lightning Pool adopts a greedy matching algorithm to select liquidity takers and makers and adopts a uniform pricing rule.

6.9.1 Environment Setup

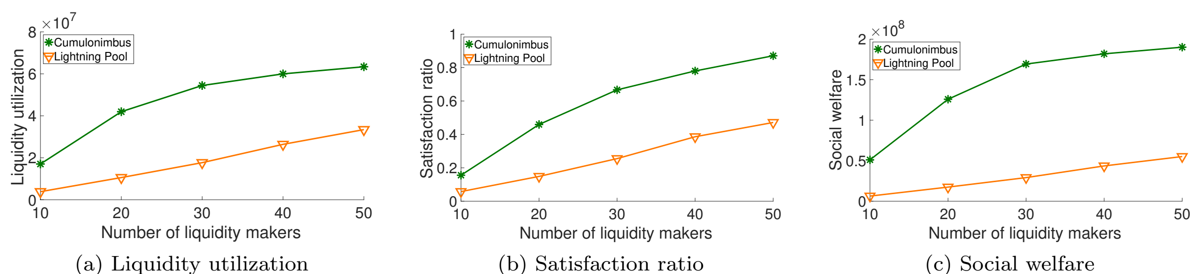
We use the liquidity demand information from the Bitcoin Lightning Network [36], which is the most widely used real-world PCN. In particular, we crawled a snapshot topology of the Lightning Network on September 6, 2021. To crawl the Lightning Network, we ran the Bitcoin Core daemon (bitcoind) [41], built a c-lightning [42] node on mainnet, and connected it to an existing Lightning node, which is the Bitstamp's Lightning Network node [43]. The network consists of $15,413$ nodes and $65,505$ channels. We use the liquidity supply information from the Bitcoin [66], which is the most widely used real-world blockchain-based cryptocurrency.

6.9.2 Performance Metrics

We use the following metrics for performance evaluation:

- *Liquidity utilization:* The total of liquidity allocated from liquidity makers to liquidity takers.
- *Satisfaction ratio:* The percentage of liquidity takers whose inbound liquidity demands are fully satisfied.
- *Social welfare:* The summation of the auctioneer's payoff and all the participants' utilities.

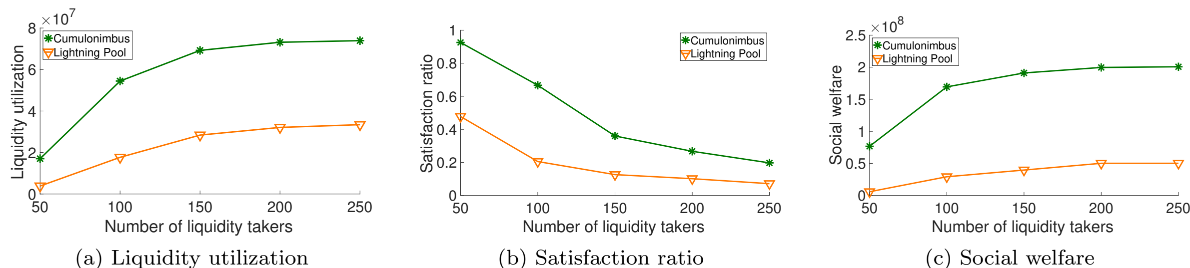
6.9.3 Evaluation of Cumulonimbus



(a) Liquidity utilization (b) Satisfaction ratio (c) Social welfare

Figure 6.2 Impact of number of liquidity makers on Cumulonimbus and Lightning Pool.

Figure 6.2 shows the impact of the number of liquidity makers on the liquidity utilizations, satisfaction ratios, and social welfares of Cumulonimbus and Lightning Pool. The number of liquidity takers is 100 and the number of liquidity makers varies from 10 to 50 with an increment of 10. It can be observed that Cumulonimbus outperforms Lightning Pool due to its demand indivisibility and supply divisibility guarantees. We can witness the growing gap between Cumulonimbus and Lightning Pool, which indicates that the greedy allocation without considering the demand indivisibility and supply divisibility decreases the satisfaction ratio and liquidity utilization significantly. Cumulonimbus outperforms Lightning Pool due to its maximization on social welfare. Lightning Pool gives a lower social welfare, because it applies a uniform pricing rule.



(a) Liquidity utilization (b) Satisfaction ratio (c) Social welfare

Figure 6.3 Impact of number of liquidity takers on Cumulonimbus and Lightning Pool.

Figure 6.3 shows the impact of the number of liquidity takers on the liquidity utilizations, satisfaction ratios, and social welfares of Cumulonimbus and Lightning Pool. The number of liquidity takers is 50 and the number of liquidity takers varies from 50 to 250 with an increment of 50. In Figure 6.3(b), it can be observed that both mechanisms have dropping satisfaction ratios with more liquidity takers. This is because the liquidity makers cannot provide sufficient supply to satisfy all the liquidity takers. Figure 6.3(a) shows that Lightning Pool gives a lower liquidity utilization, because it adopts greedy allocation algorithm without considering demand indivisibility and supply divisibility. Figure 6.3(c) indicates that

Cumulonimbus outperforms Lightning Pool, because Cumulonimbus aims to maximize social welfare.

6.10 Conclusion

In this chapter, we designed an incentive mechanism Cumulonimbus for crypto capital commitment in PCNs, which motivates participants to sell and buy inbound liquidity, while guaranteeing demand indivisibility and supply divisibility. We analyzed Cumulonimbus and proved that it satisfies truthfulness, individual rationality, budget balance, and computational efficiency. Extensive simulations demonstrated that Cumulonimbus achieved outstanding satisfaction ratio, liquidity utilization, and social welfare compared to a state-of-the-art mechanism Lightning Pool.

REFERENCES

[1] David Chaum. Blind signatures for untraceable payments (1983). In *Advances in Cryptology*, 1982.

[2] Laurie Law, Susan Sabett, and Jerry Solinas. How to make a mint: the cryptography of anonymous electronic cash. *Am. UL Rev.*, 46:1131, 1996.

[3] L Lamport, R Shostak, and M Pease. The byzantine generals problem acm transactions on progamming languages and syetems, vol. 4 no. 3 pp382-401, 1982.

[4] Nick Szabo. Secure property titles with owner authority. *Online at http://szabo. best. vwh. net/securetitle. html*, 1998.

[5] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed computing*, 11(4):203–213, 1998.

[6] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008.

[8] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[9] Stuart Haber and W Stornetta. How to time-stamp a digital document, crypto90, lncs 537, 1991.

[10] Henri Massias, X Serret Avila, and J-J Quisquater. Design of a secure timestamping service with minimal trust requirement. In *the 20th Symposium on Information Theory in the Benelux*. Citeseer, 1999.

[11] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[12] Ripple. URL https://www.ripple.com/.

[13] Ethereum project. URL https://www.ethereum.org/.

[14] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *SSS*, pages 3–18. Springer, 2015.

[16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *FC*, pages 106–125. Springer, 2016.

[17] Manny Trillo. Stress test prepares visanet for the most wonderful time of the year, 2013.

[18] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016. URL lightning.network/lightning-network-paper.pdf.

[19] Raiden network. URL https://raiden.network/.

[20] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *ICC*, pages 1–6. IEEE, 2019.

[21] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. SilentWhispers: Enforcing security and privacy in decentralized credit networks. In *IACR Cryptology ePrint Archive*, pages 1054–1071, 2016.

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

[23] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. In *Whitepaper*, 2016.

[24] Elias Rohrer, Jann-Frederik Laß, and Florian Tschorsch. Towards a concurrent and distributed route selection for payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 411–419. Springer, 2017.

[25] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks. In *ICCCN*, pages 1–9. IEEE, 2018.

[26] Paul F Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very ltaiarge networks. In *SIGCOMM*, volume 18, pages 35–42. ACM, 1988.

[27] Christos H Papadimitriou and David Ratajczak. On a conjecture related to geometric routing. In *Theoretical Computer Science*, volume 344, pages 3–14. Elsevier, 2005.

[28] Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. Towards an economic analysis of routing in payment channel networks. In *SERIAL*. ACM, 2017.

[29] Richard Bellman. On a routing problem. In *Quarterly of applied mathematics*, volume 16, pages 87–90, 1958.

[30] Lester R Ford Jr. Network flow theory. Technical report, RAND CORP SANTA MONICA CA, 1956.

[31] Yuhui Zhang and Dejun Yang. Robustpay: Robust payment routing protocol in blockchain-based payment channel networks. In *ICNP*, pages 1–4. IEEE, 2019.

[32] Yuhui Zhang and Dejun Yang. RobustPay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks. *IEEE/ACM Transactions on Networking*, 2021.

[33] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. Enforcing private data usage control with blockchain and attested off-chain contract execution. *arXiv preprint arXiv:1904.07275*, 2019.

[34] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *arXiv preprint arXiv:1904.04098*, 2019.

[35] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. Routing cryptocurrency with the spider network. In *HotNets*, pages 29–35. ACM, 2018.

[36] The Lightning Network. URL https://lightning.network/.

[37] Vivek Bagaria, Joachim Neu, and David Tse. Boomerang: Redundancy improves latency and throughput in payment networks. *arXiv preprint arXiv:1910.01834*, 2019.

[38] Chung-Lun Li, S Thomas McCormick, and David Simchi-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics*, 26(1):105–115, 1990.

[39] John W Suurballe and Robert Endre Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[40] Lightning network dataset. URL https://people.mines.edu/djyang/research/project-pcn/.

[41] Bitcoin core daemon (bitcoind), . URL https://github.com/bitcoin/bitcoin/.

[42] c-lightning daemon. URL https://github.com/ElementsProject/lightning/tree/master/lightningd/.

[43] Bitstamp's lightning network node, . URL https://www.bitstamp.net/lightning-network-node/.

[44] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *NSDI*, pages 777–796. USENIX, 2020.

[45] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *SIGSAC*, pages 439–453. ACM, 2017.

[46] Olaoluwa Osuntokun and Conner Fromknecht. Amp: Atomic multi-path payments over lightning. URL https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

[47] LND. LND: The Lightning Network Daemon, 2017. URL https://github.com/lightningnetwork/lnd.

[48] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In *CoNEXT*, pages 370–381. ACM, 2019.

[49] Lalitha Muthu Subramanian, Guruprasad Eswaraiah, and Roopa Vishwanathan. Rebalancing in acyclic payment networks. In *PST*, pages 1–5. IEEE, 2019.

[50] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *JACM*, 35 (4):921–940, 1988.

[51] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

[52] Charles Rackoff and Daniel R Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 433–444. Springer, 1991.

[53] Machine Learning Group ULB. Credit card fraud detection. URL https://www.kaggle.com/mlg-ulb/creditcardfraud.

[54] Olaoluwa Osuntokun, Conner Fromknecht, Wilmer Paulino, Oliver Gugger, and Johan Halseth. Lightning pool: A non-custodial channel lease marketplace. 2020.

[55] Lightning Labs. Lightning pool. URL https://lightning.engineering/pool/.

[56] Mo Dong, Qingkai Liang, Xiaozhou Li, and Junda Liu. Celer network: Bring internet scale to every blockchain. *arXiv preprint arXiv:1810.00037*, 2018.

[57] Tom Walther. An optimization model for multi-asset batch auctions with uniform clearing prices. In *Operations Research Proceedings 2018*, pages 225–231. Springer, 2019.

[58] ZmnSCPxj. Towards a market for liquidity providers enforcing minimum channel lifetime. URL https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-November/001555.html.

[59] Bitrefill. Thor: Lightning channel-opening service. URL https://www.bitrefill.com/thor-lightning-network-channels/?hl=en.

[60] Alex Bosworth. Yalls. URL https://yalls.org/about/.

[61] lnbig. URL https://lnbig.com/.

[62] Lightningto.me. URL https://lightningto.me/.

[63] Moshe Babaioff and Noam Nisan. Concurrent auctions across the supply chain. *Journal of Artificial Intelligence Research*, 21:595–629, 2004.

[64] F Bruce Shepherd and Adrian Vetta. The demand-matching problem. *Mathematics of Operations Research*, 32(3):563–578, 2007.

[65] Leon Yang Chu. Truthful bundle/multiunit double auctions. *Management Science*, 55(7):1184–1198, 2009.

[66] Blockchair. Bitcoin addresses and balances. URL https://gz.blockchair.com/bitcoin/addresses/.

[67] Yuhui Zhang, Dejun Yang, Guoliang Xue, and Ruozhou Yu. Counter-collusion smart contracts for watchtowers in payment channel networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[68] Fraud-fighting watchtowers to arrive in next bitcoin lightning release, . URL https://www.coindesk.com/fraud-fighting-watchtowers-are-coming-with-the-next-big-lightning-release.

[69] Bitcoin wallet electrum now supports lightning, watchtowers and submarine swaps, . URL https://www.coindesk.com/bitcoin-wallet-electrum-now-supports-lightning-watchtowers-and-submarine-swaps.

[70] Bitcoin lightning fraud? laolu is building a watchtower to fight it, . URL https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud.

[71] Blockstreams watchtowers will bring a new justice system to the lightning network, . URL https://www.coindesk.com/blockstreams-watchtowers-will-bring-a-new-justice-system-to-the-lightning-network.

[72] Olaoluwa Osuntokunn. Hardening lightning. *BPASE*, 2018.

[73] Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.

[74] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. In *AFT*, pages 31–40. ACM, 2019.

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

[76] Thaddeus Dryja and Scaling Bitcoin Milano. Unlinkable outsourced channel monitoring. *Talk transcript) https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring*, 2016.

[77] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *CCS*, pages 211–227. ACM, 2017.

[78] Kevin Leyton-Brown and Yoav Shoham. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis lectures on artificial intelligence and machine learning*, 2(1):1–88, 2008.

[79] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, pages 281–310. Springer, 2015.

[80] Cerberus script. URL https://github.com/OrfeasLitos/cerberus-script.

[81] Ethereum wiki: Geth, 2015. URL https://github.com/ethereum/go-ethereum/wiki/geth.