Design and Analysis of Routing Protocols in Payment Channel Networks in Blockchain Networks

Thesis submitted to the
Indian Institute of Technology Guwahati
for the award of the degree

of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by

Neeraj Sharma

Under the guidance of

Prof. Kalpesh Kapoor
Prof. Hemangee K. Kapoor

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

June 13, 2025

Dedicated to

Radharani

Declaration

I certify that:

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisors.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor is not in a position to check for any possible instance of plagiarism within this submitted work.

Date: June 13, 2025
Place: Guwahati

Neeraj

(Neeraj Sharma)

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisors *Dr. Kalpesh Kapoor* and *Dr. Hemangee K. Kapoor* for their consistent support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for their ethical beliefs and philosophy which made me mature not only as a scientific researcher but also as a human.

I am highly grateful to my wife *Pooja Sharma* for his invaluable support and encouragement throughout my Ph.D. I would also like to thank members of my Doctoral Committee - *Dr. Tamarapalli Venkatesh*, *Dr. Ramesh Kumar Sonkar* and *Dr. Ashok Singh Sairam* for their insightful comments and suggestions which made me improve the quality and clarity of my work.

I am also thankful to the anonymous reviewers of my research work in various forums for their critical comments which helped me to add quality to my work.

I want to thank the heads of the Department of Computer Science and Engineering during my PhD at IITG - *Prof. Diganta Goswami* and *Prof. Tamarapalli Venkatesh* for allowing me to use the facilities and the available resources. I acknowledge the Technical staff of the Department of Computer Science and Engineering - Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah, Mr. Hemanta Kumar Nath, Mr. Pranjitt Talukdar, Mr. Raktajit Pathak and Mr. Nava Kumar Boro for solving any engineering related issues. I am deeply thankful to - Mr. Gourish Mazumder, Ms. Mithu Lakra, Ms. Gauri Khuttiya Deori and Mr. Prabin Bharali for efficiently handling the administrative work. I am obliged to all the faculty members, the staff and security personnel for their constant help and support. I would also like to thank the Academic Affairs office staff who supported processing my applications and grant requests.

I would like to gratefully acknowledge *MHRD*, *Govt. of India* for the financial support rendered throughout my years of PhD without which this research could not have taken shape. I would also like to acknowledge the Department of Computer Science and Engineering and the Welfare Board of IITG for the travel grants that helped me present my research work at the national and international levels.

I am indeed thankful to my fellow lab mates. - Palash sir, Sheel sir, Arijit sir, Aswathy Di, Maithilee Di, Imli, Swati, and Rishab for creating a wonderful experience at my workplace. The stimulating discussions, brainstorming, and sleepless nights working together contributed a significant portion to my development as an independent researcher. I would also like to thank the research scholars of the Department of Computer Science and Engineering at IITG for creating a warm atmosphere of mutual support and encouragement.

I am fortunate to have good friends - Uday, Imli, and Swati with whom I have shared some ineffaceable moments of my life at IITG. I am thankful to all my colleagues and friends during my journey as a Ph.D. scholar.

I want to thank my friends from my school days - Kamal, Vivek, and Sidharth for the beautiful memories and cherished moments spent with them.

Finally, yet importantly, I would like to thank Almighty God and my family. They never doubted my intentions and wholeheartedly supported me in all my endeavours. I fall short of words to express my gratitude to them.

Certificate

This is to certify that this thesis entitled, **“Design and Analysis of Routing Protocols in Payment Channel Networks in Blockchain Networks”**, being submitted by **Neeraj Sharma**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for the award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of my knowledge, it has not been submitted elsewhere for the award of the degree.

Date: June 13, 2025
Place: Guwahati

Dr. Kalpesh Kapoor
Professor
Department of Mathematics
IIT Guwahati

Dr. Hemangee K. Kapoor
Professor
Department of Computer Science and Engineering
IIT Guwahati

Abstract

The Payment Channel Network, a layer-two solution, offers a scalable alternative for blockchain networks by enabling off-chain transactions between users, bypassing the delays and costs of on-chain settlements. However, the expanding size of networks like the Lightning Network poses challenges for source routing, which becomes impractical for large-scale implementations. Additionally, static distributed routing algorithms struggle to efficiently manage the increasing transaction volume. This thesis addresses these challenges by proposing novel distributed routing methodologies aimed at enhancing concurrency, channel balancing, transaction fees, and resolving deadlocks. It also investigates vulnerabilities to theft attacks in distributed routing systems.

We introduce the Swift routing system, which focuses on fee reduction through the use of a "cloud," a local graph maintained by non-landmark nodes to improve inter-subtree transaction efficiency. The system employs secret routing to distribute edge loads, enhancing throughput and achieving up to 21% fee optimization compared to state-of-the-art methods. Further, the theoretical reduction of network structures into cycle combinations enables a deeper analysis of fee optimization with varying cloud sizes.

To improve concurrency, we propose the maxECW and maxSCL techniques, which alleviate congestion and prevent channel exhaustion through self-rebalancing mechanisms and channel weighting. These solutions demonstrate a 50% improvement in processing concurrent transactions compared to existing methods. We also develop the maxREE technique, which further enhances concurrency by 8% relative to maxSCL and maxECW while significantly reducing overhead related to transaction route length, tree restructuring, and privacy concerns. Our proposed methods are inherently dynamic, updating the routing trees to enhance transaction flow while maintaining overhead within a certain range.

We note that the decision problem of the existence of deadlock-free routing for two transactions in PCN is NP-complete by relating it to the known two-commodity flow problem. The suggested DEPR routing technique avoids deadlocks, in contrast to previous methods that address deadlock by using locking mechanisms or priority queues. The suggested solution improves transaction count in Speedy and Webflow algorithms by 41% and 27%, respectively.

We have examined the effects of wormhole and flood-loot attacks on distributed routing systems and evaluated the attack gain, attack cost, and attack transaction ratio. We have positioned the attackers randomly within the network to assess the attack across several topologies. In the case of the wormhole attack, the average ATR is 41%, with 9.6% malicious nodes, while in the flood-loot attack, it is 16.23%, with malicious nodes ranging from 0.66% to 3.3%. The experiments have been done using an in-house simulator, DRLNsim, to assess our suggested algorithms in comparison to state-of-the-art solutions.

Contents

List of Figures

List of Tables

1 Introduction

Distributed Ledger Technology (DLT), often synonymous with blockchain technology, is a transformative approach to data management that eliminates the need for a central authority. By distributing data across multiple nodes in various geographical locations, DLT ensures resilience and transparency. Each node maintains a complete copy of the ledger, enabling a decentralized system that is less vulnerable to single points of failure.

The core components of DLT are peer-to-peer architecture, consensus mechanisms, and cryptography. Peer-to-peer architecture ensures direct communication between nodes without intermediaries, while consensus mechanisms enable agreement on the validity of transactions and updates. Cryptography secures these communications and protects data integrity, providing robust security against tampering and unauthorized access.

Consensus protocols are essential for maintaining consistency across the distributed network. They govern how nodes reach agreement on the ledger's state and significantly influence system performance, including scalability, fault tolerance, and transaction throughput. Widely used mechanisms include Proof of Work (PoW) [67], Proof of Stake (PoS) [31], Proof of Authority [5], and Proof of Elapsed Time [84]. Bitcoin's use of PoW to address the double-spending problem [67] sparked extensive research into blockchain-based consensus protocols.

DLT's architecture offers numerous benefits, such as enhanced transparency and reliability. Transaction data is accessible and verifiable by all participants, reducing the risk of manipulation. The system's ability to operate even when some nodes are offline or compromised underscores its robustness. Unlike traditional centralized systems, DLT decentralizes control, empowering nodes to function independently without reliance on intermediaries.

Although the concept of distributed data storage predates DLT, the technology advances this idea by implementing a decentralized ledger that operates without assuming trust among participants. Blockchain data structure [4] provides immutable and reliable record-keeping through its hash chain structure. In this design, each block incorporates the hash of the preceding block, ensuring that recorded data remains tamper-proof and trustworthy.

Systems like Bitcoin [67] have revolutionized digital payments and demonstrated blockchain's potential to transform financial services and various non-financial industries. DLT represents a paradigm shift in data management, offering a decentralized, transparent, and secure alternative to traditional centralized systems. Its potential to drive innovation across industries stresses the importance of understanding its foundational principles and exploring its diverse applications.

1.1 Scalability Challenges in Blockchain Networks

Blockchain networks, particularly Bitcoin, have faced significant scalability limitations, hindering their ability to handle a large volume of transactions efficiently. The throughput, or the number of transactions processed per second, of the Bitcoin network is notably low compared to traditional payment systems like VISA cards [105]. This disparity has become increasingly apparent as cryptocurrency adoption has grown.

The Nakamoto consensus protocol, utilized in the Bitcoin network, enabled Bitcoin to become the first decentralized digital currency capable of preventing double-spending attacks in a peer-to-peer network with minimal trust. However, as the network expanded, the protocol revealed performance limitations and sustainability challenges. Central to this is the PoW mechanism, which underpins mining – the process of validating transactions and adding them to the blockchain by solving computational puzzles. While PoW ensures security, it is resource-intensive and contributes to issues like unsustainable energy consumption, limited transaction throughput, inadequate scalability, and security vulnerabilities when mining incentives decrease.

One of the primary factors contributing to Bitcoin's low throughput is the size of its blocks and the lengthy block finalization time. The average block size of 1 MB [4] and the 10-minute block generation time [4] limit the number of transactions that can be processed within a given timeframe. While increasing block size and reducing block finalization time might seem like straightforward solutions, they are fraught with challenges [93]. Larger block sizes have led to contentious forks, demonstrating the complexities of network upgrades. Additionally, shorter finalization times can reduce mining difficulty, increasing the likelihood of multiple blocks being mined simultaneously, which can result in forks.

While the blockchain layer can function independently, its scalability is also constrained by the inherent limitations of its consensus mechanism. In Bitcoin, for instance, the consensus process requires approximately 10 minutes to generate a new block [4], resulting in relatively slow transaction confirmation times. This delay can be problematic for applications demanding frequent and rapid transactions. Furthermore, the blockchain's emphasis on security and decentralization contributes to its scalability limitations. The requirement for all nodes to verify and agree upon every transaction imposes significant computational overhead and slows down the network. Additionally, the energy-intensive mining process, designed to protect the network from attacks, further exacerbates these limitations.

Researchers have proposed several alternative strategies. Optimizing block capacity is one approach; for example, the SegWit soft fork increased Bitcoin's block capacity by four times without altering the block size [9]. Beyond this, the development of new consensus protocols offers a promising avenue for improving transaction throughput and overall network efficiency.

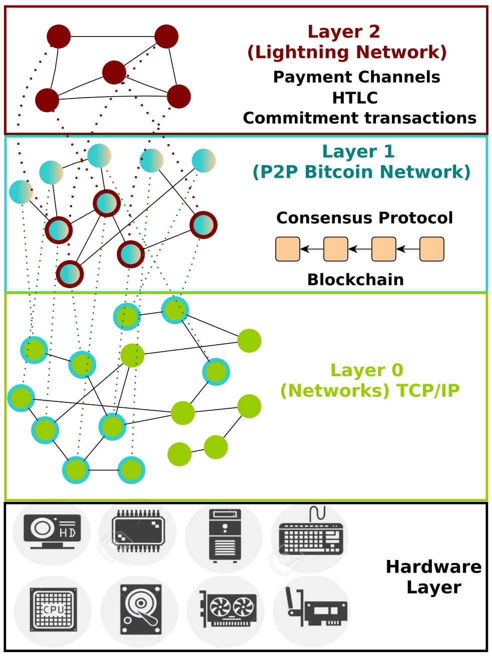


Figure 1.1: *Various layers of a blockchain network*

In response to broader scalability challenges, a multi-layered architecture has emerged in blockchain networks, with Layer-2 solutions playing a pivotal role. As illustrated in Figure 1.1, a typical blockchain network consists of several interdependent layers [70]. The foundation layer comprises the underlying hardware infrastructure, including standard computers, mobile devices, and specialized ASIC (Application Specific Integrated Circuit) [35]. Above this lies the network layer, where nodes communicate using TCP/IP protocols to form a decentralized peer-to-peer network. The blockchain layer, representing the distributed ledger itself, records and verifies transactions through a consensus protocol.

Layer-2 solutions have emerged as a vital approach to addressing the scalability challenges of blockchain networks. By enabling transactions to occur off-chain, these mechanisms reduce the computational and storage load on the main blockchain, thereby improving transaction speed and lowering costs. The Payment Channel Network (PCN) exemplifies this approach, allowing for faster and more efficient transactions while preserving the security and decentralization of the underlying system. A key focus of this thesis is the performance of PCNs, particularly the role of routing protocols, which are influenced by factors such as concurrency, cost optimization, deadlock prevention, and security concerns.

These innovative off-chain solutions not only alleviate the limitations of traditional blockchain systems but also enhance the overall user experience. By addressing core inefficiencies, mechanisms like PCNs pave the way for broader adoption and support the diverse applications of blockchain technology.

1.2 Payment Channel Networks

PCNs [76] are a Layer-2 solution that enable fast, low-cost transactions. A payment channel is a bilateral agreement between two parties, allowing them to exchange funds off-chain. This means transactions between the parties are not recorded on the main blockchain, thereby reducing congestion and fees. The technical architecture involves opening and closing channels, conducting off-chain transactions, and ensuring security through protocols and standards. Popular implementations include the Lightning Network [76] for Bitcoin and the Raiden Network [94] for Ethereum. This thesis focuses on the Lightning Network, although the techniques and analyses discussed are applicable to other PCN systems.

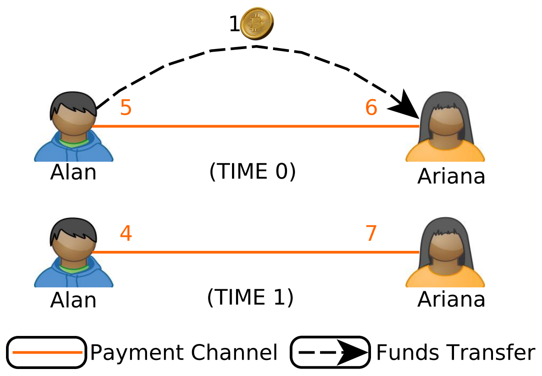


Figure 1.2: *Alan transferring* 1 *unit to Ariana*

A payment channel is established when two parties contribute funds to a shared pool. These funds are then locked within the channel and can only be used for transactions between the two participants. The channel capacity is the total amount of funds available for transactions.

Transactions within a payment channel are recorded in a sidechain or a separate database. Only the initial channel opening and the final channel closing transactions are recorded on the main blockchain. This significantly reduces the number of on-chain transactions, improving scalability.

Consider a payment channel between Alan and Ariana, as depicted in Figure 1.2. Alan contributes 5 to the channel, while Ariana contributes 6. Initially, each party is limited to sending only their own funds. The channel capacity is the total amount of funds available for transactions, which in this case is 11. Small-value transactions between Alan and Ariana can be conducted within the channel without being recorded on the blockchain. For example, if Alan sends Ariana 1 unit, their balances become 4 and 7, respectively.

The channel remains active until a closing transaction is broadcast to the main blockchain network. Only the initial channel opening and the final closing transactions are recorded on the main chain, significantly reducing the number of on-chain transactions. Funds locked within a payment channel are unavailable for use on the main blockchain network until the channel is closed.

PCNs can facilitate transactions between parties who do not have a direct payment channel established. This is achieved through a network, which is a collection of interconnected payment channels. These channels form a PCN, enabling multi-hop transactions that route payments through intermediaries. To ensure the security and atomicity of multi-hop transactions, several mechanisms are employed:

Hashed Time-Lock Contracts (HTLCs) [77] HTLCs provide a "timelock" feature that prohibits expenditure until a set time or block height is attained. This timelock can be set to expire after a certain amount of blocks have been appended to the chain. These contracts ensure that funds are either transferred in full or not at all. If the secret is revealed within the specified time, the funds can be claimed. Otherwise, they are refunded.

Commitment Transactions and Revocation Secrets [76] These mechanisms protect against malicious behavior by participants in the routing path. Commitment transactions are created to bind parties to their actions, while revocation secrets allow parties to cancel transactions if necessary.

Onion Routing [40] This protocol provides anonymity by encrypting the transaction data and routing it through multiple intermediaries, each of which only knows the next hop in the route.

In a multi-hop transaction, the sender creates a transaction with a hash of a secret value. This transaction is then forwarded through the network of intermediaries. The recipient can only claim the funds by revealing the secret within the specified time. If the secret is not revealed, the funds are refunded to the sender.

Consider the PCN in Figure 1.3, where Cora needs to transfer 1 unit to Beau through Dario and Dax. Beau begins by generating a random text, $\mathscr{R}$, and then transmitting the

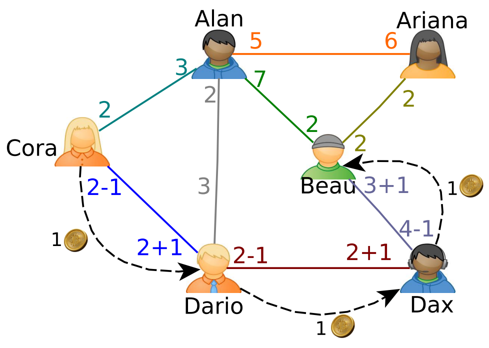


Figure 1.3: *A payment channel network*

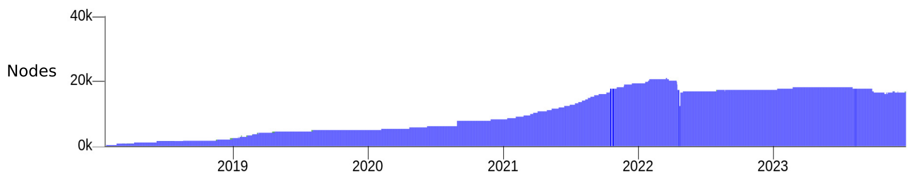
sender the string's hash value, $\mathscr{H}$. The double-signed transaction contract is then added by the sender and any intermediate peers for $\mathscr{H}$. After that, the recipient can only get the money if the sender has the recipient's secret $\mathscr{R}$.

It is important to note that intermediate nodes in multi-hop transactions may charge fees, which can reduce the amount of funds that ultimately reaches the recipient. Therefore, it is essential to consider these fees when planning multi-hop transactions.

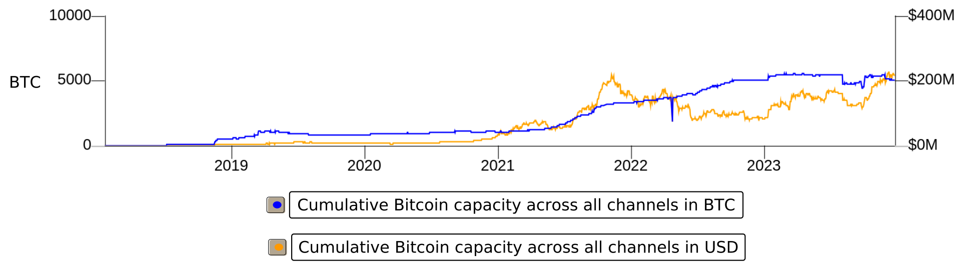
1.3 Motivation

Currently, Lightning Network employs source routing, where each node stores the complete network data locally and updates it using a gossip protocol. For routing, the source node finds a fee-optimal and reliable path to the destination using its local network. It then proceeds with onion routing to settle the transaction [76]. This approach is limited by the resource requirements for nodes to store the entire network and find optimal paths. Lightweight nodes, in particular, cannot store the complete topology. Moreover, routing concurrent transactions in source routing is challenging.

As the size of the Lightning Network continues to grow, as depicted in Figure 1.4 [49], source routing becomes increasingly infeasible for large-scale networks [83]. This necessitates the development of distributed routing algorithms. Existing distributed routing algorithms, such as Speedy [82], Silent [61], and Webflow [111], are static and do not adapt to changes in network parameters. By repeatedly utilizing the same static tree, these protocols, particularly Silent and Speedy, can lead to link bottlenecks during periods of high transaction volume. Many channels remain unused and are pruned from routing paths due to strategies employed by Silent [61] and others [104]. Buffer queues have been proposed to mitigate online rebalancing by managing channel in and outflow [99]. However, in scenarios with unidirectional traffic, buffer queues can lead to starvation and increased transaction latency. While learning techniques can improve routing [57], it comes at the cost of reduced network privacy. These methods are ill-suited for managing concurrent transactions due to the dy-



(a) *Nodes*



(b) *Channel capacity*

Figure 1.4: *Ligtning Network growth over years*

namic nature of the network. Additionally, existing algorithms do not adequately address potential issues like congestion attacks, structural attacks, and deadlocks during transaction routing.

This thesis aims to propose novel routing algorithms that enhance PCN routing, prioritizing low overhead and high performance. Beyond performance optimization, we delve into various network issues, including deadlocks, channel balancing, and security attacks, to further improve routing algorithms.

1.4 Thesis Objectives and Contributions

The objective of this research is to investigate the effectiveness of routing algorithms in enhancing transaction success within PCNs. An overview of our contributions is provided below.

(a) **Design and Evaluation of *Swift* Routing for PCNs:** We present Swift, a decentralized routing algorithm focused on fee optimization. Swift introduces the concept of a "secret path" to reduce the path length between sender and receiver, which minimizes transaction fees. The secret path also supports edge load sharing, thereby improving throughput.

(b) **Distributed Routing Algorithms for Concurrent Transactions in PCNs:** We propose two algorithms, $maxECW$ and $maxSCL$, designed to manage concurrent transactions more efficiently. These algorithms consider channel weights and introduce rebalancing mechanisms to prevent directional capacity saturation. $maxECW$ updates the routing tree greedily by adjusting channel weights in each iteration to select optimal channels, while $maxSCL$ alternates channel flow directions to alleviate congestion.

(c) ***maxREE:* Enhancing Transaction Flow via Edge Replacement in Lightning Network:** In this work, we introduce the $maxREE$ algorithm. This study extends our prior research and integrates the advantages of $maxECW$ and $maxSCL$ with negligible overhead. This algorithm substitutes exhausted edges with superior alternatives, thus preventing channel saturation while preserving the routing tree's structure. The self-rebalancing and link load-sharing features of $maxREE$ significantly enhance transaction flow.

(d) **Deadlock Prevention in PCNs:** We examine deadlock scenarios in distributed routing algorithms. We explore the "deadlock trilemma" and demonstrate that the D2TIR problem is NP-complete. The decision version of the safe routing problem for finding deadlock-free routes for transaction groups is also shown to be NP-complete. We introduce the DEPR approach that leverages significant nodes to facilitate transaction flow and maintain balanced channel capacities, thereby preventing deadlock.

(e) **Defense Against Attacks in Distributed Routing Protocols in PCNs:** We investigate vulnerabilities in PCNs with local knowledge-based routing algorithms. Malicious nodes can exploit these networks by sharing confidential information to gain an advantage. Specifically, we analyze the impact of wormhole and flood-loot attacks on the Swift algorithm, Speedy Murmurs, and Silent Whispers.

1.5 Organization of Thesis

The rest of the thesis is organized as follows:

Chapter 2 Provides background information and reviews relevant prior work.

Chapter 3 Proposes the concept of secret pathways based on cloud size, which minimizes path length and transaction fees. A theoretical analysis of the proposed algorithm using cycle topology and k-cycle graph is provided.

Chapter 4 Introduces two new algorithms, $maxECW$ and $maxSCL$, which leverage channel weights and degrees to enhance transaction feasibility. The chapter also presents the design of $DRLNsim$, a simulator used to evaluate the performance of routing tree algorithms for Lightning Networks.

Chapter 5 Presents the *maxREE* algorithm, which utilizes an edge coloring approach to preserve optimal channels in the routing tree. The chapter includes a theoretical analysis of congestion and load balancing.

Chapter 6 Examines the possibility of deadlocks in spanning tree algorithms using resource allocation graphs. The chapter analyzes the dynamic behavior of resources that can exacerbate deadlock scenarios, proves the NP-completeness of the D2TIR problem, and discusses the deadlock trilemma.

Chapter 7 Investigates potential attacks on distributed routing protocols, focusing on the impact of the wormhole effect on landmark routing protocols. The parameter D_r is used to model attacker deployment in the routing tree.

Chapter 8 Concludes the thesis.

2 Literature Survey

The blockchain trilemma, a widely recognized hypothesis, posits that blockchains can prioritize only two of the following three characteristics: decentralization, security, and scalability. While decentralization and security are fundamental to blockchain's essence, scalability remains a persistent challenge. Despite a decade of development, blockchain technology still lags behind traditional payment methods in terms of transaction speed and throughput.

To address scalability, it is crucial to understand the underlying factors that limit blockchain's performance, such as fixed block size, block creation time, and the linear structure of the blockchain (as illustrated in Figure 2.1). The fundamental consensus mechanisms that govern blockchain networks intrinsically link these factors.

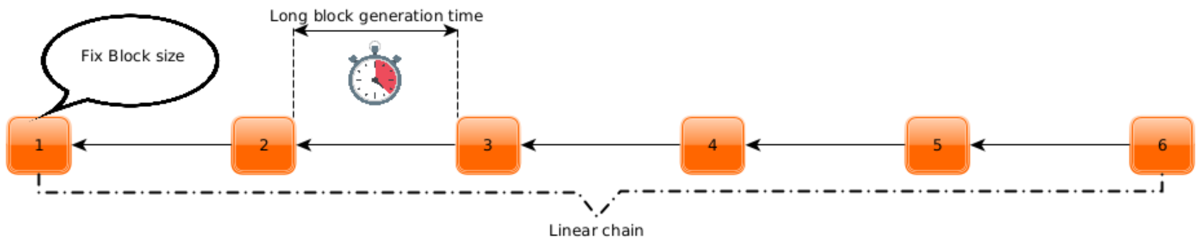


Figure 2.1: *Scalability challenges in Blockchain*

This chapter delves into various solutions designed to improve blockchain scalability. We begin by examining Proof of Work (PoW), the earliest consensus protocol deployed for DLT, followed by Layer-1 scalability solutions. Next, we will discuss the Layer-2 solution. The operation of PCNs and the current routing protocols within them will then be thoroughly covered.

2.1 Nakamoto Consensus

Nakamoto consensus [67], the foundation of Bitcoin, is a distributed consensus protocol that ensures the security and integrity of the blockchain. It relies on a Proof of Work (PoW) mechanism, where miners compete to solve complex cryptographic puzzles. The miner who successfully solves the puzzle first adds a new block to the blockchain and is rewarded with newly minted coins.

The concept of PoW can be traced back to the early days of the internet. In the 1990s, as email usage surged, so did the problem of spam. To mitigate this, the idea of pricing functions was introduced [23] where senders had to solve computational puzzles to reduce spam.

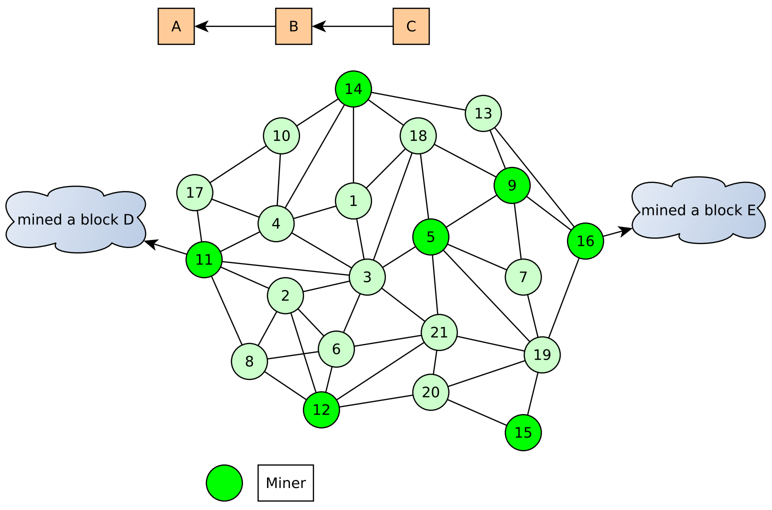


Figure 2.2: *Bitcoin network*

Building upon this concept, Adam Back proposed HASHCASH [7] in the late 1990s. The idea aimed to limit the abuse of shared resources by requiring users to perform computational work before accessing these resources. This early form of proof of work laid the groundwork for Nakamoto's groundbreaking work.

In Bitcoin, miners compete to solve a cryptographic puzzle, which involves finding a nonce value that, when combined with the block header, produces a hash value less than a target difficulty. This process requires significant computational power and energy.

The difficulty of the puzzle is adjusted periodically to maintain a target block time of approximately 10 minutes. This ensures a steady rate of block production and prevents the network from becoming overwhelmed or underutilized.

Components of Nakamoto Consensus

Transaction Validation Nodes validate transactions before including them in a block. This ensures that transactions are valid and follow the rules of the network.

Block Proposal Using PoW Miners compete to solve a cryptographic puzzle. The first miner to solve the puzzle adds a new block to the blockchain.

Block Propagation Once a block is mined, it is propagated across the network to other nodes.

Block Finalization A fork occurs when multiple blocks are mined nearly simultaneously. See, for example, Figure 2.2 in which miners 16 and 11 each mine and broadcast a new block at the same time, creating two blockchain branches on the network. Until a new block is mined, the network remains in an inconsistent state. If miner 5 then mines another block (see Figure 2.3), following the **longest chain rule**, all nodes will adopt the chain with the most accumulated work, thereby discarding the shorter branch (see Figure 2.4).

Incentive Mechanism Miners are rewarded with newly minted coins (block rewards) and transaction fees for successfully mining blocks.

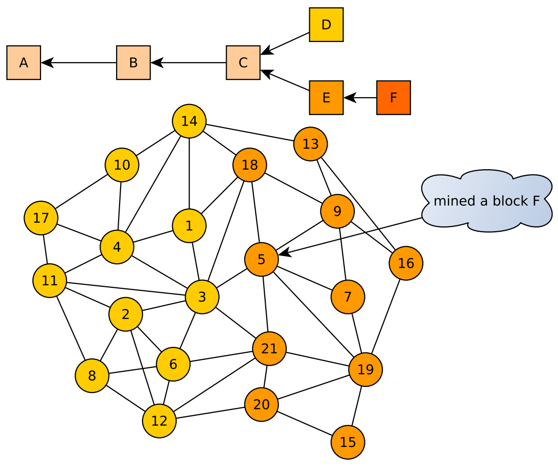


Figure 2.3: *Bitcoin's fork*

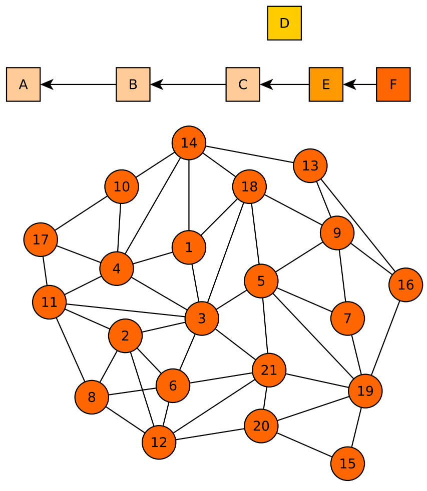


Figure 2.4: *Bitcoin's longest chain rule*

Additionally, the consensus mechanism includes features like transaction prioritization and orphan transaction pools to handle complex transaction scenarios. These components work together to ensure the security, decentralization, and integrity of the Bitcoin network.

The PoW mechanism in Nakamoto consensus helps prevent Sybil attacks [66] by requiring a costly computational process. PoW depends on the hardware capabilities of the miner's system. Classical distributed consensus protocols define fault tolerance based on the ability to tolerate a certain number of Byzantine faulty nodes. Garay et al. [29] have demonstrated that as long as malicious nodes control less than 50% of the network's hashing power, blocks generated by honest miners will propagate across the network.

2.2 Optimizing Scalability at Layer-1

Consensus rules in the Bitcoin network are essential for determining the validity of transactions and blocks. Over time, these rules may need to be modified to fix bugs or introduce new features. Changes to the existing consensus protocol can be implemented through two methods: a hard fork or a soft fork. A soft fork is not a full "fork" in the traditional sense. It is a term used to describe a change in the protocol that does not alter the consensus protocol. A hard fork occurs when the main blockchain splits into two separate chains due to a change in the consensus protocol. These two chains follow different consensus rules, which leads to a partitioning of the network into two distinct groups of nodes. A hard fork happens when a new software implementation with an updated consensus protocol is adopted by the majority of nodes in the network. Some nodes will adopt this new implementation, while others will continue following the old rules. As a result, nodes following the old rules will reject transactions from nodes using the new protocol, and vice versa, causing the network to split.

This partitioning also affects miners, who will now be divided between the two chains. Consequently, the hashing power will be split, and the block interval time will increase on both chains until 2016 blocks are generated. Hard forks are considered risky because they force the minority of nodes to either upgrade or continue using the old protocol. Additionally, hard forks are not forward-compatible, which is why many researchers prefer alternative mechanisms.

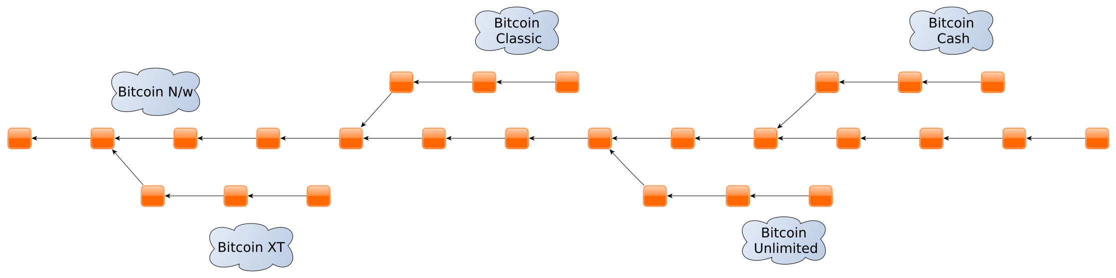


Figure 2.5: *Bitcoin hard forks*

Increase in Block size

As discussed earlier, the block size can limit transaction throughput, and one way to increase the block size is by performing a hard fork in the existing blockchain. Figure 2.5 shows all hard forks in Bitcoin from the augmentation in block size. An augmentation of block size in Bitcoin XT theoretically enhances throughput to 24 TPS (transactions per second) [38], in contrast to 7 TPS in Bitcoin. Bitcoin Cash has enhanced its capacity to 100 TPS [93]. However, none of the hard forks illustrated in Figure 2.5 has succeeded in terms of global merchant acceptance or attaining substantial throughput in practice. Another reasons for the failed hard fork attempt, aside from forward compatibility are as follows.

- Increasing the block size or reducing the block time will result in more frequent forks due to latency concerns, thereby causing more inefficiency in the network.

- The network's simplistic decision to augment the block size and diminish the block time to accelerate the overall chain will render the public network exceedingly inefficient, as it rapidly resolves the PoW algorithm while propagating slowly, ultimately leading to numerous forks that significantly benefit him. As a result, these networks exhibit considerably lower security and greater centralization compared to the Bitcoin network [93].

Alteration to Consensus Protocol

The following changes have been proposed to enhance the scalability of the Nakamoto consensus by optimizing fork chains and block generation time.

Ghost Rule In Nakamoto consensus, the 10-minute block interval significantly limits throughput. The Ghost rule [90], offers an alternative to the longest chain rule, allowing for a reduction in block interval time. Under this rule, orphan blocks are also considered part of the chain and contribute to the incentive structure. This modification increases transaction capacity and was later implemented in Ethereum. However, its execution presents obstacles such as increased complexity, elevated processing and storage requirements, and possible delays in transaction finality [30].

Bitcoin-NG Bitcoin-NG [90] introduces a two-phase block generation process consisting of key blocks and macroblocks. A key block does not contain transactions, so miners can mine a key block without waiting for transactions. Once a key block is mined, it becomes the leader and can generate macroblocks that contain the transactions. However, the current incentive analysis of Bitcoin-NG has numerous shortcomings. The influence of network capacity is overlooked. An integrated incentive analysis that simultaneously examines both the main blocks and the microblocks is still lacking [68].

Proposing a new Consensus Protocol

Researchers have been working on a number of projects to create new cryptocurrencies since the success of Bitcoin. A new cryptocurrency uses different consensus protocols or alters existing protocols. Some of the prominent consensus protocols are Proof of Stake [45, 105], Chain of Activity [10], hybrid consensus protocols [14, 32, 58, 71], For more details, refer to Appendix A.1. Consensus protocols exhibit two essential approaches for achieving consensus, as follows:.

Proof of Resource Ownership It permits any node to engage in consensus protocols at any moment, with the exception of certain specialized designs like proof of authority and proof of reputation. The benefit is the absence of authority, lack of authorization, and complete decentralization. This configuration additionally facilitates enhanced anonymity. The account is not linked to the owner's identity. A user may possess numerous accounts that remain uncensored. Nevertheless, increased decentralization typically results in diminished performance and extended confirmation latency. Another characteristic of proof-based protocols is that they may solely attain probabilistic consensus. The achieved consensus does not ensure unanimous endorsement from participating nodes. Every consensus round is probabilistic; however, an increased number of rounds results in heightened endorsement by participating nodes with enhanced certainty. The greater the depth of the deal, the lesser the likelihood of cancellation. Multiple rounds, as opposed to a single round, prolong the confirmation latency.

The Committee-based Protocols The committee-based protocols originate from traditional distributed consensus methods like Practical Byzantine Fault Tolerance. A predetermined number of people constitute a committee and achieve consensus. Prior to the initiation of the consensus protocols, the quantity of participating nodes is established, and the identities of these nodes are authenticated and kept transparent. Anonymity is disregarded in this instance, as there is consistently an entity or mechanism responsible for overseeing the integration of nodes. New nodes are unable to dynamically engage in the consensus process. Committee-based consensus procedures typically ensure that consensus is reached by participating nodes through a majority vote, hence achieving deterministic consistency. Upon inclusion in a block and subsequent addition to the blockchain, the transaction is deemed confirmed. In comparison to proof-based consensus systems, it can attain superior throughput and reduced confirmation delay.

PoW and PoS are predominantly utilized in consensus algorithms based on proof. Proof of Stake consensus protocols no longer necessitate the consumption of computational power. Nodes possessing certain cryptocurrencies or stakes can engage in the consensus mechanism, which is more accommodating to nodes with limited computational resources. Nonetheless, one could contend that the decentralization of PoS blockchains is inferior to that of PoW blockchains. A limited quantity of nodes possesses the majority of the cryptocurrency in a PoS blockchain system. Nodes possessing a greater value of cryptocurrency inside the PoS-based consensus process exhibit a higher probability of generating new blocks. Facilitating the accumulation of wealth among the affluent leads to centralization and security concerns. The stakeholders may mortgage or assign their stakes to other nodes for profit. It further undermines decentralization.

There are 10,309 cryptocurrencies in the world right now [22]. Bitcoin continues to lead the charts with 57% the market, followed by Ethereum with 12%, Tether (4.2%), XRP (3.8%), BNB (2.8%), Solana (2.7%), DOGE (1.3%), and USDC (1%). The remaining cryptocurrencies have a market share below 1% [87]. In the context of cryptocurrencies as a transfer medium, security, risk perceptions, performance expectation, effort expectancy, and social influence were essential aspects in its adoption [3]. Moreover, implementing Layer-1 scaling solutions frequently necessitates fundamental alterations to the blockchain protocol, potentially introducing complications and dangers. Altering fundamental protocol components necessitates meticulous planning and comprehensive testing to guarantee compatibility, stability, and security. The implementation of new features or consensus methods may present unanticipated risks or vulnerabilities, requiring comprehensive risk assessment and mitigation strategies.

Layer-2 scaling solutions function over the foundational Layer-1 protocol and seek to enhance blockchain scalability by processing transactions off-chain or via auxiliary protocols. In contrast to Layer-1 scaling solutions, which necessitate fundamental alterations to the underlying protocol, Layer-2 solutions prioritize scalability enhancement without directly altering the main blockchain design. Layer-2 solutions facilitate expedited and more efficient transaction processing by executing transactions off-chain and reconciling them on the main chain solely when required. This method alleviates congestion on the primary chain, enhances transaction throughput, and optimizes overall network performance.

2.3 Off-Chain Scaling

Layer-2 methods reduce the need for slow and expensive blockchains, which speeds up transaction processing times and fees. In reality, the main chain only sends a few transactions to the main chain. This is so that Layer-2 users can build trust and easily settle disagreements. As a result, Layer-2 blockchain protocols revolutionize the field. Some of the prominenet layer protocols are PCN, Commit chains [42, 43], Plazma [75], optimistic-rollups, zk-rollups, zk-SNARK [27], For more details, refer to A.2. Different Layer-2 solutions use different technologies and principles to give different functions while also trying to scale blockchains. There are a lot of things that make these solutions different.

When it comes to speed, PCNs offer faster transaction processing (that is, Layer-2 transfers that happen almost instantly) at a very low transaction cost. The basic ideas behind payment channels offer instant finality, while most other options offer delayed finality.

To take out amounts from channels, you only need one on-chain confirmation. Commit chains and Rollups usually make exit times longer to protect balances and settle disputes. Adding risk insurers or liquidity providers can lessen these kinds of long periods of time. On the other hand, there are worries about how reliable these providers are. In the best case, it takes less time to take out the balance on Layer-1 from *zk*-Rollups. You can change how final a transaction is on other Layer-2 solutions as well to give the customer instant confirmation, but only PCNs can offer full security guarantees [28].

Standard encryption primitives are used by most Layer-2 protocols. SNARKs and STARKs are used a lot in zero-knowledge protocols. The idea of user liveness is very important. This means that a user has to stay online to receive and confirm transactions, keep an eye on disputes, and deal with rude counterparties. Commit chains are different from payment channels in that they let users accept transactions even when they are not online. Users are told to check their checkpoint pledge every so often, though. This guess about a user's online presence can be given to a reliable third party that keeps an eye on activities for the user. Even so, this kind of third party might give in if the reason to break the rules is stronger than its guarantee deposit. All users can only safely withdraw in a short amount of time through commit chains. This is mostly for security reasons, and none of the protocols allow validators to take participants' funds.

For the reasons listed above, in order to continue our research, we concentrated on PCN, Layer-2 solution. In PCN, there is still room to enhance the scalability of distributed routing protocols. The next section will cover the operation of PCN and current routing algorithms.

2.4 Payment Channel Networks

In Chapter 1, we have discussed in brief; in this section, we will discuss in detail the working of PCN, a Layer-2 solution.

To reduce congestion on the main network, some transactions can be sent off-chain for settlement. Routine validation is limited to the transacting parties, with the global delayed conflict resolution method used only in the event of a dispute. To establish a payment channel, two parties deposit initial amounts and then update their coin balances. This occurs just within the channel, minimizing the number of minor transactions on the global network unless there is a dispute. Once the parties have completed their transactions, the channel is closed.

Channels offer a possible solution for blockchain scalability. Furthermore, they provide participants with privacy by not disclosing their transaction frequency or volume. Channels build a Layer-2 on top of the blockchain, requiring no changes to the fundamental protocol. Layer-2 assumes transaction integrity (only valid transactions included in the blockchain) and ultimate synchrony with an upper time bound (all valid transactions recorded to the ledger).

Bitcoin's Layer-2 protocol is named Lightning [76], while Ethereum's is called Raiden

[94]. Currently, three compatible implementations exist: LND, Eclair3, and C-Lightning. This thesis focuses on payment channels, which present unique research challenges in the field of networking. We will review their functioning, focusing on Lightning in particular. Payment channels are most effective for frequent, smaller-scale transactions between the same nodes. This sort of payment is also known as micropayments or "streaming" payments. Their qualities indicate that they are not worth the wait in the main chain and should be expedited. The lifecycle of the payment channel is illustrated in Figure 2.6.

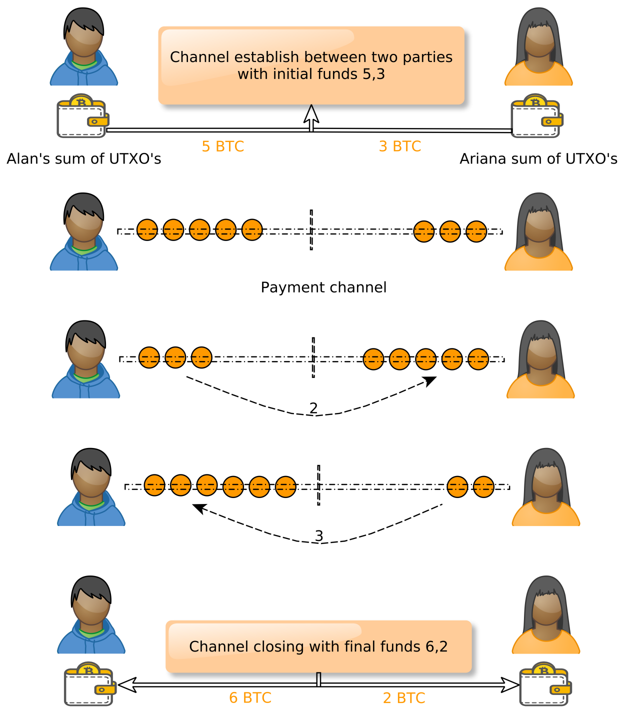

Channel establish between two parties with initial funds 5,3
5 BTC
3 BTC
Alan's sum of UTXO's
Ariana sum of UTXO's
Payment channel
2
3
Channel closing with final funds 6,2
6 BTC
2 BTC

Figure 2.6: *Payment channel lifecycle*

2.4.1 Channel Establishment and Working

The channel is created by a MULTISIG Bitcoin transaction, which is a transaction output necessitating signatures from several parties for expenditure. Two parties intending to establish a channel generate a transaction that they both endorse, in which they allocate beginning amounts. The total of these balances constitutes the channel capacity, which remains constant during the channel's lifetime. The transaction is submitted to the Bitcoin network, and parties must await confirmation through the standard protracted process. Upon confirmation, they alone exchange updates regarding the balances on both sides without engaging with the main chain (thus the phrase "off-chain transactions").

Both nodes endorse the updates, referred to as “commitment transactions,” which include a timelock: they cannot be redeemed prior to the expiration of the HTLC. If one party attempts to deceive, the other party can utilize the most recent update endorsed by both nodes, which constitutes a legitimate Bitcoin transaction, and submit it to the primary chain. Subsequent commitment transactions possess lower timelocks compared to their predecessors. Consequently, a node engaged in a channel may be assured that it will not forfeit any funds, even if the counterparty attempts to submit an earlier commitment transaction to the blockchain.

Consequently, the channel adheres to the “trust-free” idea of blockchain technology. When the nodes no longer require the channel, they terminate it by executing an additional on-chain transaction and consolidating the final channel balances into their on-chain balances. The funds a node possesses in a channel remain secured within the channel until its closure. This indicates that the node is unable to utilize them for another on-chain transaction or to create a new channel. Furthermore, funds cannot be transferred from their current channel to a different channel. A payment channel visually resembles a row of an abacus, where the overall number of beads remains constant, although the beads can be allocated in any configuration across the two sides of the row.

2.4.2 Network of Payment Channels

The network constituted by coin owners as nodes and payment channels as links is referred to as a Payment Channel Network (PCN). In contrast to the primary network, where any participant can transact with another provided they possess the recipient’s address (public key), a PCN does not constitute a complete graph. The reasons for this include the limited quantities available to the nodes on the blockchain, the locking of funds within a channel for its entire duration, and the infrequency or complete absence of transactions between certain nodes. These characteristics make the creation of channels between each pair of nodes wasteful and unnecessary. When a transaction is required between two nodes lacking a direct channel, two options are available: they can establish a new channel or attempt to route the payment across many existing channels, using the cash held by other nodes.

Multihop payments bring various new dimensions of PCNs. A multihop payment must occur atomically: either all steps will be successfully completed, or none will be executed. In Lightning channels, this is assured through the cryptographic mechanism of Hashed Time-Lock Contracts (HTLCs) [7], which facilitates the sequential execution of payments, thereby circumventing issues such as one node completing its payment while another node in the transaction pathway fails to cooperate and misappropriates cash. Secondly, certain route discovery tasks are required. This illustrates the necessity for payment routing within PCNs. Prior to the introduction of routing protocols in PCN, it is essential to delineate certain characteristics that contribute to successful payment.

Network Topology known to Nodes

Initially, we will delineate the knowledge possessed by the nodes inside the network. A channel is created using a standard Bitcoin transaction that resides on the main chain, making it accessible to anyone. The link topology of the PCN, along with the initial (but not the current) balances, is universally accessible, as it can be ascertained by merely analyzing the blockchain. The awareness of the starting balances also discloses the capacity of each channel, which is the aggregate of the beginning balances and, as previously stated, remains constant during the channel's duration. The nodes are unaware of the current channel balances for channels in which they are not involved. A node is aware of the current balances in her channels with neighbors; however, it lacks knowledge of the current balances of non-neighbor nodes other than the initial balances.

Incentives/Fee

The next feature required for multihop payments in PCNs is an incentivization mechanism for the intermediary nodes within a payment path. Be aware that the execution of a multihop payment entails multiple stages, and all participating nodes temporarily secure some amounts during this process until its conclusion. While they cannot forfeit their cash, they are also unable to utilize it for personal advantage, either within or beyond the channel, until they are liberated upon the successful or unsuccessful completion of the end-to-end payment. What motivates an intermediary node to facilitate payments by providing liquidity and securing its funds for the advantage of others? The response is relay fees. In the Lightning network, the time-value of fees compensates for the consumption of time (say, 3 days) and is conceptually analogous to a gold lease rate devoid of custodial risk; it represents the time-value for utilizing access to funds for a brief period [76]. Consequently, multihop payments function in the following manner: The initial payer transmits a somewhat greater sum through the payment pathway to the ultimate recipient, with the surplus amount appropriated as fees by the relays. Consequently, the aggregate balance of a node over all its channels will experience a marginal increase. The existence of fees raises several issues concerning incentives and network architecture.

Steps involved in Multi-hop Transaction

Examine the illustration presented in Figure 2.7, wherein node B desires to transmit a single unit to node F. According to the current method, F must generate a random string $\mathcal{R}$ and send the sender $\mathcal{H}$, which is the hash value. The path was found, and B opted to utilize route B→A→E→D→F for the payment. The sender and any intermediate peers sign the double-signed transaction contract along with $\mathcal{H}$. These changes will be implemented just if the payment is successfully completed in its entirety, atomically. To accomplish this, F must disclose the secret number R to all participants in the reverse sequence of the path. After that, the recipient will only be paid if the sender is able to acquire the secret $\mathcal{R}$ from

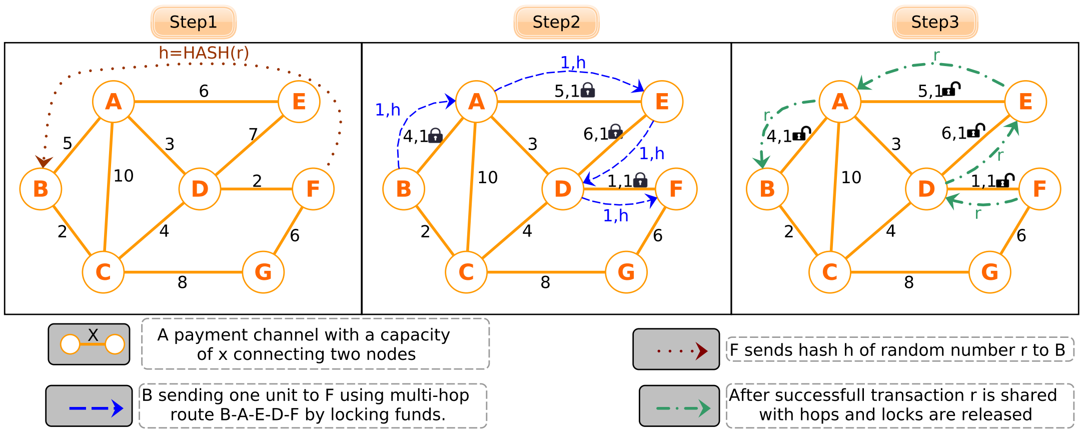

Step1
Step2
Step3
h=HASH(r)
A
B
C
D
E
F
G
1,h
5,1
4,1
6,1
1,1
r
A payment channel with a capacity of x connecting two nodes
B sending one unit to F using multi-hop route B-A-E-D-F by locking funds.
F sends hash h of random number r to B
After successfull transaction r is shared with hops and locks are released

Figure 2.7: *Steps in Multi-hop transaction in PCN*

the recipient. The multihop payment process is facilitated by onion routing, which involves the nested encryption of payment information for successive steps. This ensures that relay nodes receiving a payment are only aware of the sender and recipient they received it from, without disclosing the original sender and final recipient. In the unfortunate event that, for example, a node is uncooperative, the deadline's expiration will enable all other participants to withdraw their funds without incurring losses [28]. With an understanding of the precise functions of a payment channel and a PCN, we can now proceed to investigate the necessity of a routing protocol in this area.

2.5 Routing Protocols in PCN

Payment routing is of the utmost importance in PCNs, as it is a necessary operation whenever a payer and a payee are not directly connected. Depending on the necessary efficiency, speed, and privacy levels, routing can take on a variety of forms, as we explain below. Routing algorithms implemented in other forms of networks may serve as sources of inspiration and analogies. The balances on the route and whether they are known to the nodes, the relay fees, and the payment deadlines are all fundamental factors that influence the selection of a path.

There are two methods for routing in PCN, namely source and distributed. In the former, each network node is responsible for maintaining the global routing table and network topology. Since source routing protocols have complete knowledge of the underlying network, sending transactions from one node to another is simple. Such transaction routing can be modelled as a 1-commodity flow problem [98], which can be solved using existing Max-flow algorithms [33]. Such an approach has limitations, as a node in Bitcoin can be either complete or lightweight. The latter type, which often involves a device with constrained memory and computing capability, like a cell phone, does not store entire knowledge about the underlying network. In a Bitcoin network, it is anticipated that there will be a significant number of these lightweight nodes. The resources available with these nodes are insufficient to store the whole topology.

2.5.1 Source Routing Protocols

Flare

This routing algorithm for the Lightning Network employs a trustless source routing methodology [55]. The algorithm consists of two distinct stages: proactive and reactive.

- **Proactive component:** Every node in the network formulates a routing table. The routing table comprises a selection of payment channels that facilitates the identification of a path to the receiver. If the information is inadequate for route identification, the table is exploited to determine the beacon nodes that facilitate route discovery. At the outset, a node has a distinct perspective of the adjacent nodes. Beacon nodes enhance the visibility of a node's network by integrating random nodes, extending its reach beyond the immediate vicinity. Consequently, with a high degree of certainty, a node determines its capacity to access a certain node.
- **Reactive component:** Upon receiving a request to route a transaction to a certain payee, a node utilizes its routing table along with the payee's routing table to determine a pathway from the payer to the payee. In the absence of such routes, the beacon node associated with the payee, together with other nodes, can contribute their routing tables to establish a suitable path. If multiple routes exist, they can be prioritized accordingly. The ranking is established according to a specified cost function. The cost function is determined by static factors, such as route length, and dynamic factors, such as channel usage fees. According to the ranks designated for the routes, the sender identifies the most viable channel for transmitting its payment.

In this technique, each node monitors all neighbors at a hop distance and calculates the weights of the links within this k-neighborhood. Privacy is infringed upon. Furthermore, each credit modification must be conveyed inside the k-neighborhood, resulting in message flooding throughout the network, which is highly inefficient.

Flash

Flash addresses the issue by distinguishing between substantial payments, referred to as elephant payments, and minor payments, known as mouse payments [102]. A modified max-flow approach, derived from Edmond-Karp [24], is employed to identify k optimal pathways with adequate capacity for facilitating substantial elephant payments. An optimization software is developed to distribute payments among k pathways in order to minimize fees.

In mouse payments, each node sustains a routing table that determines the route to the recipient. Each payer determines m such that m is less than k for the shortest paths. If such paths are nonexistent, they can be dynamically generated utilizing the local topology and available routing information. The routing table is regularly updated according to alterations in the topology of the PCN. Upon identifying such paths, the sender attempts to transmit the entire payment amount using a single path. If unsuccessful, it assesses the remaining amount to be routed and attempts to route it through the next available path. This will delay transaction completion as path selection will take time. Probing for paths can take more than 5% of the attempted payments [74].

Spider

Spider [86] has been suggested to encapsulate transactions, divide them into many transaction parts, and transmit them through various routes at varying rates. Each transaction unit may be transmitted with onion encryption to obscure the complete route from intermediary routers. Dividing payments into segments facilitates the execution of substantial transactions on low-capacity payment channels incrementally. Congestion control across several pathways guarantees balanced channel utilization and equitable flow distribution.

Every payment channel possesses a router that manages price variables, which are periodically changed according to the current transaction unit arrival rate, the available channel balance, and the number of transactions presently queued in the router. Transaction requests accumulate in the router of each payment channel when the channel is insufficiently funded to process them promptly. The queuing of the transaction suggests that either the transaction is being handled too rapidly and requires a reduction in speed or the channel is insufficiently capacitated. If the node employs a congestion control protocol that manages queues, it may identify both capacity and imbalance infractions. Any conventional congestion management method employed for network routing can be modified for use in PCN. The routers track the duration each packet remains in its queue. If the duration surpasses a specific threshold, the packet is designated. If the packet is marked, it is subsequently forwarded. Upon receipt of the packet, the receiver transmits an acknowledgment to the sender, indicating the presence of a route. Spider [86] disregards the influence of limitations such as transaction fees and payment timeliness.

RobustPay$^+$

Zhang et al. [112] suggested CheaPay, which seeks to reduce the routing price for payments via a single path. The limitation is that the algorithm lacks resilience to transaction failures. A proficient routing algorithm has been suggested that is resilient to transaction failures while minimizing the maximum transaction fee for a certain payment. The protocol has three phases:

- **Path Construction:** Two node-disjoint paths are established for a specified payment to ensure robustness. The reasoning for this is because if one route fails, payment can be redirected through the alternative method. A distributed 2-approximation technique is employed to minimize the worst-case transaction charge for payment routing.

- **Establishment HTLC:** The HTLC is amended to facilitate the cancellation of payments. If the conditional payment is annulled or unmet in one of the pathways, the coins secured in the conditional payment through the second pathway are returned to the original sender. The alteration is executed through the application of two distinct preimages along each path corresponding to each hash value.
- **Payment Forwarding:** Following the establishment phase of the HTLC, if the payment is successfully transmitted through one way, the conditional payment through the alternate path must be rendered invalid. If the recipient attempts to claim both payments, the sender will initiate a refund on one of the transactions.

Another study [16] introduces an algorithm that identifies the most cost-effective route for a certain payment by formulating a transaction fee model.

2.5.2 Distributed Routing Protocols

Ant Routing

Ant routing [34] is a decentralized routing algorithm for Lightning Network that enables any node to trade by requiring knowledge solely about its close neighbors. The algorithm is modeled after the foraging behavior of ants in their habitat. Ants initiate their foraging in arbitrary directions and deposit pheromone trails that signal their route to food and facilitate their return. The authors are driven by the observation that natural selection has permitted the survival of just the most effective "food finding" algorithm ant species currently extant. The subsequent text succinctly elucidates the ant algorithm for transactions in the Lightning Network:

- The sender and receiver concur on a random number Q (seed) and securely exchange it with one another.
- The sender appends 0 to Q to create P (S) (pheromone seed for the sender) and transmits it to its neighbors in LN. Likewise, the receiver appends 1 to Q to create P (R) and transmits it to its neighbors.
- Each neighbor retains and transmits these pheromones onward.
- Ultimately, there will be nodes designated as matching nodes that will receive both pheromones, P(S) and P(R). The corresponding nodes would append 0 to P (S) and P (R) to generate M (S) and M (R) (matching seeds for sender and receiver, respectively) and transmit them back along the path from whence they were received.

- The sender and receiver would obtain these corresponding pheromones, enabling them to facilitate the transaction.
- Each node efficiently stores these pheromones only until a specified expiry timestamp via balanced binary search trees.
- Malicious nodes are penalized by excluding them from payment routing since nodes maintain a score based on their payment history with other nodes to facilitate transactions through trustworthy nodes.

This algorithm lacks a center point, and all nodes perform uniform tasks. Utilizing numerous matching nodes for algorithm routing enables the attainment of value privacy, as no intermediary node can deduce the total transaction amount. The ant routing algorithm possesses several drawbacks. The algorithm disregards any fee concerns during transaction routing. Pheromones are disseminated omnidirectionally, resulting in heightened message complexity within the network and an increased risk of privacy breach assaults. The storage of information pertaining solely to immediate neighbors by each node constitutes a significant underutilization of resources. Despite numerous efforts to address the routing issue, the suggested algorithms are deficient in several areas, including fee optimization and network overhead. Each routing method prioritizes the enhancement of a particular facet while neglecting others.

Silent Whispers

SilentWhispers [61] employs a distributed landmark routing-based methodology [100] for routing transactions. A set of landmark nodes has been identified. A highly connected node typically constitutes the landmark. A payer follows a path to a landmark, while a payee follows a path to a landmark. A route from payer to payee is constructed by concatenating both elements. The protocol's steps are as follows:

- **Routing:** Assuming the PCN network is a linked graph and the collection of landmarks is constant, each landmark periodically generates a spanning tree to reflect changes in the network. To construct the spanning tree, the landmark performs two instances of the Breadth-First Search (BFS) method, originating with itself. The initial instance computes the shortest path from the landmark to each node, referred to as arborescence, while the subsequent instance determines the shortest path from each node back to the landmark, known as anti-arborescence.
- **Determining credit on a path:** Multi-Party Computation is employed for the secure calculation of the credit accessible along a specific path. The landmarks are allocated a portion of the credit for each connection between the sender and the receiver. Based on this information, the collection of landmarks collaboratively calculates the credit for the full route. None of the landmarks get knowledge regarding the outcome of the computation or the attribution of credit for each connection.

- **Path construction:** In constructing the path, a user routing payment from sender to receiver employs a short-term verification key in conjunction with the long-term key. A user initially signs a new verification key with her long-term key and generates a sequence of signatures.
- **Accountability:** To adjudicate the conflict between two users of a link, the user logs are utilized to ascertain the accurate value of the link.

Speedy Murmur

SilentWhisper exhibits significant computational overhead, and the path generated by the algorithm may not be optimal. SpeedyMurmur [82] addresses the issue by considering the availability of funds in the channel and the proximity of a neighbor to a specified destination. This yields an efficient algorithm characterized by flexible path selection. The primary subroutines are outlined as follows:

- **establishRoutes:** Upon selection of a landmark, an empty vector is designated as its coordinate. The landmark identifies all neighboring nodes and assesses whether the connections to these nodes possess sufficient credit and have not been allocated any coordinates. If a node is identified, it is incorporated into the spanning tree. The coordinate assigned to a node is derived from the concatenation of the parent's coordinate and a random b-bit string. In an asynchronous context, a node establishes an upper limit on the time constraint. For a neighbor of a node to be incorporated into the spanning tree, it is required to transmit a message within the specified time constraint. Upon receiving such a request, the node designates this neighbor as its parent. A node initially includes all neighbors connected by bidirectional links with non-zero capacity. Subsequently, all nodes with unidirectional non-zero capacity links are included. The procedure is iterated for the k landmarks, yielding k spanning trees and corresponding coordinates for each spanning tree. This is referred to as prefix embedding. Let the coordinates with respect to landmark L_1 for nodes u and v be denoted as $id_1(u)$ and $id_1(v)$. The distance between nodes u and v is defined as follows:

  $$dist(id_1(u), id_1(v)) = |id_1(u)| + |id_1(v)| - 2cpl(id_1(u), id_1(v)) \tag{2.1}$$

  Here, $id_1(u)$ represents the coordinate length, while $cpl(id_1(u),\ id_1(v))$ indicates the length of the common prefix, which refers to the matching portion of the string assigned to each node u and v. The common prefix indicates a shared ancestor. This value must be deducted to prevent double counting.

- **setCred:** A pair of nodes may require adjustments to the capacity of the link (or channel) connecting them. Subsequently, they examine whether this alteration in capacity results in a change in the coordinate. This paper outlines scenarios in which the coordinates may change.

  - New non-zero link: For a given pair of nodes, if one node is not part of the tree, it designates the other node as its parent.
  - New non-zero link: For nodes(u, v), if u possesses a parent with a unidirectional link and subsequently v establishes a bidirectional link with u, then u is required to update its parent to v.
  - When a node loses its connection to its current parent (that is, the credit in the link decreases to 0), it must seek a new parent.

  When a node alters its parent, its descendants are required to experience a corresponding change in coordinates. Their neighbors are informed about the deletion of the existing coordinates. A new parent is subsequently selected, followed by a coordinate reassignment. A node that receives multiple requests from neighboring potential parents selects the one with the shortest path to the landmark. The configuration change is documented for each landmark node.

- **routePay:** To facilitate a payment of value v from the payer to the payee, the payer divides the total transaction value into k shares, denoted as $v_1, v_2, ..., v_k$, corresponding to each of the k landmarks. Paths must be identified for transmitting each payment, resulting in the generation of k distinct addresses for the receiver, denoted as $dest_i(recv)$, where $i \in [1, k]$. To determine the share v_i, the path must be identified based on the coordinates assigned using the reference landmark L_i, where $i \in [1, k]$. Beginning with the payer, a neighboring node, denoted as m, is chosen based on the criterion that the distance to the destination node is minimized, specifically where $d(id_i(m), dest_i(recv))$ is at a minimum, and the link possesses a capacity least. v_i. This process is repeated for the remaining $k - 1$ shares. In the event of routing failure, the route-search procedure is reiterated for an alternative distribution of the transaction value.

Coinexpress & Webflow

For the purpose of resolving concurrency among several simultaneous requests in a particular channel, it employs a locking approach and makes use of the distributed Ford-Fulkerson algorithm [106]. One of the drawbacks of the algorithm is that it does not protect users' privacy. This is because intermediary parties that route payments are able to learn about the channels and their residual balances when they participate in such a path.

WebFlow [111] is a coordinate-based geographic routing mechanism for PCNs. It enables each node to compute a set of Euclidean coordinates and employs these coordinates to execute coordinate-based greedy routing. We develop a method wherein nodes sustain a multi-hop Delaunay triangulation (MDT) [47] utilizing exclusively channels with trusted users to attain a high success rate in pathfinding. To enhance the anonymity of senders and recipients, our significant invention is utilizing the characteristics of a distributed Voronoi diagram to accomplish routing tasks. Unlike conventional greedy routing, it does not necessitate the coordinates of the destination in a routing message. Instead, it introduces a directional vector to ascertain the trajectory that conceals the true destination.

Summary to Routing

Due to the possibility that certain nodes may not have access to the entire topology of the lighting network, a centralized method cannot be implemented. Thus, in a realistic situation, distributed protocols are preferable. The landmark route was first proposed by the Silent Whisper [61] for PCN. A Breath-first search (BFS) is performed twice during landmark routing, using forwarding and reversing edges. As PCN is dynamic, the landmark nodes occasionally carry out this task. First, utilizing BFS and all accessible landmark nodes, the shortest path is determined from a source to a landmark and from there to a destination. Second, the maximum transaction value that may be routed through each path, that is, its transaction capacity, is determined. In the end, a single route is chosen.

Compared to Silent Whisper, Speedy Murmurs [82] reduces the overhead of managing links by taking advantage of embedded-based routing for PCN. Assign coordinates to each network node in embedding-based routing so that nodes may determine the shortest path based on these coordinates. If there is no shortcut, it will adhere to a tree-based routing system that resembles landmark routing. Webflow [111] takes further steps to shorten transaction paths by avoiding landmark nodes. During a transaction, Webflow uses Multihop Delaunary Triangulation (MDT), in which landmark nodes are utilized to distribute virtual coordinates and determine the shortest route between the sender and the recipient.

Wo et al. [104] employed a pooling and pruning method to eliminate nodes that are not needed for routing. In practice, these techniques rely on the concept of supernodes. However, a longer routing path and a larger transaction cost may follow from the fewer nodes. Varma et al. [99] recommended buffer queues to reduce the requirement of online rebalancing to aid routing, although they cannot handle all types of flows. Additionally, buffer overflow might result in transaction loss when there is a lot of network traffic. Luo et al. [57] developed a strategy to control the transaction flow to reduce the latency of the query on the intermediate nodes. Deep Q-learning balances the cost of the forwarding charge and network throughput. Chen et al. [16] improved the directed Dijkstra approach to find the optimum path. A PCN operator oversees the whole network and chooses the optimal route for users. The imbalance penalty and the time value cost are new additions to model fees, and they depend on the distance between the intermediate node and the destination node. These express the price intermediary nodes are prepared to pay to reach the best desirable condition for a payment channel. We have examined several more Layer-1 and Layer-2 methods that enhance scalability during the preliminary research phase, which can be referenced in the Appendix A. Next, we will discuss different aspects with their scope of improvement in routing protocols.

2.5.3 Routing Protocol Elements

Routing may exhibit numerous variations, as detailed below, contingent upon the requisite efficiency, speed, and privacy levels. Analogies and insights can be derived from routing algorithms utilized in various network types. Key criteria influencing the selection of a path are the balances along the route and their visibility to the nodes, the relay fees, and the payment deadlines. Essential criteria for building a routing protocol encompass the autonomy of each node to operate independently, adaptability to network changes, trustlessness, and resistance against nodes exhibiting arbitrary or malicious behavior.

Fee optimization

Silent Whispers and Speedy Murmurs [82] use a landmark-based approach to route transactions in PCNs. Speedy Murmurs uses the VOUTE [81] algorithm, primarily developed for F2F networks, to build embedded coordinate systems. Further, VOUTE presents a strategy to calculate the distance between nodes using hashes of the coordinates without revealing the actual coordinates, hence protecting the identity of a node. Speedy also optimizes routes with the same subtree of the spanning tree compared to Silent Whispers.

Speedy Murmurs has certain limitations. Though the algorithm can find a route for the transaction, it does not optimize the transaction fees while discovering a route. It allows the sender to put a cap on the transaction fee, but this is not an optimization. Speedy Murmurs will only require nodes to store their coordinates and information about subtrees, which may lead to poor utilization of resources in some nodes.

Ant routing [34] algorithm does not take into account any fee considerations while routing a transaction. Pheromones are broadcasted in every direction, leading to an increase in message complexity in the network and the possibility of privacy violation attacks. Each node only storing the information of its immediate neighbor is an underutilization of its resources.

The current state-of-art leaves scope for further improvement in routing algorithms, particularly fee optimization and network overhead. Chapter 3 attempts to address fee optimization.

Concurrency

Routing, as articulated, solely takes into account a single payment within the network. However, when numerous simultaneous payments occur, as is often the case in reality, if many of them traverse the same route, certain paths may get crowded. Given that payments have specific deadlines for successful completion, it is crucial that the routing and scheduling of payments are managed to optimize channel traffic and ensure maximum successful payment processing. Load balancing can occur at the channel, route, or node level.

- The load traversing specific channels can be allocated to prevent channel overload, which may adversely affect the network, particularly if the channel is a central one.

- Load between proximal sources and destinations can be allocated across various pathways, akin to the load balancing observed in Internet traffic.
- A node may prefer not to route an excessive number of payments in specific directions within its channels to prevent depletion and ensure availability of funds for higher-priority transactions.

Congestion control can similarly be examined, as the routing algorithm actively restricts the load placed on a channel, path, or node. Existing distributed routing protocols [61, 82] suffer from congestion and exhaustion of links, described in Chapter 3. We have proposed algorithms to improve the concurrent execution of transactions, avoiding congestion and exhaustion of links in Chapters 4 and 5.

Deadlocks

PCNs must contend with concurrency concerns, as the simultaneous scheduling of many transactions may lead to deadlocks [103]. Werman and Zohar [103] proposed a deadlock prevention algorithm for a centralized approach. After demonstrating the NP-completeness of deadlock-free routing for concurrent transactions, the author suggests an approach that locks every edge before initiating any transactions in order to prevent deadlock. However, such locking of edges can lead to starvation in two ways:

(a) If two transactions are in deadlock, they will try to lock edges but will be aborted as they will not lock all edges due to deadlock; they will try the same path and continue, thus leading to starvation.

(b) Malicious nodes may lock edges periodically to create congestion; the other fair transactions will starve due to this congestion.

Transaction identifiers were utilized by Malavolta et al. [62] to determine when to terminate transactions after links became saturated. If the transaction's identifier value is greater than that of another concurrent transaction, it is added to the queue and placed in a waiting state. As a result, the transaction must wait until the opposite flow of transactions refills the link. If the inflow and outflow of the transaction are not constant, the transaction will starve. Currently, there is no distributed PCN protocol that can effectively manage multiple concurrent transactions while considering deadlock constraints. Therefore, a distributed PCN protocol is required in order to boost the landmark routing mechanism's effectiveness while avoiding deadlocks. In Chapter 6, we have discussed this aspect in detail.

Channel Balance Saturation

Lightning channels are defined by their capacity over their lifespan. This may pose challenges in scenarios when one side of the channel exhibits significantly higher transaction volume than the other. This would result in a depletion of funds on one side and an increase of cash on the other. This would thus lead to the rejection of following transactions on the depleted side, as there would be insufficient cash to process them. Addressing this issue necessitates adjusting the channel. A significant metric to examine for any rebalancing approach is the routing capacity between the source node and the target destination node, namely the maximum total volume that can be transmitted to the destination across all available channels. Rebalancing may be accomplished by the following methods:

- **Off-chain** A node possessing several channels, which includes a drained channel, can identify a circular route in the PCN that leads back to itself, utilizing the drained channel as the final step, and subsequently transmit a payment to itself through this route [41, 74]. This would result in a reallocation of the node's total off-chain funds among its channels, hence enhancing liquidity on the depleted side. Certainly, if this cycle encompasses channels from additional nodes (which is highly probable), the rebalancing will incur some relay fees. Nodes must possess the capability to identify circular paths within their vicinity (for example, by the application of a pathfinding algorithm to the identical source and destination).

- **Adverse Fee** A node can equilibrate its channels by incentivizing other nodes to utilize them for transactions by establishing low, zero, or even negative costs [77]. Implementing negative fees implies that the node compensates others for routing payments through its channels, which may be justified if rebalancing certain channels in this manner facilitates greater profits (for example, by restoring routing capabilities on the previously depleted side of the channel).

In [99], buffer queues are recommended to lessen the requirement for online rebalancing to aid routing, although they cannot handle all sorts of flows. Additionally, buffer overflow might result in transaction loss when there is a lot of network traffic. In [57], a strategy is developed for controlling transaction flow to reduce query latency on the intermediate nodes. Deep Q-learning balances the cost of the forwarding charge and network throughput. In [16], an improved directed Dijkstra approach may find the optimum path. A PCN operator oversees the whole network and chooses the optimum path for users. The imbalance penalty and the time value cost are new additions to model fees which depend on the distance between the intermediate node and the destination node. These express the price intermediary nodes are prepared to pay to reach the best desirable condition for a payment channel.

Rebalancing methods required some amount of funds to be needed to maintain the balance on the channel. In Chapter 4 and 5, we tried to combine the concurrency with the self-rebalancing to improve the throughput.

Security

While HTLC and onion protocols guarantee the security and anonymity of multi-hop transactions, they also expose the system to various potential attacks, such as balance assaults

[37, 97], lockdown[73], griefing [56], congestion [65, 96], and wormhole [63]. These attacks have been analyzed within the framework of source routing, where each node updates its local network using a gossip protocol and stores all network data locally.

Three categories can be used to group PCN attacks. Attackers seeking to stop PCN transactions without profiting monetarily fall under the first category. Grieving transactions can be used to accomplish this, opening the door to numerous potential attacks. A balance lockout attack prevents a victim node from participating in payment routing; it is also known as a balance availability attack [73]. By blocking the victim node's balance funds, the attack stops it from taking part in further transactions. An attack of this kind reduces the system's efficiency and provides the attacker the upper hand. In particular, it gives attacker-favored routes a competitive advantage by enabling the attacker to block particular payment pathways. The attacker routes a payment through the target node by using their in-depth understanding of the network topology. Blocking victims' currency is the goal of a congestion attack ([56, 65]), which is less expensive than a balance lockdown attack. Targeting PCNs, the attack stops several payment channels at once and consistently. It uses a trustless payment system and a long HTLC expiration period. In this case, the source and destination of a payment are an attacker. It can configure multiple of these payments to increase network coverage. Subsequently, the attacker uses money sent down these channels to block nodes on each.

Finding the PCN balance is the second kind of attack. On the main blockchain, payment channel capacities are accessible, but the balance or intermediate values of the channels are not known. An attacker can select which nodes to attack to gain an advantage by being aware of these data. These attacks are referred to as "balance attacks" [37, 97]. A balance discovery attack aims to compromise user privacy by disclosing each user's individual balance. Any node that opens a channel with a malicious node is open to attack. Moreover, the attack incorporates all nodes that are just next to the victim node right away.

In the third type of attack, an attacker wants to steal the incentives of other nodes in the network. We have tested the impact of these attacks in distributed routing algorithms in Chapter 7.

2.6 Summary

We discussed the challenges for scalability in blockchain networks. The primary factors limiting the scalability of standard protocols are restricted block size and higher block creation time. In Layer-1 solutions, hard forks modify the current consensus mechanisms; nevertheless, they lack forward compatibility. Augmenting the block size and reducing the block generation time result in frequent forks and diminished security. Introducing an alternative consensus protocol requires essential modifications; new functionalities may introduce unforeseen risks or weaknesses, necessitating thorough risk evaluation and mitigation techniques.

Layer-2 solutions enhance scalability without altering the foundational consensus mechanism. We have examined other Layer-2 protocols, but PCNs emerged as the most viable method for enhancing scalability. Furthermore, we have examined the current routing protocols in PCN and their limitations. We elucidated several facets of routing (fee optimization, concurrency, deadlocks, channel balancing) with a focus on enhancement. In the subsequent chapters, we optimize the routing protocols concerning various aspects of routing as outlined in the preceding section.

3
Swift Routing

Among the several aspects of routing protocols discussed in Chapter 2, we first focus on fee reduction by optimizing the transaction route lengths to lower transaction costs. We propose *Swift* Algorithm; swift refers to being quicker in reference to existing protocols Slient [61] and Speedy [82]. Our proposed algorithm optimizes the routing by exploiting the concept of cloud. The cloud is used to optimize the inter-subtee paths, using the direct and indirect edges, as well as optimize resource utilization. Further, we conducted a theoretical analysis of our suggested method utilizing a k-cycle graph that is based on the cycle topology and makes use of junction and junction-cycle. Using *DRLNsim* Simulator (described in Chapter 4), we assess the efficacy of the suggested method in several network scenarios.

The rest of this chapter is organized as follows: Section 3.1 describes the primary design that is consistently referenced throughout this thesis. Sections 3.2 and 3.3 describe the proposed algorithm and its theoretical analysis, respectively. Section 3.4 presents simulations and evaluates the performance of the proposed algorithm. We finally conclude in Section 3.5.

3.1 Main Design

We model PCN as a directed graph $G(V, E, W)$, where V is the set nodes in the network, and E refers to directional links between the nodes. The weight function $W : E \to \mathbf{R}$. The initial values of $W(u, v)$ and $W(v, u)$ are investments made by u and v, respectively, while opening the channel between u and v. When a fund, say of amount ϵ, is transferred from u to v, the value of $W(u, v)$ decreases by ϵ. If $(u, v) \notin G(E)$ then $W(u, v) = 0$. At any point of time $W(u,v)$ must be non-negative. $W(u,v)$, also referred to as channel balance, $\mathbb{B}(u,v)$, refers to the maximum funds which u can transfer to v.

The channel capacity, $\mathbb{C}$, is also a function from $G(V) \times G(V) \rightarrow \mathbb{Z}$ and is defined as $\mathbb{C}(v,u) = \mathbb{C}(u,v) = \mathbb{B}(u,v) + \mathbb{B}(v,u)$. Note that while the value of $\mathbb{B}(u,v)$ changes with a transfer, the value $\mathbb{C}(u,v)$ remains constant throughout the lifetime of a channel.

Let $p_i = \langle u = u_1, u_2, \ldots, u_{k-1}, u_k = v \rangle$ be a path in G, where $(u_j, u_{j+1}) \in G(E)$ for $1 \leq j < k$. The maximum amount that can be transferred through p is $\min\{\mathbb{B}(u_i, u_{i+1}) \mid 1 \leq j < k\}$. A transaction T_i is a 3-tuple, (s, d, x), in which the source node s intends to transfer x units to the destination d.

We also have a set of landmark nodes known to other nodes in the network, denoted by LM. The nodes with higher degrees, also referred to as the most influential nodes in the network, make up the landmark node-set that is employed in distributed routing. Since the set of landmark nodes is dynamic, the system is expected to remain decentralized. This is because the new nodes take the place of many influential nodes that had previously lost the opportunity to be a landmark node [108].

(a) **setRoutes(LM, T)**: The set, T, is the number of transactions' requests received by a landmark node at an instant of time. A landmark node receives the sender and destination node's information in the transaction set; the transaction amount is not shared. The landmark node will return the path to the respective source of the transaction.

(b) **feasiblePath(c, u, v)**: A path is received from a landmark node to transfer funds, c, from node u to node v, such that all links in the path should have sufficient funds and value of c also include transaction fee collected by intermediate nodes in the path, thus

$$c \leq \min\{\mathbb{B}(u_i, u_{i+1}) \mid 1 \leq j < k\}$$

(c) **fee(T_i)**: The sum of fees charged by intermediate nodes $u_2, \ldots, u_{k-1}$ when transaction T_i routes a transfer to d from s is denoted by fee(T_i). The fees charged by intermediate nodes have two components:

  (i) base fee, which is fixed for each transaction, and

  (ii) proportional fee, which depends upon the amount to be transferred.

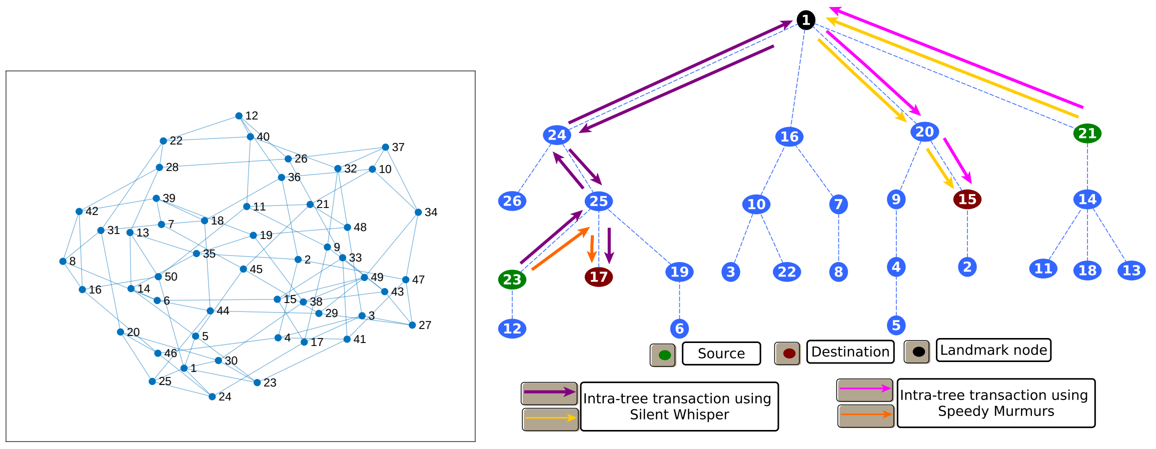
3.2 Swift Routing

Swift here refers to quick, as we intend to reduce the fee by reducing the path length, so our proposed algorithm will be quicker than the existing routing algorithms. To overcome the limitations of existing techniques and take advantage of the capability of nodes, a node maintains information about the local subnetwork but not the complete network. We define

Cloud Size as the number of hops up to which a node can store information about its neighbors. This is also referred to as the Ego network and K-Step Neighbourhood in the study of social networks [1][36, chap. 9]. Let $u, v \in V$. Let $\delta(u, v)$ be the minimum number of channels on a shortest path from u to v in the underlying graph.

Definition 3.2.1 (Cloud)**.** Let G be a graph, $u \in G(V)$ be a node and k be a positive integer referred to as *Cloud Size*. The cloud graph with respect to u and k is defined as G_u^k where $G_u^k(V) = \{z \in G(V) \mid \delta(u, z) \leq k\}$ and $G_u^k(E) = \{(x, y) \in G(E) \mid x, y \in G_u^k(V)\}$.

In the year 2018, the Lightning Network had around 2344 nodes [85], which has increased to 23000 in 2022 [49]. In future, it is expected to increase at a higher rate. Most of the distributed routing algorithms in the literature assume that a node will store just the information about the landmark node. This can lead to the underutilization of resources available to nodes. Further, the shortest path algorithms known for asynchronous fully distributed networks have exponential message complexity in the worst case [60]. Speedy murmurs reduce the path length of a transaction compared to silent whispers, but if both the sender and receiver belong to the same subtree. Suppose there are K landmark nodes. Each such node will be the root of a spanning tree. Consider the following two routing scenarios.


Source
Destination
Landmark node
Intra-tree transaction using Silent Whisper
Intra-tree transaction using Speedy Murmurs

(a) *A Lightning Network with 26 nodes* **(b)** *Routing tree with node 1 as the landmark node*

Figure 3.1: *Intra-tree Transactions*

Intra-tree Transaction Figures 3.1a and 3.1b shows a LN graph with 26 nodes and a routing tree with node 1 as the landmark node, respectively. Consider a transaction from node 23 to node 17; Silent Whisper will follow a longer path than Speedy Murmurs as shown in Figure 3.1b. But if the transaction is between two different subtrees, such as from node 21 to node 15, both the algorithms will follow the same path through the landmark node as shown in Figure 3.1b. Although these nodes are neighbors in the underlying LN graph (see Figure 3.1a), the routing will still be done through a longer path resulting in a higher transaction fee. This is a severe underutilization of resources.

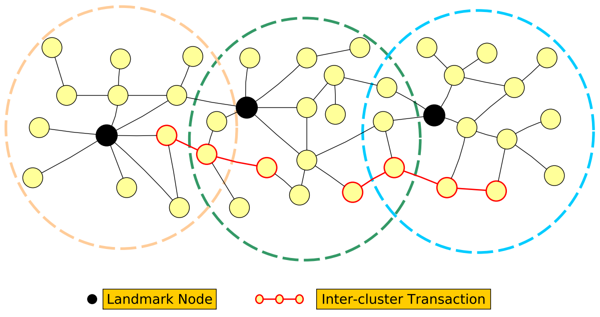


Figure 3.2: *Inter-tree transactions*

Inter-tree Transaction Suppose the sender and receiver lie in different spanning trees. In that case, the transaction is sent from the sender to its respective landmark node, and then the landmark node will forward the transaction to other landmark nodes of the spanning tree with the respective receiver.

In both intra- and inter-tree transactions, it will be helpful for a node to maintain information about its neighbours up to a few hops. It can be used to reduce the path length, hence the transaction fee. Some nodes in a network can be in more than one spanning tree to help in redirecting a transaction to the receiver without following the longer path through the landmark nodes, as shown in Figure 3.2. The spanning tree used for routing can be improved by exploiting the following paths:

(a) When a **direct path** is present in the tree, Swift routing can take such a path to reach a receiver quickly without travelling through tree edges, as shown in Figure 3.3.

(b) Swift routing can also route the transaction through **cross path** in the absence of a direct path. But the sender and receiver should be present within h hops, where h is the cloud size, as shown in Figure 3.3.

In a network graph, G, a set of nodes is designated as Landmark Nodes (LM). LM and non-LM node maintains a spanning tree and local graph, respectively. An LM constructs a spanning tree using BFS and VOUTE [81] coordinates. A landmark node assigns random VOUTE coordinates to the nodes in the spanning tree as shown in Algorithm 3.1. So each node $l \in LM$ has this spanning tree. LM nodes use this spanning tree to provide a path upon request from a sender. The routes from a sender to a LM node, say l, and from l to the ultimate recipient are joined to create a path. The same spanning tree is used for these routes by reversing the edges.

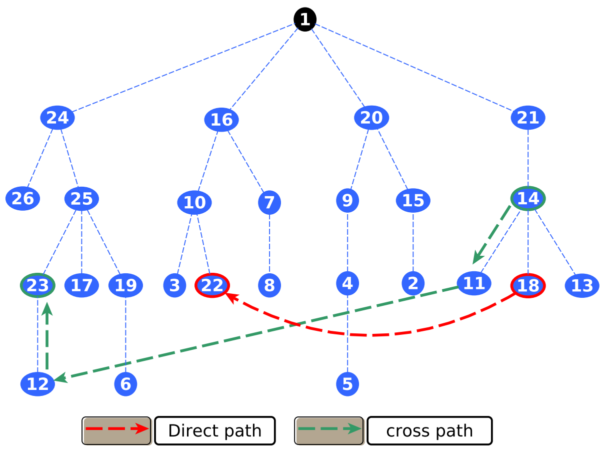


Figure 3.3: *Routing tree with* 1 *as the landmark node, cloud size* 4*. Routing from sender* 18 *to receiver* 22 *represents the direct path, and routing from sender* 14 *to receiver* 23 *represents the cross path. Red and green edges represent the direct and cross paths, respectively.*

Algorithm 3.1 Spanning tree construction

```
 1: procedure BuildSpanningTree($G$, $l$)            ▷ Executed by a landmark node $l \in LM$
 2:     $Q \leftarrow \emptyset$
 3:     $t_l \leftarrow \emptyset$                   ▷ $t_l$ is the spanning tree constructed at $l$
 4:     $Q.enqueue(l)$
 5:     while $Q \neq \emptyset$ do
 6:         $u \leftarrow Q.dequeue()$
 7:         for each $v \in \text{adj}(u)$ do        ▷ adj($u$) refers to neighbours of node $u$
 8:             if $v \notin t_l$ then
 9:                 add edge $(u, v)$ to tree $t_l$  ▷ Coordinate assignment
10:                 $id(v) = \text{CONCATENATE}(id(u), r)$   ▷ $r$ is a random $b$-bit number
11:                 $Q.enqueue(v)$
        return $t_l$
```

Each node in the network independently maintains its *cloud* whose size depends on the cloud size. The local subgraph is obtained using Algorithm 3.2 which has two procedures that run in parallel. The procedure Init at node u sends the InitMessage to its neighboring nodes. The procedure UpdateGraph iteratively updates the local graph g_u on receiving the messages from other nodes. The received message will update the local graph g_u and be forwarded by concatenating their edge information until cloud size becomes zero.

Transaction Routing: Consider a transaction from a sender (say s) to a receiver (say t) of value x. First, t sends its hashed coordinates to s. Then, s will initiate the transaction by calling the RouteTransaction(s, t, x), see Algorithm 3.3. This will first compare

Algorithm 3.2 Local graph construction

```
1: $g_u \leftarrow \emptyset$                                  ▷ Local graph $g_u$ maintained at node $u$
2: procedure INIT($u$)                                         ▷ $u \in G(V) - LM$
3:     $c \leftarrow$ cloud_size                               ▷ size of the cloud
4:     for each $v \in adj(u)$ do                              ▷ Process initiation at u
5:         INITMESSAGE($id_u$, $id_v$, $c$, $[u, v]$)

1: procedure UPDATEGRAPH( )
2:     while TRUE do
3:         RECEIVEMESSAGE($id_x$, $id_u$, $c$, $path$)         ▷ path: nodes followed by message
4:         if $path \notin g_u$ then
5:             add $path$ to $g_u$
6:         if $c > 0$ then      ▷ Message is forwarded until the current cloud size $c$ reaches 0
7:             for each $v \in adj(u) - \{x\}$) do
8:                 FORWARDMESSAGE($id_u$, $id_v$, $c - 1$, path.append(v))
```

the coordinates of s and t if they lie in the same subtree, then the path is finalized as it will be the shortest. But if there is no coordinate match, the receiver is located elsewhere then the local subgraph g_u is exploited to find a path between s and t. If the latter is not found in g_u, the path through the landmark node l and intercloud are compared to get the appropriate path finally. When multiple paths satisfy the minimum path requirement, the sender s chooses a random path among them. Once the path is selected, node s will initiate the onion routing [40] by sending the onion message, including the path and funds to be transferred. Upon receiving the onion message, each node will peel it to find the next destination point of the message, build the *HTLC* with the next node, and forward the message. This process repeats until t receives the transaction.

3.3 Theoretical Analysis of Swift Routing

We now theoretically analyze the total optimization that can achieved through the Swift routing algorithm. The path formed by selecting the direct and cross edges is called the secret path. Swift routing posts improvement in transaction fees by choosing shorter paths when it picks secret paths. Secret paths also improve the transaction fee of inter-cluster transactions by redirecting them through the intersection nodes using direct or cross paths. The possibility of a direct path will be less, so most of the optimization is consummated by cross edges. It can be observed in Figure 3.3 that the cross path, $(14 - 11 - 12 - 23)$, and the actual path, $(14 - 21 - 1 - 24 - 25 - 23)$, together form a cycle. This means cross paths will reduce the path length by redirecting transactions using the cycle topology.

Let G be a LN graph, and l be a landmark node. Further, $T^{'}$ be the coordinate tree

Algorithm 3.3 Transaction routing

```
1: procedure RouteTransaction(sender, receiver, amount)
2:     if receiver.coordinate ∈ subtree then    ▷ subtree belongs to spanning tree at $l \in LM$
3:         path ← ComputePath(sender.coordidnate, receiver.coordinate)
4:         SendOnion(path, amount)
5:     else if  receiver.coordinate ∈ $g_s$ then
6:         path ← ShortestPath($g_s$)
7:         SendOnion(path, amount)
8:     else
9:         lpath ← LMPath(l, sender, receiver)    ▷ l will return path using spanning tree
10:        dpath ← IntercloudPath(sender, receiver)    ▷ path via cloud
11:        path ← lpath if length(lpath) < length(dpath) else dpath
12:        SendOnion(path, amount)

1: procedure ReceiveOnion(path, amount)
2:     OnionPeel()
3:     next ∈ Neighbours(i)    ▷ next node in the path
4:     Build HTLC with the next node in path with amount
5:     SendOnion(path −{i}, amount)
```

built by l using the spanning tree algorithm.

Definition 3.3.1 (Junction Nodes)**.** A pair u, v of nodes in $T^{'}$ are called junction nodes when $(u, v) \in G(E)$ and the least common ancestor of u and v in $T^{'}(V)$ is neither u nor v.

Definition 3.3.2 (Junction Cycle)**.** Let u, v be junction nodes in $T^{'}$ with p as the least common ancestor. The paths from p to u, u to v and v to p will form a cycle. This cycle is called as junction cycle of the junction nodes u and v.

Theorem 3.3.1. *Let $T^{'}$ be a coordinate tree with no direct edges. Then, $T^{'}$ can be divided into a combination of junction cycles and subtrees attached to the junction cycles.*

Proof. The tree $T^{'}$ will have two types of edges, viz. tree edges and cross edges. Tree edges will not result in a junction as a parent is always the least common ancestor. But cross edges will always result in a cycle and a junction. Since direct edges are not present, cycles are only caused by cross edges. For every cross edge, there exists a corresponding junction cycle. A set of nodes will be a part of these junction cycles. Nodes not present in any of the junction cycles will be a part of a subtree attached to the junction cycle as $T^{'}$ is a connected graph. □

Now the transactions in graph G can be classified into following categories:

(a) Transactions within a junction cycle,

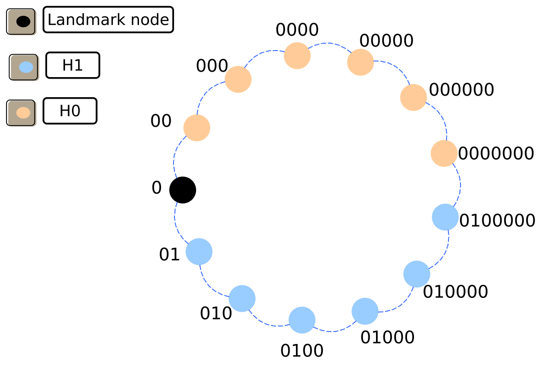
(b) Transactions between junction cycle,

(c) Transactions within a subtree,

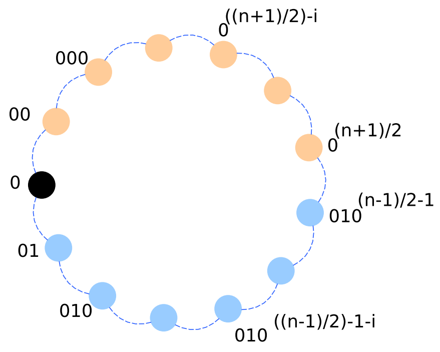
(d) Transactions between a subtree and a junction cycle, and

(e) Transactions between subtrees.

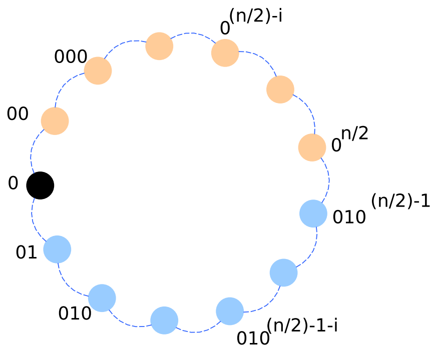
Transactions within a subtree will not observe any improvement in Swift routing because there is only a single simple path between any two nodes. Transactions between a subtree and a junction cycle or transactions between subtrees will always pass through the nodes present in junction cycles. No optimization is observed in the sub-paths within a subtree. So for studying optimization, sub-paths in junction cycles are sufficient. Hence, transactions between a subtree and a junction cycle or between subtrees can be seen as transactions between or within a junction cycle. Next, we will look at the analysis of transactions within a junction cycle.



(a) *Graph with $n = 13$*



(b) *Graph with odd n*



(c) *Graph with even n*

Figure 3.4: *Cycle graphs*

3.3.1 Cycle Topology

Consider a cycle graph of size n. A node is selected randomly as a landmark and assigned 0 as a coordinate. Nodes on the left side of the landmark node are assigned with coordinate 0^j, $2 \leq j \leq \frac{(n+1)}{2}$; and nodes right to landmark node are assigned with coordinate 010^j, $0 \leq j \leq \frac{(n-1)}{2} - 1$ as shown in Figure 3.4. Let H_0 and H_1 be the sets of nodes with coordinate 0^j and 010^j, respectively. Let each node store a local neighborhood of size $2c$, with c nodes on each side. The edge connecting H_0 and H_1 is called the junction. The following types of transactions can take place.

(a) Between two nodes in H_0 or two nodes in H_1, or between nodes of H_0 or H_1 and 0. In these cases, Swift routing will perform routing through the same route as the basic routing algorithm.

(b) Between nodes of H_0 and H_1. Swift routing will find shortcuts in some cases. For the transactions where the sender is at most $c-1$ hops away from the junction of H_0 and H_1, Swift routing will optimize the transaction.

Definition 3.3.3 (Distance ($\mathbb{D}$))**.** Let G be the cycle graph and u and v be nodes in G. The distance, $\mathcal{D}(u, v)$, between the nodes u and v, in G is defined as the number of nodes on the path from u to v. It can be computed using the node labelling with respect to a landmark node (0) for $y, z \geq 1$ as follows:

$$\begin{aligned}
\mathbb{D}(0^z, 0^y) &= |z - y| && \text{where, } u, v \in H_0 \text{ or } u, v \in H_1 \\
\mathbb{D}(0^z, 010^y) &= |z + y| && \text{where, } u \in H_i,\ v \in H_j,\ i, j \in \{0, 1\} \text{ and } i \neq j
\end{aligned}$$

Next, we analyze cases when n is odd.

1. **(Sender and Receiver are nodes of $\mathbf{H_1}$ and $\mathbf{H_0}$, respectively)** Let the sender be at i hops from the node $010^{\frac{(n-1)}{2}-1}$. So the coordinates of the sender are $010^{\frac{(n-1)}{2}-i-1}$. Now, the two farthest points in the local graph of the sender are $010^{\frac{(n-1)}{2}-i-c-1}$ in H_1 and $0^{\frac{(n+1)}{2}+i+1-c}$ in H_0. Let 0^x be the receiver.

   The goal is to find the number of values of x for which the sender will route optimally. For this, the distance between $0^{\frac{(n+1)}{2}+i+1-c}$ and 0^x must be lower than the distance between $010^{\frac{(n-1)}{2}-i-c-1}$ and 0^x.

$$\begin{aligned}
\frac{(n+1)}{2} + i + 1 - c - x &< x + \frac{(n-1)}{2} - i - c - 1 \\
\Rightarrow \qquad x &> i + \frac{3}{2}
\end{aligned}$$

Therefore, all the transactions with receiver 0^{i+2} to $0^{\frac{(n+1)}{2}}$ and sender $010^{\frac{(n-1)}{2}-i-1}$ can be routed optimally. The sum over all i will be $\displaystyle\sum_{i=1}^{c-1}\left(\frac{(n+1)}{2}-i-1\right)$. To route a transaction from the sender $010^{\frac{(n-1)}{2}-i-1}$ to the receiver 0^x, Swift routing will take $(n-\mathbb{D}(010^{\frac{(n-1)}{2}-i-1}, 0^x)) = \frac{(n+1)}{2}+i+1-x$ hops. For the same transaction, existing routing algorithms will take $\mathbb{D}(010^{\frac{(n-1)}{2}-i-1}, 0^x) = \frac{(n-1)}{2}-i-1+x$ hops. Hence the optimization observed is $2x-2i-3$.

The optimization observed over all the senders is $\displaystyle\sum_{i=1}^{c-1}\sum_{x=i+2}^{\frac{(n+1)}{2}}(2x-2i-3)$.

2. **(Sender and Receiver are nodes of $\mathbf{H_0}$ and $\mathbf{H_1}$, respectively)** Let the sender be at i hops from the node $0^{\frac{(n+1)}{2}}$. So the coordinate of the sender is $0^{\frac{(n+1)}{2}-i}$. Now, the two corner points in the local graph of the sender are $010^{\frac{(n-1)}{2}+i-c}$ in H_1 and $0^{\frac{(n+1)}{2}-c-i}$ in H_0. Let 010^x be the receiver.

The goal is to find the number of values of x for which the sender will route optimally. For this, the distance between $010^{\frac{(n-1)}{2}+i-c}$ and 010^x should be lower than the distance between $0^{\frac{(n+1)}{2}-c-i}$ and 010^x.

$$\begin{aligned}\frac{(n-1)}{2}+i-c-x &< \frac{(n+1)}{2}-c-i+x\\ \Rightarrow \qquad x &> i-\frac{1}{2}\end{aligned}$$

Therefore all the transactions with receiver 010^i to $010^{\frac{(n-1)}{2}-1}$ and sender $0^{\frac{(n+1)}{2}-i}$ are routed using secret paths. The sum overall i will be $\sum_{i=1}^{c-1}\frac{(n-1)}{2}-i$.

To route a transaction from sender $0^{\frac{(n+1)}{2}-i}$ to the receiver 010^x, Swift routing will take $(n-\mathbb{D}(0^{\frac{(n+1)}{2}-i}, 010^x)) = \frac{(n-1)}{2}+i-x$ hops. For the same transaction, existing routing algorithms will take $\mathbb{D}(0^{\frac{(n+1)}{2}-i}, 010^x) = \frac{(n+1)}{2}-i+x$ hops. Hence the optimization observed is $2x-2i+1$.

The optimization observed over all the senders is $\displaystyle\sum_{i=1}^{c-1}\sum_{x=i}^{\frac{(n-1)}{2}-1}(2x-2i+1)$.

Finally, the total number of transactions with optimization is

$$\sum_{i=1}^{c-1}\left(\frac{(n+1)}{2}-i-1\right)+\sum_{i=1}^{c-1}\left(\frac{(n-1)}{2}-i\right)=n(c-1)-c^2+c-1$$

Therefore, the total optimization is

$$\sum_{i=1}^{c-1}\sum_{x=i}^{\frac{(n-1)}{2}-1}(2x-2i+1)+\sum_{i=1}^{c-1}\sum_{x=i+2}^{\frac{(n+1)}{2}}(2x-2i-3)=\frac{(n^2(3c-3)-n(6c^2-6)+4c^3-c-3)}{6}.$$

Next, we analyze cases when **n is even**.

3. **(Sender and Receiver are nodes of $\mathbf{H_1}$ and $\mathbf{H_0}$, respectively)** Let the sender be at i hops from the node $010^{\frac{n}{2}-1}$. So the coordinates of the sender are $010^{\frac{n}{2}-i-1}$. Now, the two farthest points in the local graph of the sender are $010^{\frac{n}{2}-i-c-1}$ in H_1 and $0^{\frac{n}{2}+i+1-c}$ in H_0. Let 0^x be the receiver.

   The distance between $0^{\frac{n}{2}+i+1-c}$ and 0^x must be lower than the distance between $010^{\frac{n}{2}-i-1-c}$ and 0^x.

$$\begin{aligned} &\frac{n}{2}+i+1-c-x<\frac{n}{2}-i-1-c+x \\ \Rightarrow \qquad & x>i+1 \end{aligned}$$

   Therefore all the transactions with receiver 0^{i+2} to $0^{\frac{n}{2}}$ and sender $010^{\frac{n}{2}-i-1}$ can be routed optimally. The sum overall i will be $\sum_{i=1}^{c-1}\left(\frac{n}{2}-i-1\right)$. To route a transaction from sender $010^{\frac{n}{2}-i-1}$ to the receiver 0^x, Swift routing will take $(n-\mathbb{D}(010^{\frac{n}{2}-i-1},0^x))=\frac{n}{2}+i+1-x$ hops. Hence the optimization observed is $2x-2i-3$.

   The optimization observed over all the senders is $\sum_{i=1}^{c-1}\sum_{x=i+2}^{\frac{n}{2}}(2x-2i-2)$.

4. **(Sender and Receiver are nodes of $\mathbf{H_0}$ and $\mathbf{H_1}$, respectively)** Let the sender be at i hops from the node $0^{\frac{n}{2}}$. So the coordinate of the sender is $0^{\frac{n}{2}-i}$. Now, the two corner points in the local graph of the sender are $010^{\frac{n}{2}+i-c}$ in H_1 and $0^{\frac{n}{2}-c-i}$ in H_0. Let 010^x be the receiver.

The distance between $010^{\frac{n}{2}+i-c}$ and 010^x should be lower than the distance between $0^{\frac{n}{2}-c-i}$ and 010^x.

$$\begin{aligned} & \frac{n}{2}+i-c-x < \frac{n}{2}-c-i+x \\ \Rightarrow \quad & x > i \end{aligned}$$

Therefore all the transactions with receiver 010^{i+1} to $010^{\frac{n}{2}-1}$ and sender $0^{\frac{n}{2}-i}$ are routed using secret paths. The sum over all i is $\sum\limits_{i=1}^{c-1} \frac{n}{2} - i - 1$.

To route a transaction from sender $0^{\frac{n}{2}-i}$ to the receiver 010^x, Swift routing will take $(n - \mathbb{D}(0^{\frac{n}{2}-i}, 010^x)) = \frac{n}{2} + i - x$ hops. For the same transaction, existing routing algorithms will take $\frac{n}{2} + x - i$ hops. Hence the optimization observed is $2x - 2i$.

The optimization observed over all the senders is $\sum\limits_{i=1}^{c-1} \sum\limits_{x=i+1}^{\frac{n}{2}-1} (2x - 2i)$.

Finally, the total number of transactions with optimization is

$$\sum_{i=1}^{c-1} \left(\frac{n}{2} - i - 1 \right) + \sum_{i=1}^{c-1} \left(\frac{n}{2} - i - 1 \right) = n(c-1) - (c-1)(c+2)$$

Therefore, the total optimization is

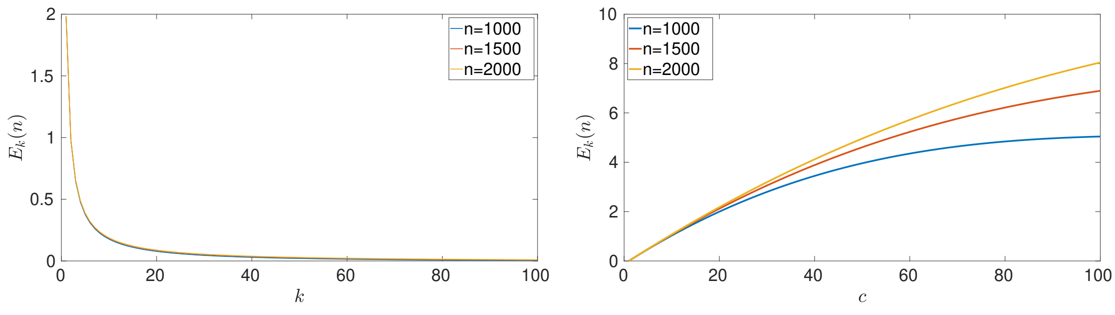
$$\sum_{i=1}^{c-1} \sum_{x=i+1}^{\frac{n}{2}-1} (2x - 2i) + \sum_{i=1}^{c-1} \sum_{x=i+2}^{\frac{n}{2}} (2x - 2i - 2) = \frac{(n^2(3c-3) - n(6c^2-6) + 4c^3 - 4c)}{6}.$$

3.3.2 K-cycle Graph

Only the nodes in the c neighborhood of junctions reap the benefits of the Swift routing algorithm in intra-cycle transactions. Let the discrete random variable X_k denote the improvement in the transaction from a pair of nodes in a network of n nodes with k equal length cycles (k junctions). The expected value of the random variable X_k is calculated using $E(X_k) = \sum x_i f(x_i)$, where $f(x_i)$ is the PMF (Probability Mass Function) of the random variable X_k. Assume that the PMF is the uniform distribution, meaning that every transaction is equiprobable. Using $f(x_i) = \frac{1}{n^2}$ and using the formula for improvement from the above cycle analysis, we get

$$E_k(n) = \frac{k\left(\frac{n^2}{k^2}(3c - 3) - \frac{n}{k}\left(6c^2 - 6\right) + 4c^3 - c - 3\right)}{6n^2}$$

The expected improvement per transaction decreases as junctions increase, as shown in Figure 3.5a. This is because the length (the number of hops) of transactions in which



(a) $E_k(n)$ *vs* k, $c = 4$ **(b)** $E_k(n)$ *vs* c, $k = 4$

Figure 3.5: *Theoretical analysis of expected improvement with change in network parameters*

improvement is observed is reducing as cycle size decreases. But on the contrary, the number of nodes getting improvement is increasing due to an increase in the number of junctions. So, there is a tradeoff between the expected improvement per transaction and the number of senders enjoying improvement. The above formulae are valid under the assumption that a node shouldn't have knowledge of the entire cycle, that is, $2(c-1) < \dfrac{n}{k} \Rightarrow c < \dfrac{n}{2k} + 1$. The expression $E_k(n)$ is cubic in c and has local extremas, where

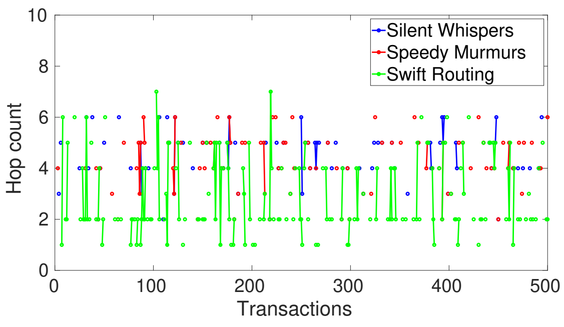
$$\frac{\partial E_k(n)}{\partial c} = \frac{k\left(3\left(\frac{n}{k} - 2c\right)^2 - 1\right)}{6n^2}$$
$$\frac{\partial E_k(n)}{\partial c} = 0, \text{ and } c = \frac{\frac{n}{k} \mp \frac{1}{\sqrt{3}}}{2}$$

With the increase in the neighborhood hop count, the expected improvement per transaction increases in a parabolic fashion until the entire cycle is in the neighborhood, as shown in Figure 3.5b.

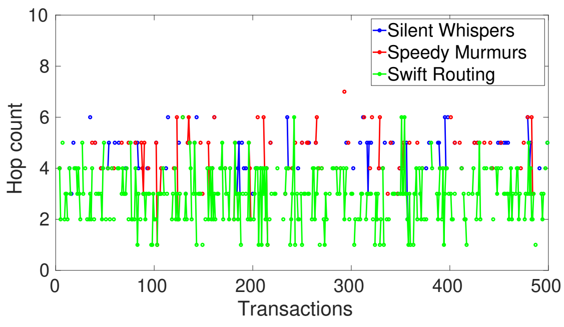
3.4 Simulations

Simulation Setup

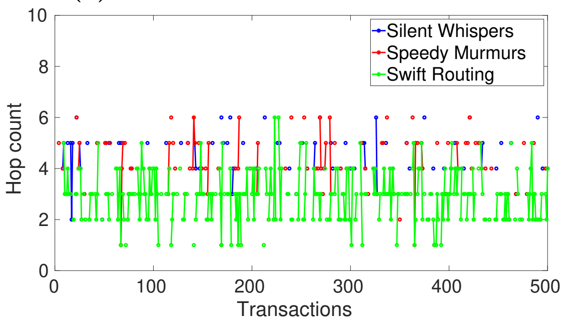
We have run different simulations to find the hop count by varying the input variables of the simulator. Initially, we fix the channel capacities, transaction range, average degree, and iterations; and vary the number of nodes, LM degree, and transaction count. Nodes in the Lightning Network follow the power law, which means nodes try to join the nodes with higher degrees [64], so there is a centralized set of nodes in the network, which can be referred to as landmark nodes. However, with time the influential nodes change, so there is no issue of centralization. So we have considered the LM degree between 4% to 10% of the number of nodes in the network [17]. The number of nodes is varied from 1000-5000. Every



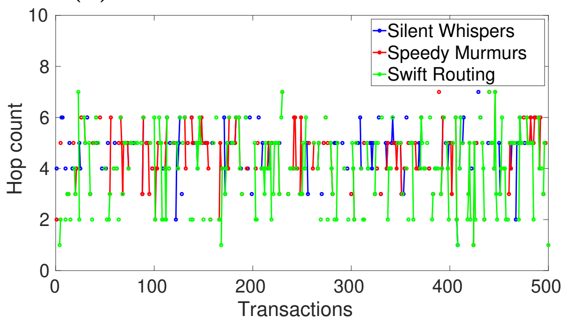
(a) *For 1000 nodes with cloud size 3*



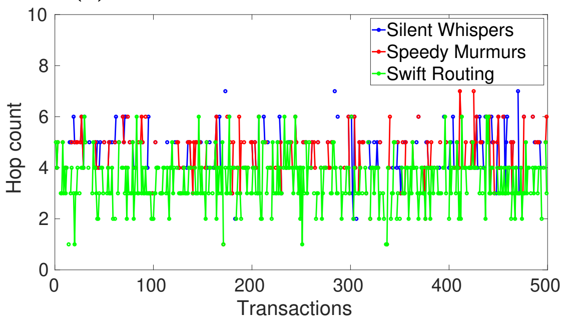
(b) *For 1000 nodes with cloud size 5*



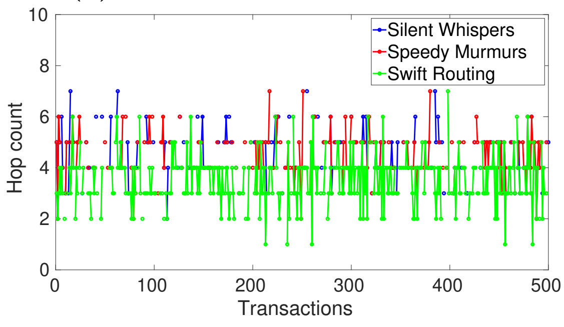
(c) *For 1000 nodes with cloud size 7*



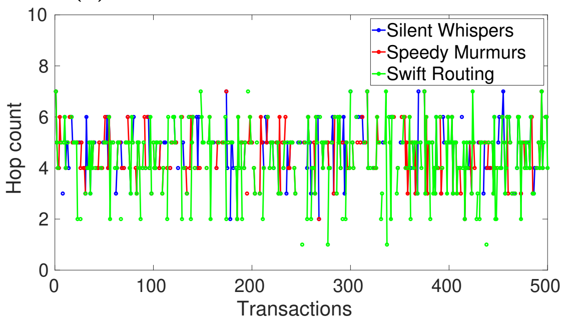
(d) *For 3000 nodes with cloud size 3*



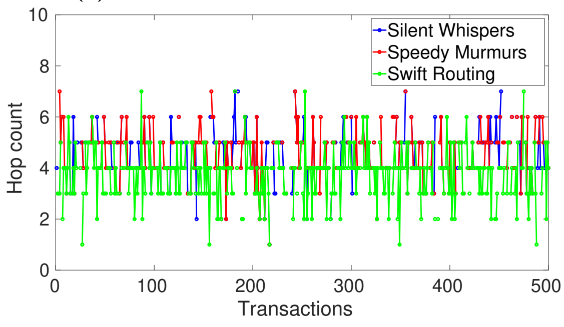
(e) *For 3000 nodes with cloud size 5*



(f) *For 3000 nodes with cloud size 7*

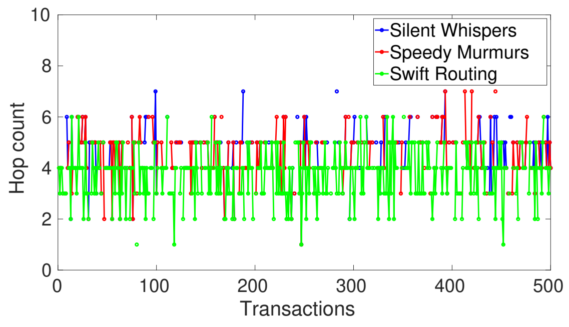


(g) *For 5000 nodes with cloud size 3*



(h) *For 5000 nodes with cloud size 5*

Figure 3.6: *Hop count versus transactions over δt for different number of nodes and cloud sizes*



(i) *For 5000 nodes with cloud size 7*

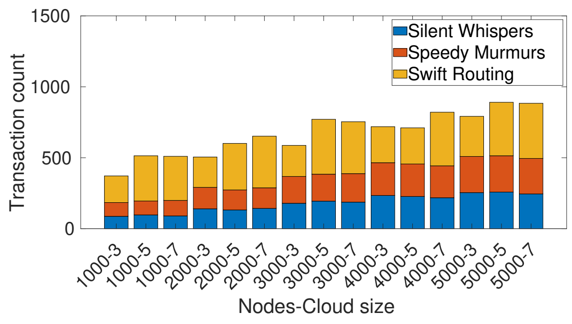
Figure 3.6: *Hop count versus transactions over δt for different number of nodes and cloud sizes*

channel has a capacity that is arbitrarily determined between 500 and 1000. A random pair is chosen as the sender and recipient for each transaction. Every transaction has a fund value between 100 and 500 which is selected randomly. The range of cloud size is 5 to 7. The average degree of non-LM nodes is kept as 7 [85].

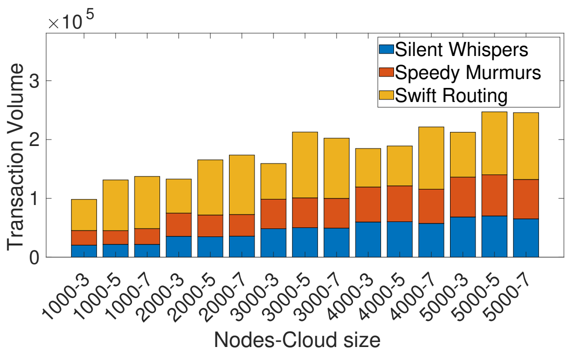
Then we vary the LM degree and average channel degree of the network by fixing the other input metrics. The average degree, transaction number, transaction range and channel capacity for this 2000 node simulation are 7, 500, 100-500, and 500-1000, respectively; the LM degree is adjusted from 80 to 200. The average degree is varied from 5-10 and the LM degree is set to 80, the rest of the parameters are the same. As the Lightning Network is growing with time, the number of nodes in the Lightning Network is doubling rapidly. New nodes in the network tend to join the node with a higher degree [64]. As the LM degree will increase with time, it is important to find how our proposed algorithm will respond to an increase in the LM degree and channel average degree. We also computed the transaction count and volume by varying the network and cloud sizes. As the Swift routing algorithm picks a secret path between inter-subtree transactions, it will be interesting to find out whether Swift improves throughput while optimizing hop count.

Results

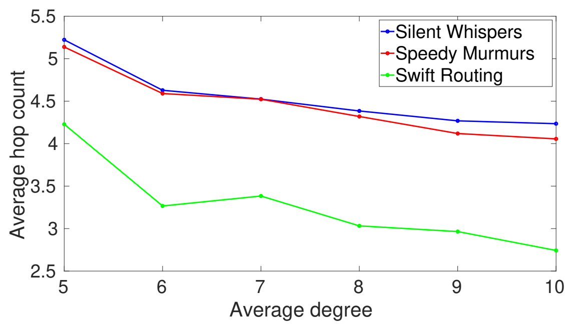
Case 1 - Exploitation of Secret Path in Swift Routing: The plots in Figure 3.6 show the variation of hop counts with the number of transactions. The hop count refers to the number of nodes a transaction requires to reach the destination. The hop count of transactions through Swift routing is less than the state-of-the-art. When cloud size is 3, the possibility of secret paths is reduced. Thus improvement is observed in a few cases, so we have more overlapping lines in plots 3.6a, 3.6d and 3.6g, which represents that in most of the cases, the transactions are routed through landmark nodes. But when cloud size is set to 7, almost all transactions have a possible secret path to route the transaction. Thus there are very few cases where Swift routing overlaps with the other algorithms. Comparatively,



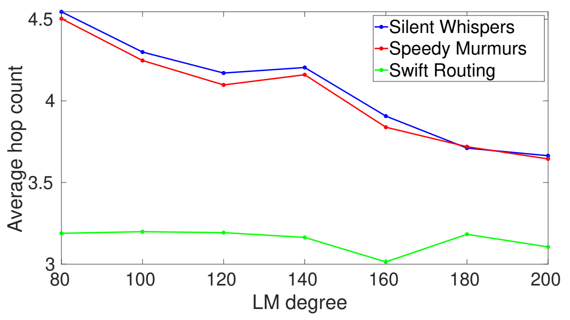
(a) *Transaction count over the change in network and cloud size*



(b) *Transaction volume over the change in network and cloud size*



(c) *Hop count over the change in average degree*



(d) *Average hop count over the change in LM degree*

Figure 3.7: *Parametric evaluation of Swift algorithm*

speedy murmurs have less optimization as their maximum cases overlap with the silent whisper. When cloud size is 5, we have intermediate values.

Case 2 - Throughput of Routing Algorithms: Throughput here refers to the successful transactions or transaction count when the same set of transactions are served to the routing algorithms. We conducted numerous simulations by varying the network and cloud sizes, as shown in Figure 3.7a. Transaction count increases with cloud size, as the sender will have multiple paths to route a transaction. This will also lead to load sharing among the edges, and chances of congestion will also be reduced. Figure 3.7b shows that the transaction count and volume are the same in these instances. This means transaction volumes are proportional to its transaction count, which validates the improvement of our proposed Swift routing algorithm over the existing ones.

Case 3 - Change in Other Network Parameters: Figure 3.7c and 3.7d shows the change in the hop count with an increase in the landmark node degree and average degree of nodes. In general, if we increase the average degree of nodes or landmark degree in the routing tree, the flow of transactions will increase due to an increase in the possibility of paths to route the transaction. As more possibilities are there to route the transaction, hence more possibilities of secret paths for Swift routing. Figure 3.7c, 3.7d shows that as

$G(V, E)$	cloud size	**Algorithm**	**Transaction count**	**Transaction Volume**	**Average path length**
(1000,3515)	3,5,7	**Swift-Silent**	80.00%	90.03%	−36.44%
		Swift-Speedy	73.82%	87.93%	−33.67%
(2000,7034)	3,5,7	**Swift-Silent**	75.24%	86.78%	−27.52%
		Swift-Speedy	67.15%	76.18%	−25.71%
(3000,10556)	3,5,7	**Swift-Silent**	57.28%	68.00%	−20.01%
		Swift-Speedy	54.53%	63.93%	−18.36%
(4000,14074)	3,5,7	**Swift-Silent**	25.63%	29.06%	−12.04%
		Swift-Speedy	24.92%	27.58%	−10.98%
(5000,17596)	3,5,7	**Swift-Silent**	38.57%	45.81%	−14.37
		Swift-Speedy	37.84%	44.98%	−13.93
Overall Performance of Swift			**53.56%**	**63.23%**	**−21.30%**

Table 3.1: *Performance of Swift routing algorithm with state-of-the-art algorithms*

we increase the landmark and average degree, the hop count will decrease as the network becomes denser and the reachability of nodes in the network improves; thus, the hop count will decrease.

Simulations and Theoretical Analysis As noted in Section 3.3 while analyzing the proposed algorithm with the increase in the neighborhood hop count or the cloud size, the improvement per transaction increases in a parabolic fashion. A similar phenomenon is observed in Case 1 when the cloud size is 3 where very few transactions show improvement. When it increases to 5, the difference is noted in graphs 3.6b, 3.6e and 3.6h, our proposed algorithm can find secret paths for more than half of the transactions. Finally, when cloud size is set to 7, almost all transactions have less hop count. This simulation result supports our theoretical result, as shown in Figure 3.5b.

We have observed that our proposed algorithm's hop count is significantly less than existing algorithms in all cases, as our proposed algorithm aims to optimize the transaction fee, which depends on the path length. The performance of our proposed algorithm is given in Table 3.1, showing that they performed well compared to the Silent Whisper [61] and Speedy Murmurs [82] for distributed routing. Secret paths help reduce the path length and improve the throughput, as observed in Case 2. This is indicated by negative values in the average path length column in Table 3.1. Secret paths utilize the links not used by the existing routing protocols, which distributes the load and improves channel utilization. As the cloud size increases, our proposed algorithm reduces the hop count, reducing the transaction fee. With the change in network parameters, the performance of our algorithm remains unaltered, thus it is scalable with network size.

3.5 Conclusions

We have proposed an algorithm, Swift Routing, to minimize the transaction fee in PCNs. Secret paths used in our algorithm not only reduce the transaction fee but also distribute the load over multiple edges resulting in higher throughput. Our algorithm considers the dynamic nature of the channel capacities in the cloud and optimally chooses the path. We theoretically analyzed our proposed algorithm using cycle topology to discover the expected improvement. The theoretical analysis shows that the expected improvement per transaction follows the parabolic pattern which coincides with our experimental evaluation. Our experimental results have shown that the proposed algorithm reduces fees by up to 21% and increases throughput by up to 63% compared to the current state of the art. In the subsequent chapter, we will demonstrate methods for transaction concurrency and channel rebalancing.

4

Concurrency-enhancing Algorithms *maxECW* and *maxSCL*

We now focus on the transaction concurrency and channel balancing aspects related to routing. As discussed in Chapter 2, the existing routing algorithms do not hold well with concurrent transactions and can lead to congestion and exhaustion of links. Congestion is caused by existing routing protocols like Silent Whisper and Speedy Murmurs, disregarding concurrent transaction execution by allocating the same path based on BFS. With a high rate of concurrent transactions, many transactions are expected to get aborted. To accommodate the execution of concurrent transactions at a high rate, we need a robust routing algorithm to deal with the issues faced by existing protocols. We propose new distributed routing algorithms that maximize the execution of concurrent transactions. The idea of self-rebalancing is presented, where channels are infinitely long-lived and never exhausted. Two algorithms are proposed, *maxECW* and *maxSCL*, which use the channel's weights and degrees to improve the transaction feasibility. The simulation and performance testing of these algorithms is done on *DRLNsim* simulator.

The rest of this Chapter is organized as follows: The self-rebalancing factor and importance of channel weights are explained in Section 4.1, Section 4.2 decribes our proposed algorithms; Section 4.3 represents simulator flow and setup; and evaluate the performance of proposed algorithms, and we finally conclude in Section 4.4.

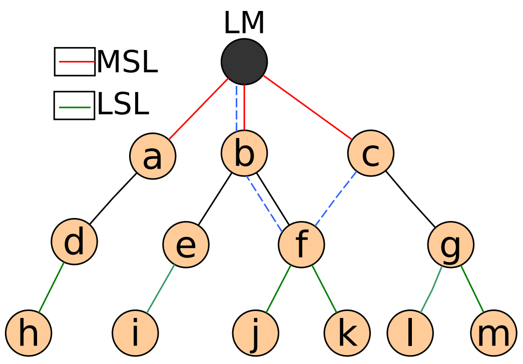
4.1 Motivation

Landmark routing finds the shortest path using BFS to minimize the path length. Selecting the shortest path is optimal for minimizing transaction fees, but it reduces concurrency. Consider the example shown in Figure 4.1a representing a BFS tree rooted at a landmark root. There are two types of links in the landmark routing tree. Most Significant Links (MSL) are links directly connected to landmark nodes, and the maximum number of transactions will route through these links. Least Significant Links (LSL) are links that are connected between the intermediate node and leaf node, as only one node will send or receive transactions through these links. The other links are between MSL and LSL. As BFS does not consider channel capacity, this will affect the feasibility of transactions. We will consider two cases where it hinders performance.

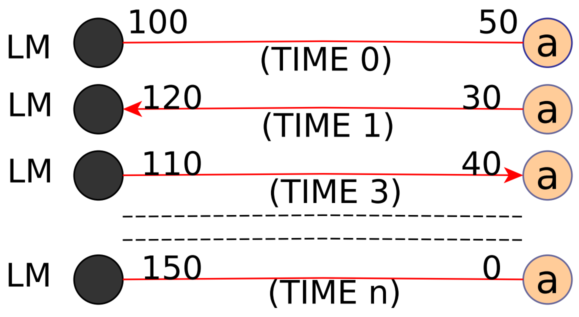
Congestion due to poor MSL If MSL links have low channel capacity, it will reduce subsequent transaction flow. For example, consider an MSL between node c and LM. If its channel capacity is very low, it will hinder the flow of transactions. Less channel capacity means the weight on that particular link is less than the average transaction size. So this channel will limit the transactions through it. The average lifetime of the channel is 30522 blocks or 212 days [107], so every time, the same route is followed by nodes in a particular sub-tree. However, this is the shortest path but not the desired path.

All MSLs with poor channel capacity will lead to congestion and deteriorate the throughput of the Lightning Network. This can be improved by considering channel capacity while forming a routing tree. In the same scenario, consider there exists another path with better channel capacity c–f–b–LM (see Figure 4.1a). This will increase the path length but improve the throughput. In the case of concurrent transactions, there will be a drastic difference in throughput. If we consider channel capacity in forming routing trees, LSL will automatically be set for leaf nodes. Thus transactions will not be dropped due to insufficient balance on channels.

Exhaustion of MSLs After a few iterations, MSL's capacity will get exhausted as maximum transactions flow through these links. Such an example is shown in Figure 4.1b. Initially, the channel is created between link (LM–a) at *Time 0*. When a set of concurrent transactions use this link, the net effect of channel capacity is shown in *Time 1*. Similarly, it is shown in *Time 3*. However, if the inflow to LM is greater than the outflow after *Time n*, this link will get exhausted. After *Time n*, no more transactions from the sub-tree attached to the node a can be transferred to other nodes in the network. Transactions have to wait until this link is refilled back. Many transactions will be dropped during this time due to insufficient balance on the channel. Generally, a landmark node has a very high degree, so many channels, such as (LM–a), may exist as shown in Figure 4.1b. The higher the number of links in this category, the greater the probability of transactions dropping. This effect will have more impact on the channels neighboring the landmark node; its effect will decrease as we move down toward the leaf node in the landmark routing tree. It implies that it is not good to have the same routing tree on each iteration; some changes are required depending



(a) *Landmark routing tree*



(b) *Link capacity at different timestamps*

Figure 4.1: *Congestion in landmark routing*

on the channel capacities.

The solution for the above issues is self rebalancing which means avoiding channels getting exhausted and maintaining the incoming and outgoing balance on the channel. The existing routing protocol does not have self rebalancing factor. In case of a concurrent transaction, it is vital to maintain balance on channels, or many links will get exhausted after a few iterations. Iteration here refers to different timestamps, each timestamp handling multiple concurrent transactions. If the same link is used in a toggled way, first for receiving the transactions and then for sending the transactions, the balance on the channel can be maintained in both directions, as shown in Figure 4.2. In *iteration 1*, channels which are used for sending the transactions to landmark nodes, their direction is reversed in *iteration 2*; and vice versa. These directions are toggled in each iteration; thus, channel capacity will last forever. The benefit of self-rebalancing is that no extra effort is required to maintain the balance on channels; it happens automatically.

4.2 Main Design

The landmark routing-based protocols [61, 82] uses the BFS algorithm. But these protocols do not consider the dynamics of the network, such as channel capacities and load balancing on channels. Therefore, these algorithms fail to handle the concurrent transactions as discussed in Section 4.1. We need to improve routing tree algorithms so that they will consider all network dynamics. The average lifetime of the channel can be considered the same for time δt [107]. Let this time δt have m timestamps. At each timestamp, the landmark node will receive Tx containing n number of transactions. Now, the landmark node should choose a routing tree optimally to maximize the transactions with *feasiblePath* at each timestamp. Let set TF of size n, if their exist *feasiblePath* for $Tx_i \in Tx$,then $TF_i \in TF$ is 1 else 0. So

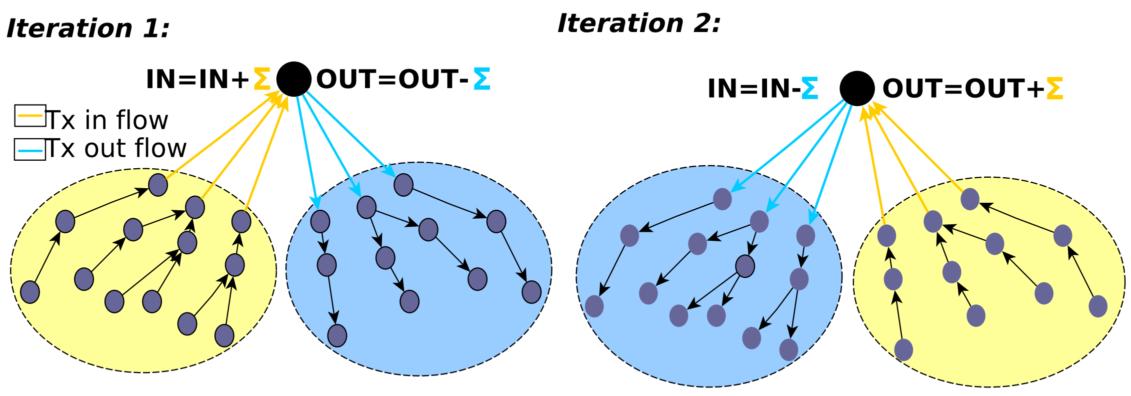


Figure 4.2: *Self Re-balancing*

we aim to maximize the total number of feasible transactions during time δt

$$Maximize \sum_{i=1}^{m}\sum_{j=1}^{n} TF_{ij} \tag{4.1}$$

We propose $maxECW$ and $maxRCL$ routing tree algorithms that try to meet objective 4.1 and provide their pseudo-codes in the Algorithm 4.1 and Algorithm 4.2, respectively.

4.2.1 Maximizing Flow by Exploiting Channel Weight (maxECW)

In Algorithm 4.1, we optimize routing trees by exploiting the weights on the channels. Input to this algorithm is the graph G, representing the lighting network and landmark node LM. In landmark routing, the tree is calculated using BFS. BFS iteratively adds neighbors to the tree irrespective of channel capacities or weight on the edges. The routing tree generated by BFS is used by landmark nodes for sending and receiving transactions by reversing the edges. In our algorithm, two different trees $t1, t2$ are generated for sending and receiving transactions to/from the landmark node.

Initially, l is added to queue Q. Then $maxECW$ starts traversing the graph using this Q until it is empty. Iteratively node u from Q is deleted, and its outgoing edges $e_{u,v}$ whose weight $w_{u,v}$ is greater than threshold (TH) is added to t_1, and endpoints v of edges are added to Q. TH is the average sum of weight on the outgoing edges. If there is an edge with a node v already in $t1$, that edge is not added to $t1$ to avoid cycles. After all these iterations, the remaining nodes $G - t_1$ are added to queue R. Each node u in queue R is added to $t1$ with highest possible edge weight $w_{u,v}$ $(v \in t_1)$ to $t1$ by referring graph G. Algorithm 4.1 returns t_1 for sending transactions from landmark nodes. The same algorithm is used for receiving transactions by considering incoming edges rather than outgoing ones, which will return the routing tree t_2. This way, landmark nodes have routing tree t_1, t_2;

Algorithm 4.1 maxECW

```
1: procedure MAXECW($G, l$)
2:     $Q \leftarrow \emptyset$ , $R \leftarrow \emptyset$ , $t1 \leftarrow \emptyset$
3:     $Q.enqueue(l)$                                          ▷ $l \in LM$
4:     while $Q \neq \emptyset$ do
5:         $u \leftarrow Q.dequeue()$
6:         $TH \leftarrow 0$
7:         $OD = outdegree(G, u)$
8:         for all out edges from $u$ to $v$ in $G$ do
9:             $TH \leftarrow TH + w_{u,v}$
10:        $TH = TH/OD$
11:        for all out edges from $u$ to $v$ in $G$, $v \notin t1$ do
12:            if $w_{u,v} > TH$ then
13:                add edge $(u, v)$ to tree t1
14:                $Q.enqueue(v)$
15:    $R \leftarrow Nodes(G) - Nodes(t1)$
16:    while $R \neq \emptyset$ do
17:        $max \leftarrow 0$
18:        $u \leftarrow R.dequeue()$
19:        for all in edges from $v$ to $u$ in $G$, $v \in t1$ do
20:            if $w_{v,u} > max$ then
21:                $idx = (v, u)$
22:                $max = w_{v,u}$
23:        add edge $(v, u)$ with $idx$ to tree t1
24:    Output=t1
```

these are used to provide routes to the transactions. *maxECW* maximizes the objective 4.1 by maintaining the good weight edges on the routing trees in each iteration/timestamp.

4.2.2 Maximizing Flow by Self Rebalancing Channel Load (maxSCL)

In Algorithm 4.1, we considered weights on the channel/ edges. In algorithm 4.2, we will try to rebalance the weight on the channels without considering the actual weight. We proposed the concept of Self-rebalancing, as explained in Section 4.1. In self-rebalancing, instead of waiting for the channel to get exhausted, it is better to refill it iteratively so that it will last for a longer duration or throughout the lifetime of the channel. As edges in the input graph, G is bidirectional, half of the edges of each node are used for sending transactions, and half of the edges will be used for receiving transactions. This way, two routing trees are generated t_1, t_2 for sending and receiving transactions. On each iteration/timestamp

Algorithm 4.2 maxSCL

```
1: procedure maxSCL($G, l$)
2:     $Q \leftarrow \emptyset$ , $R \leftarrow \emptyset$ , $t1 \leftarrow \emptyset$ , $t2 \leftarrow \emptyset$
3:     $Q.enqueue(l)$
4:     $G1 = G$
5:     while $Q \neq \emptyset$ do
6:         $u \leftarrow$ Q.dequeue()
7:         $OD \leftarrow$ outdegree(G,u)
8:         $check_1 = 1$
9:         for all out edges from u to v in G, $v \notin t1$ do
10:            if $check_1 \leq OD/2$ then
11:                add edge $(u, v)$ to tree t1
12:                $check_1 = check_1 + 1$
13:                remove edge $(v, u)$ from tree G1
14:                $Q.enqueue(v)$
15:            else
16:                break
17:    $R \leftarrow Nodes(G) - Nodes(t1)$
18:    while $R \neq \emptyset$ do
19:        $u \leftarrow$ R.dequeue()
20:        add random edge $(v, u)$ to tree t1,($(v, u) \in G$ ,$v \in t1$)
21:    $Q.enqueue(l)$
22:    while $Q \neq \emptyset$ do
23:        $u \leftarrow$ Q.dequeue()
24:        $ID \leftarrow$ indegree(G1,u)
25:        $check_1 = 1$
26:        for all in edges from v to u in G1, $v \notin t2$ do
27:            add edge $(u, v)$ to tree t2
28:            $Q.enqueue(v)$
```

direction of these routing, trees are reversed to balance the channel load, as shown in Figure 4.1b.

Initially, l is added to queue Q. Then $maxSCL$ starts traversing the graph using this Q until it is empty. Iteratively node u from Q is deleted, and half of the out-going edges $e_{u,v}$ are added to t_1, and endpoints v of edges are added to Q. If there is an edge with a node v already in $t1$, that edge is not added to $t1$ to avoid cycles. After this step, the remaining nodes $G - t_1$ are added to queue R. Each node u in queue R is added to $t1$ with a random edge from node u to node v in $t1$ by referring to graph G. During the processing of tree t_1, graph $G1$ is updated, which is initially equal to G. Every time, when out edge $e_{u,v}$ is

```
29:    $R \leftarrow Nodes(G) - Nodes(t1)$
30:    while $R \neq \emptyset$ do
31:        $u \leftarrow R.dequeue()$
32:        add random edge $(v, u)$ to tree t1,($(u, v) \in G$ ,$v \in t1$)
33:    Output=t1,t2
```

added to tree $t1$, the same edge in the opposite direction $e_{v,u}$ is deleted from graph $G1$. As in our algorithm, a node uses different neighbors for sending and receiving transactions. Now, this graph $G1$ is used as input for tree $t2$, which will follow the same procedure. On each iteration, remaining in-coming edges are added to tree $t2$. So, $maxSCL$ will return routing tree t_1, t_2; these are used to provide routes to the transactions. It is not required to recalculate routing trees in each iteration; just edge directions are reversed.

4.3 Experimental Evaluation

This section presents simulator $DRLNsim$ designed for distributive routing protocols in PCN. We compared our proposed algorithms with the landmark routing algorithm used by [61]. A brief description and flow of the simulator is explained below.

Architecture of DRLNsim

The architecture of the Distributive Routing Lightning Network simulator (DRLNsim) simulator is shown in Figure 4.3. There already exist Lightning Network simulators such as CLoTH [18] and PCNsim [78]. Still, these simulators cannot deal with distributive algorithms and concurrent transactions in a Lightning Network. DRSLsim is specific for distributive algorithms, where routing trees are used for routing a transaction from source to destination. Simulator input parameters can be modified to any distributive algorithm using the current implementation.

The simulations begin with input variables as described in Table 4.1. Then the Lightning Network is generated using function graph generation; this function takes input; node, avg_degree, e_weight_low, e_weight_high, and LM degree. For validation, we have referred to the input values from the actual Lightning Network [64, 85, 107]. For each iteration, routing trees are formed with existing landmark routing protocol (any distributed algorithm) and proposed algorithms $maxECW$ and $maxSCL$. The transaction generation function generates a new set of transactions in each iteration, fed to the concurrent feasible transaction function with the routing trees generated by routing algorithms. The concurrent feasible transaction function will find the path of each transaction in the transaction set and check whether it is feasible by checking the channel capacity constraint. This function will store

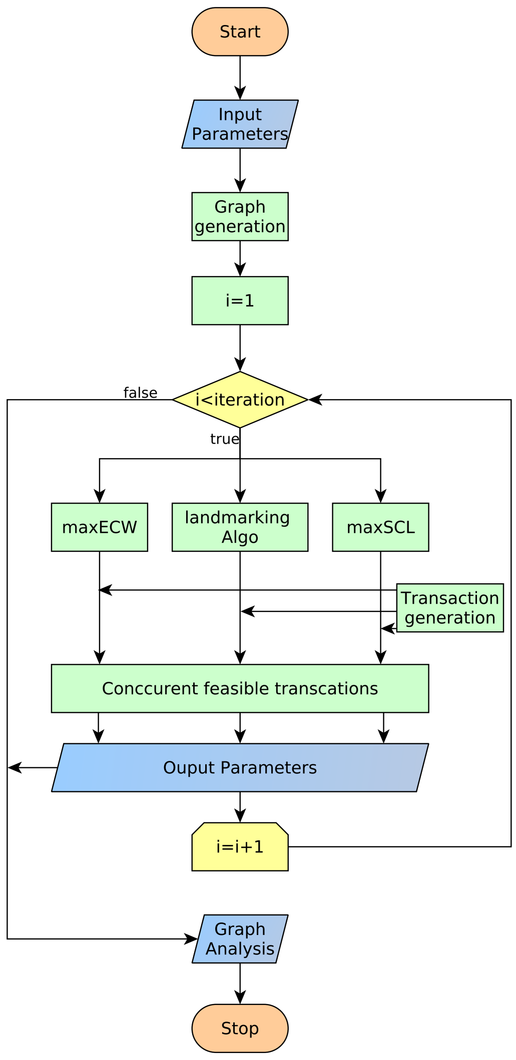


Figure 4.3: *Architecture of the simulator*

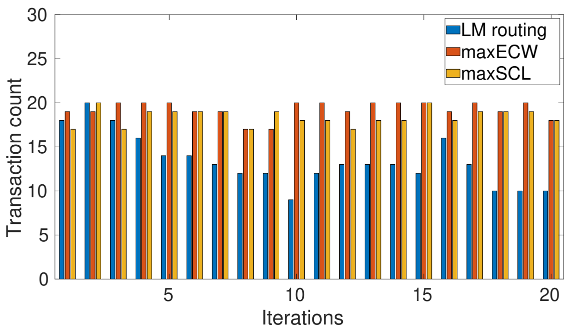
all the output parameters on each iteration, as shown in Table 4.1. When all iterations are done, all the output parameters are passed to the graph analysis function, producing all graphs corresponding to transaction volume, transaction count, and path length.

Network Topology

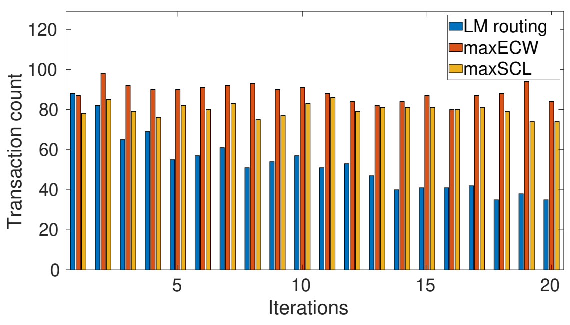
There is a centralized group of nodes in the network known as landmark nodes because nodes in the Lightning Network obey the power law, which states that nodes attempt to link the nodes with higher degrees [64]. This does not lead to complete centralization since the influential nodes alter over time [108]. In order to achieve this, we use the Barabasi-

Input	**Description**
Node	Number of nodes in the Lightning Network
Avg degree	Average degree of nodes in Lightning Network
e_weight_low	Lowest possible capacity of link in the network
e_weight_high	Highest possible capacity of link in the network
tx_number	Number of transactions in each iteration
tx_lower	Lowest amount being transferred
tx_higher	Highest amount being transferred
LM degree	Degree of landmark node
iteration	Number of timestamps in δt
Output	**Description**
Routing trees	Routing trees generated from all algorithms
feasible transaction	Binary array representing feasible/non-feasible transaction
concurrent_tx_count	Number of concurrent transactions in each iteration
Avg path length	Average path length of all feasible transactions
tx_volume	Transaction volume in each iteration

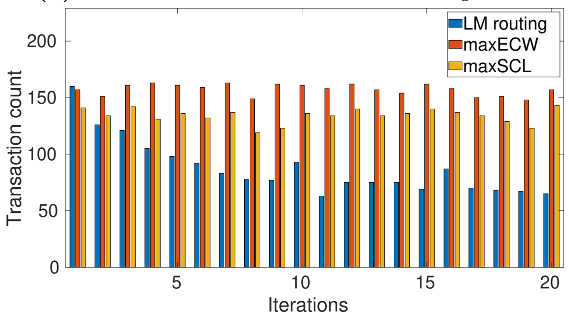
Table 4.1: *Simulator Parameters*



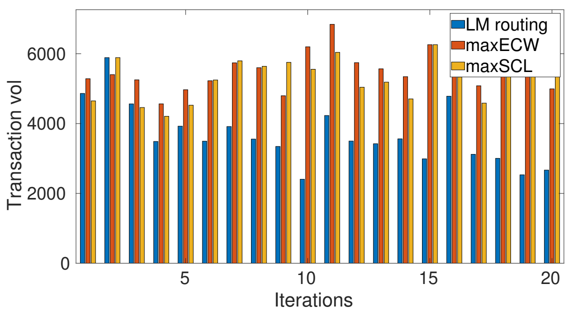
(a) *Nodes=2000 Tx num=20 LM deg=200*



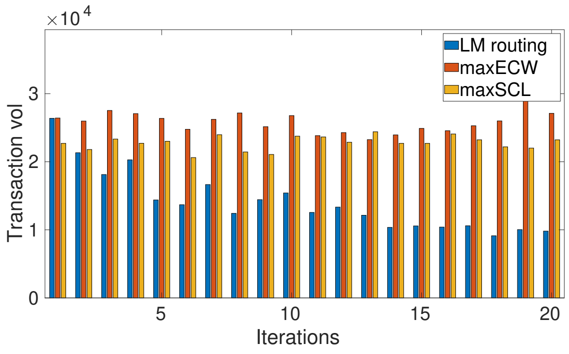
(b) *Nodes=3000 Tx num=100 LM deg=300*



(c) *Nodes=5000 Tx num=200 LM deg=500*



(d) *Nodes=2000 Tx num=20 LM deg=200*



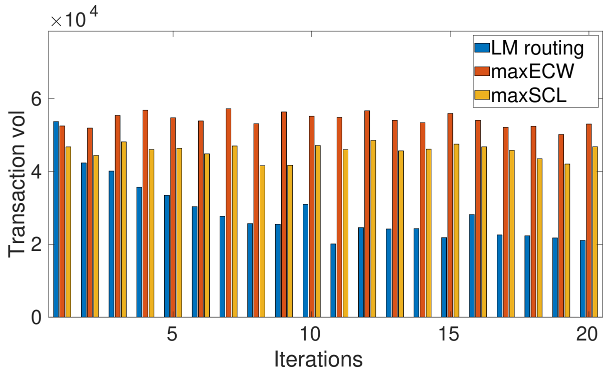
(e) *Nodes=3000 Tx num=100 LM deg=300*

Figure 4.4: *Transaction count, transaction volume, and average path length over iterations*

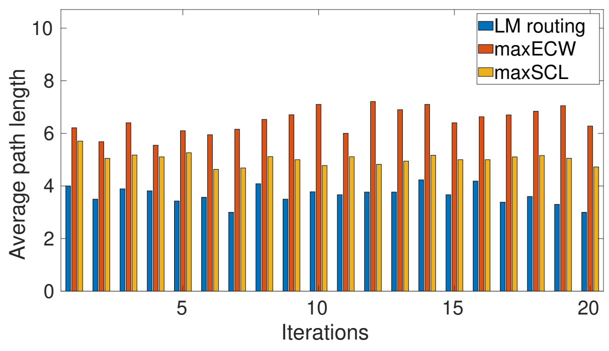
Albert (BA) model [2], which is based on the preferred attachment rule. Nodes are formed one at a time by connecting one or more edges to other existing nodes, using biassed random selection to increase the likelihood of a node with a higher node degree.

Network Parameters

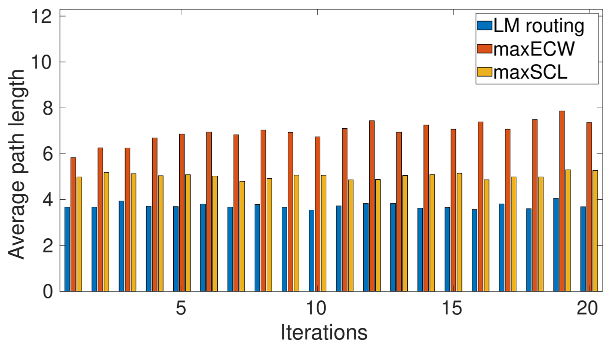
We have run different simulations by varying the simulator's input variables. The number of nodes varies from 1000 to 5000. Each channel's capacity is determined randomly between 500-1000; further, it is varied in the range 2000-2500. While the number of iterations is



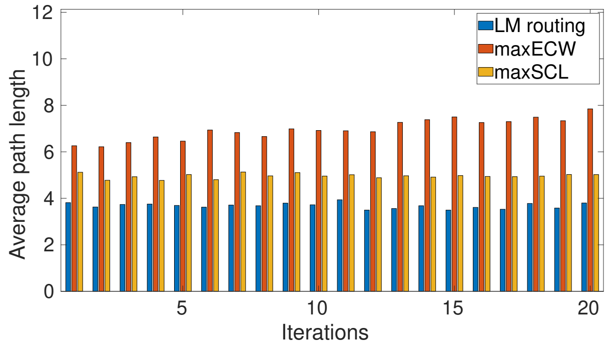
(f) *Nodes=5000 Tx num=200 LM deg=500*



(g) *Nodes=2000 Tx num=20 LM deg=200*



(h) *Nodes=3000 Tx num=100 LM deg=300*



(i) *Nodes=5000 Tx num=200 LM deg=500*

Figure 4.4: *Transaction count, transaction volume, and average path length over iterations*

set at 20, its value may be changed, but 20 timestamps are sufficient to determine how our algorithm behaves over time. A landmark node has around 20% payment channels in the network [17]. Accordingly, we have calculated the degree of each landmark node, which is around 10% of the nodes in the network. We varied landmark node degree till 25%, to observe the performance of our proposed work with a change in the network size. The average degree of non-LM nodes is set to seven [85].

Transaction Parameters

At each timestamp, the landmark node will receive 1000 randomly generated transactions. The sender-receiver pair is chosen at random for each transaction. Each transaction's fund is determined randomly between 100 and 500; further, it is varied till the size of channel capacity in [500-100].

Evaluation Metrics

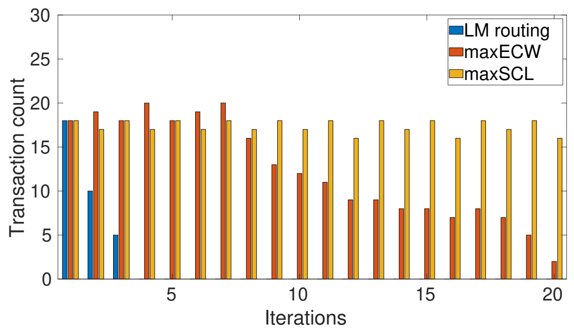
We use three evaluation metrics to compare the performance of our proposed algorithm:

Transaction count (TXC) the number of total feasible transactions that can find a reachable path in each iteration,

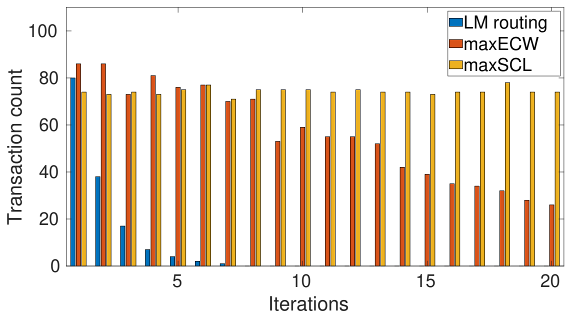
Transaction volume (TXV) the total sum of funds of feasible transactions in each iteration, and

Average path length (APL) the sum of paths of all feasible transactions over transaction count.

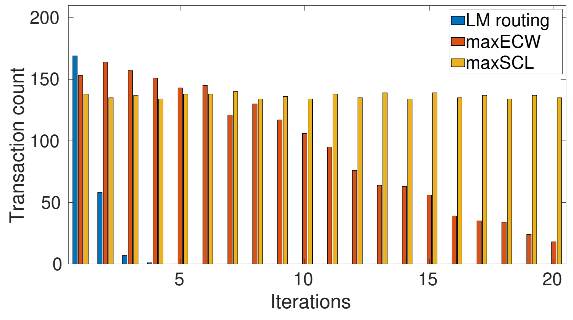
Transaction count gives the sustainability of the routing tree algorithm. Transaction count could be higher if the corresponding feasible transactions have fewer funds to transfer; thus, the transaction volume metric is required to validate the transaction count. The transaction fee is proportional to the number of intermediate hops between sender and receiver. So an efficient algorithm should have a high transaction count and volume and less average path length.



(a) *Nodes=2000 Tx num=20 LM deg=200*



(b) *Nodes=3000 Tx num=100 LM deg=300*



(c) *Nodes=5000 Tx num=200 LM deg=500*

Figure 4.5: *Transaction count in worst case when transaction set is same in all iterations*

Results

Figures 4.4a, 4.4b, and 4.4c show the transaction count on varying the number of nodes, LM degree, and number of transactions, while other input metrics are the same. In starting 2-3

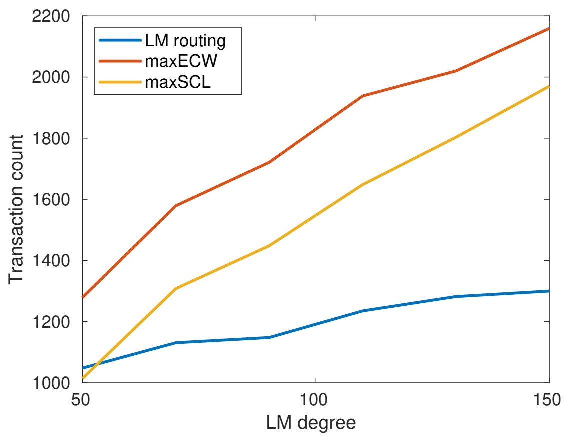
Algo/Nodes	2000	3000	5000
maxECW	43.6%	66.8%	79.96%
maxSCL	37.3%	50.09%	53.4%

Table 4.2: *Performance of the proposed Algorithms*

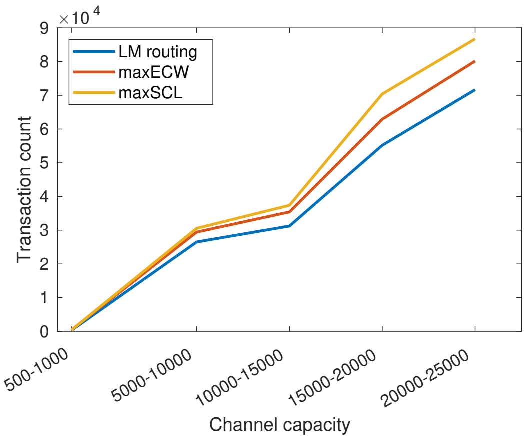
iterations, transaction count is almost the same in all algorithms. But as iterations proceed, the transaction count in landmark routing protocol decreases in all cases, as it does not use any technique to balance the channels, and weights are not considered while creating the routing tree. Transaction count does not drop to zero as some channels remain active due to random transactions that will revive the MSL channels. Both proposed algorithms perform better than landmark routing. *maxECW* performs better in all cases as it always chooses the good channels on the basis of channel weight. *maxSCL* performs almost the same in all iterations as it utilizes the concept of self-rebalancing. Figures 4.4d, 4.4e and 4.4f confirm that the transaction count in these cases corresponds to similar transaction volume. The performance of our proposed algorithms is shown in Table 4.2. Our proposed algorithms performed extremely well compared to the landmark algorithm used by [61, 82] for distributive routing. As the number of nodes and transaction number increases, our proposed algorithm does not fade, as they dynamically update with the iterations. In contrast, the landmark algorithm was not able to handle many transactions. Table 4.2 shows that our proposed algorithms performed far better than the existing ones in concurrent transactions over δt as described in Equation 4.1.

Case 1: The existing landmark routing algorithm is static and does not adapt to changes; in this case, we have considered the worst case to check the adaptability of the proposed routing algorithm. In the worst case, the transaction set in all iterations will be the same, and the rest of the simulation settings will be the same as above. Figure 4.5 shows the transaction count over iterations. The landmark routing algorithm survived a few iterations as its MSL channels got exhausted after a few iterations, while our proposed algorithm never got exhausted. However, transaction count in *maxECW* decreases over iterations compared to *maxSCL*. *maxSCL* remains stable through all iterations, as different channels are used over different iterations so that no channel will be exhausted.

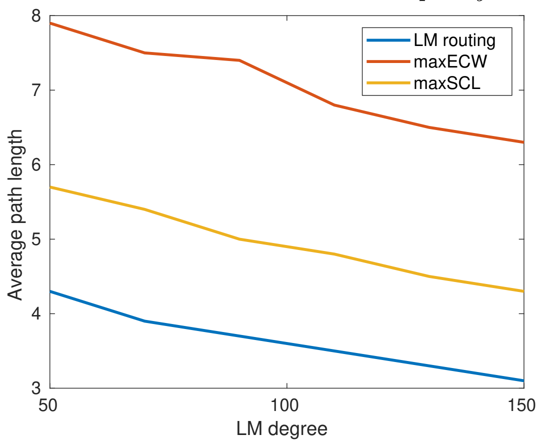
Case 2: Figure 4.6a shows the transaction count over δt with a change in degree; all other input metrics are the same during the iterations. This simulation is conducted for 1000 nodes, with an average degree, number of iterations, tx_number, and channel capacity as 7, 20, 200, and 500–1000, respectively; the LM degree is varied from 50 to 150. It is clear from the plot that the distance between curves of the existing algorithm and proposed algorithms increases drastically with a change in degree. It shows the importance of MSL in the Lightning Network. As we know, the landmark algorithm follows BFS; in BFS, the LM node has all its neighbors in its routing tree, whereas in our proposed algorithm, the



(a) *Transaction count over δt with change in degree*



(b) *Transaction count over δt with change in channel capacity*



(c) *Average path length over δt with change in degree*

Figure 4.6: *Transaction count and path length with change in parameters*

degree of the LM, the node is less than that of the landmark routing tree; then, too, they are performing very well. This concludes that selecting good channels is required to avoid congestion at MSLs; therefore, *maxECW* always performs best out of three as it always chooses the best channels in the network.

Case 3: Figure 4.6b shows the transaction count over δt with a change in channel capacities; all other input metrics are the same during the iterations. In this simulation, the number of nodes is 2000 with the average degree, the number of iterations, tx_number, and the LM degree as 7, 20, 5000, 200, respectively, where the channel capacity varies from 500 to 25000. We can see that as the channel capacities increase, the distance between curves increases, and in all cases, our proposed algorithms are performing well. If we increase the iteration value, this distance between curves will further increase. But in this case, as compared to case 1, *maxSCL* performs well compared to *maxECW*. From this, we can conclude that channel balancing plays a pivot role when channel capacities are very high so that more transactions can flow through the network.

In the *maxECW* algorithm, the LM node needs to update all channel capacities with time; a good gossip protocol is required. Moreover, we need to re-calculate the routing tree over all the iterations. However, if an LM node does not want much effort, then *maxSCL* protocol can be used as there are no channel weights required in calculating the routing tree, and there is no need to re-calculate the routing tree on each iteration. The transaction fee is directly proportional to the number of intermediate nodes in the network. Figures 4.4g, 4.4h, and 4.4i show that the average path length in *maxECW* is maximum, the in *maxSCL*, and the least in landmark routing.

Landmark routing uses BFS, which gives the shortest path from the landmark node. As *maxECW* exploits the channels' weights, poor channels are not added to the routing tree. Thus, it will increase the path length of those nodes in the network. *maxSCL* uses the concept of rebalancing; thus, its average path length is less as compared to *maxECW*. Moreover, the reason for the increase in path length is the number of transactions; the transactions that are not feasible in a landmark routing tree generally will take a longer route in the proposed algorithms. Figure 4.6c shows that with the increase in LM degree, and the difference between all curves will remain the same. Nodes in the Lightning Network will charge a base fee and fee rate for transferring transactions through it [50]. The node can update fee policies dynamically. The base fee can be reduced by Lightning nodes as it can be compensated by more transaction flow. Moreover, a negative fee [105] is also used by nodes to revive the exhausted channel, but in our proposed algorithm, the channel will not get exhausted, so there is no need for a negative fee. The selection of the proposed algorithm can be made based on the topology of the Lightning Network and the computational power of the landmark nodes.

4.4 Conclusions

We have discussed the importance of MSL channels in the Lightning Network and have proposed two algorithms, *maxECW* and *maxSCL*, to maximize the number of concurrent transactions in PCNs. To evaluate the performance of these algorithms, we designed a *DRLsim* simulator for distributive routing algorithms in PCNs. In contrast to the existing algorithms, which cannot handle concurrent transactions and do not adapt to the changes in the network, our algorithms are adaptive and take into account the dynamic nature of the channel capacities. The self-rebalancing of channels helps in avoiding the saturation of capacities over time. Our experimental results have shown that the proposed algorithms performed significantly better than the current state of the art. In the subsequent chapter, we will explore the formulation of a novel algorithm that integrates the advantages of both existing algorithms while minimizing total overhead.

5

maxREE: Maximizing Flow by Replacing Exhausted Edges

This chapter extends our previous work explained in Chapter 3 where we proposed *max-ECW* and *maxSCL* dynamic algorithms to improve concurrency by keeping track of the channel capacities using a Self-Rebalancing Factor (SRF). These earlier algorithms improve the concurrent execution of transactions, but these algorithms increase the average path length. Moreover, algorithms have other overheads, such as; updating routing trees on each iteration and ignoring edges that have not recently been used. In this chapter, we present a distributed routing technique that maximizes the efficiency of concurrent execution of multiple transactions with negligible overhead. Our proposed technique uses the concept of selective edge-replacement, which guarantees that channels will always remain balanced. Utilizing the edge coloring method, we provide the *maxREE* algorithm, which maintains optimum channels inside the routing tree. Our suggested algorithm's load balancing and congestion at the landmark node are both studied theoretically. We validate the Lightning Network's routing tree algorithm's performance using simulations run on *DRLNsim*.

The rest of this chapter is organized as follows: Section 5.1 presents limitations in protocols proposed in Chapter 3. The proposed algorithm and its theoretical analysis are described in Section 5.2 and 5.3, respectively. Section 5.4 presents the performance evaluation of the proposed algorithm, and we finally conclude in Section 5.6.

5.1 Motivation

Our previously proposed algorithms are dynamic and update routing trees to improve the concurrency of transactions. However, these algorithms update the routing trees even when not required, leading to tree construction overhead. The $maxECW$ algorithm uses the gossip protocol to get the channel balance to update the Lightning graph. Moreover, the concurrency improvement increases transaction path length, leading to high transaction fees. In this work, we retain the concurrency and reduce the transaction path length and other overheads. To our knowledge, there is no protocol for distributed PCN that takes into account concurrent execution of transactions while considering other constraints.

Next, we will examine how our previous proposed algorithms tried to maximize Equation 4.1 (explained in Chapter 4) by enhancing routing tree methods and addressing their shortcomings before moving on to our proposed approach, $maxREE$.

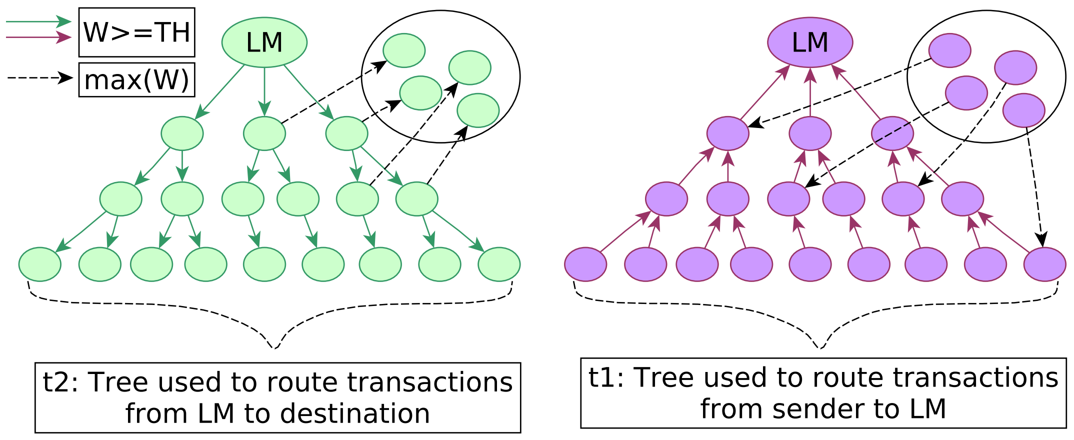

W>=TH
max(W)
LM
LM
t2: Tree used to route transactions from LM to destination
t1: Tree used to route transactions from sender to LM

Figure 5.1: $maxECW$

Our previous work proposed $maxECW$ and $maxSCL$ routing tree algorithms using the bidirectional property of $G(E)$. Both algorithms $maxECW$ and $maxSCL$ use two different trees for sending and receiving the transactions as shown in Figure 5.1 and 5.2, respectively. The $maxECW$ algorithm exploits the $\mathbb{B}$, and only the edges with weight greater than the threshold (TH)are added to the routing tree, and remaining nodes with weight less than TH are added with the maximum possible weight. On the other hand, $maxSCL$ used self-rebalancing to maintain the balance on the channel throughout the channel's lifetime. The edges of trees are reversed in each timestamp to avoid depletion of $\mathbb{B}$, as shown in Figure 5.2. Both algorithms performed well compared to the existing landmark routing protocol. $maxECW$ performs best but with an increase in average path length. Moreover, it uses a gossip protocol to update the balance, which will alter the privacy of networks. This work proposes a $maxREE$ routing tree algorithm to meet objective 4.1 and optimize the path

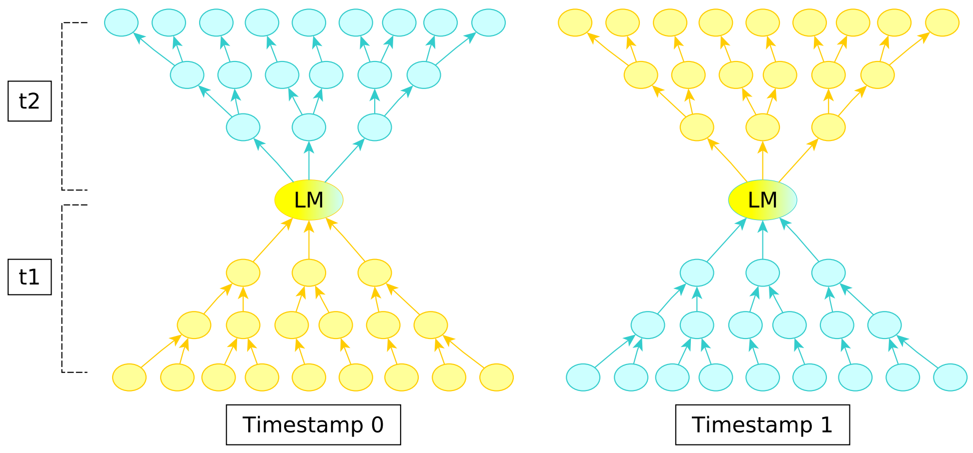


Figure 5.2: *Tree structures for two timestamps in maxSCL*

length. $maxREE$ algorithm exploits the edge coloring; there are several advantages to edge coloring, especially when it comes to problem solving. It helps differentiate between various connections, optimize resource allocation, and resolve scheduling and network issues by giving edges distinct colors. Edge coloring optimizes the channel allocation in our proposed algorithm to regulate the flow of transactions. The steps for $maxREE$ pseudo-code are described in Algorithm 5.1. Table 5.1 explains symbols used in the algorithm.

5.2 Maximizing Flow by Discarding Exhausted Edges (maxREE)

The main idea in Algorithm 5.1, is to update routing trees by replacing the exhausted edges. Each landmark node ($l \in LM$) runs Algorithm 5.1 to get the spanning tree. In landmark routing, BFS is used to calculate the tree. Without regard to channel capacity or edge weight, BFS iteratively adds neighbours to the tree. A landmark node can send and receive transactions by flipping the edges of the routing tree produced by BFS. Our approach produces two different trees, t_1 and t_2, to transmit and receive transactions to/from a landmark node. Our previously proposed algorithms, $maxECW$ and $maxSCL$, updated routing trees t_1 and t_2 on each timestamp. However, there is no need to update the whole routing tree; only the nodes that get exhausted condition to be replaced.

Initially, in timestamp 1, tree t_1 is generated using BFS, and all the edges in the routing tree and graph G are colored green, as the input graph is the same, and there is no edge update. But from the second timestamp, a landmark node, l, will receive the messages from the nodes, and about the edges, those weights are reduced by the component $\frac{1}{SRF}$, as described in Algorithm 5.2. The red and green edges list of t_1 is maintained by Q_r

Input	**Description**
G	Lightning graph
v, u	$v, u \in G(V)$
(u, v)	$(u, v) \in G(E)$
$BFS(G, l)$	returns breath first search tree rooted at node l in graph G
$level(u, t_1)$	returns level of node u in tree t_1 with l as root node
$CC(t_1, l)$	connected component of t_1 with node l
$index$	refers to the location of the respective edge in edgeList or edgeReplace
$\mathbb{C}(u, v)$	capacity of edge (u, v) when created
$\mathbb{B}(u, v)$	balance on edge (u, v) when any transaction get processed
$ID(G, w)$	indegree of node w in graph G
Q_r, Q_g	Queue

Table 5.1: *Terms in Algorithm 5.1*

and Q_g, respectively. The edges that will turn red in the routing tree t_1 are replaced with possible green edges in graph G. The red edges will be removed from tree t_1, which forms two connected components. A connected component with the landmark node l can be considered as the main component of t_1 and others may be considered as a subtree.

The algorithm aims to add the new edge to join the connected components such that the level of the subtree is the lowest among all its neighbours. *edgeList* and *edgeReplace* are used to maintain the original and replaced edges in t_1. The edges in Q_g contain the actual edges of the tree t_1, which earlier turned to red but now become green. So, these original edges are again added to the tree t_1 by replacing the edges in the *edgeReplace* list. This way, landmark nodes update the routing tree t_1. Similarly, a routing tree t_2 is maintained where in place of indegree; outdegree is exploited as directions of edges are reversed. An example demonstrating the *maxREE* algorithm is shown in Fig 5.3.

In our proposed routing algorithm, two routing trees are used for sending and receiving transactions from a landmark node, and there is no compulsion to use the same channel for sending as well as receiving transactions. As demonstrated in Figure 5.3, when a channel in routing tree t_2 is depleted, it is substituted with an active channel. The figure further illustrates how tree t_2 has changed across various timestamps. In the same way, when updating tree t_1, all the parameters are preserved. Because t_1 uses channels in the opposite direction from t_2, the channels that are exhausted in the graph maintained for updating the tree t_2 are active in the graph used to update tree t_1. Thus, when updating tree t_2, the depleted channels of tree t_1 can be utilized, and vice versa. Both trees will assist

Algorithm 5.1 maxREE

```
procedure maxREE($G$, $l$, $SRF$)
    $Q_g \leftarrow \emptyset$ , $t_1 \leftarrow \emptyset$ , $Q_r \leftarrow \emptyset$
    edgeList $\leftarrow \emptyset$ , edgeReplace $\leftarrow \emptyset$
    for each $(u, v) \in E$ do
        $(u, v).color \leftarrow GREEN$
    $t_1 \leftarrow BFS(G, l)$                                        ▷ $l \in LM$
    for each $(u, v) \in E'$ do                                       ▷ $t_1(E')$
        edgeList.append((u, v))
        edgeReplace.append($\perp$)
    for $j = 2$ to $m$ do
        $[G, t_1, Q_r, Q_g] \leftarrow$ UpdateGraph($G$, $l$, $SRF$, $t_1$)
        while $Q_r \neq \emptyset$ do
            $(u, v) \leftarrow Q_r.dequeue()$
            remove edge $(u, v)$ from $t_1$
            $p$, $q \leftarrow u$, $v$
            if $u \notin CC(t_1, l)$ then $p$, $q \leftarrow q$, $p$
            $ckLevel \leftarrow level(p, t_1)$
            $w$, $x$, minLevel $\leftarrow q, \perp, \infty$
            for each $z \in ID(G, w)$ do
                if $z \notin CC(t_1, l)$ or $(z, w).color = RED$ then continue
                tpl $\leftarrow$ level($z, t_1$)
                if tpl == ckLevel then
                    $x \leftarrow z$
                    $break$
                else if tpl < minLevel then
                    $x$, minLevel $\leftarrow z$, tpl
            if $x = \perp$ then
                add edge $(u, v)$ to $t_1$
                continue
            add edge $(x, w)$ to $t_1$
            if $(u, v) \in edgeList$ then
                index $\leftarrow$ edgeList.indexof($(u, v)$)
            else
                index $\leftarrow$ edgeReplace.indexof($(u, v)$)
            edgeReplace[index] $\leftarrow (x, w)$
        while $Q_g \neq \emptyset$ do
```

```
37:            $(u, v) \leftarrow Q_g.dequeue()$
38:            index $\leftarrow$ edgeList($(u, v)$)
39:            remove edge edgeReplace[index] from $t_1$
40:            add edge edgeList[index] to $t_1$
41:            edgeReplace[index] $\leftarrow \perp$
```

in replenishing each other's depleted channels over the timestamps. Therefore, without the need for outside assistance, the SRF will aid in the updation of trees and channel rebalancing.

In the case of routing trees t_1 and t_2, only the selected edges are replaced instead of generating a new tree on each iteration, as done by our previous proposed algorithms. The overhead of tree formation is decreased by this selective edge replacement. Furthermore, in routing trees, replacing an edge would only lengthen the path taken by transactions passing through that specific edge. When the original edge regains its capacity with the aid of self-rebalancing, this impact is reversed. Because the same edges in the LM routing tree are utilized for both sending and receiving transactions, there is link fatigue and congestion, and because channel balancing is unavailable, the tree remains static. In our suggested technique, the utilization of distinct channels in both trees aids in channel rebalancing, and SRF is employed to identify the fatigued links, which further aids in edge replacement.

5.3 Theoretical Analysis

We now compare our suggested algorithm's performance and overhead to the existing landmark routing method. As noted in Section 4.1, the concurrent flow of transactions at MSL will be impacted by congestion. We will examine the overhead in terms of the path length owing to the edge replacement policy and see how our suggested algorithm performs in terms of transaction flow when some of the MSLs get exhausted at a particular time.

Both active and passive techniques can be utilized for classifying load-balancing mechanisms [39]. The routing algorithm's built-in structure is used by passive load balancing to distribute traffic evenly across the network. On the other hand, active load balancing uses real-time traffic data to reroute traffic away from congestion areas. Silent Whisper [61] and Speedy Murmurs [82] both utilise landmark routing which is a passive technique. Our proposed algorithm is an active technique that dynamically makes changes to the routing tree to avoid congestion and improve concurrency. We assume an appropriate metric space $\mathbb{D}$ is defined from $V \times V \rightarrow \mathbb{R}$.

Let $T^{'}(V, E^{'}, W^{'})$ be the coordinate tree built by a landmark node, l, using the spanning tree algorithm. If a transaction is sent by a sender s to a receiver t, the number of hops travelled by the transaction is given by:

$$\mathbf{d}(s, t) = \mathbf{d}(s, l) + \mathbf{d}(l, t) \tag{5.1}$$

Algorithm 5.2 Graph Updation Algorithm

```
1: procedure UpdateGraph($G$, $l$, SRF, $t_1$)
2:     for each $u \in V - \{l\}$ do
3:         for each $v \in$ adj(u) do
4:             $v_{low}, v_{high} \leftarrow 0, 1$
5:         for each $v \in$ adj(u) do                                    ▷ message send by u
6:             if $SRF \times \mathbb{B}(u,v) < \mathbb{C}(u,v)$  &  $v_{low} = 0$ then
7:                 SendMessage($u$, $v$, $l$, low)
8:                 $v_{low}, v_{high} \leftarrow 1, 0$
9:             if $SRF \times \mathbb{B}(u,v) \geq \mathbb{C}(u,v)$  &  $v_{high} = 0$ then
10:                SendMessage($u$, $v$, $l$, high)
11:                $v_{low}, v_{high} \leftarrow 0, 1$
1: procedure ReceiveMessage($u$, $v$, $l$, action)                       ▷ at $l$
2:     setColor ← RED if action = low else GREEN
3:     $(u,v).color \leftarrow$ setColor                                ▷ $(u,v) \in E$
4:     if $action = low$ and $(u,v) \in E'$ then                        ▷ $t_1(E')$
5:         $(u,v).color \leftarrow$ setColor
6:         $Q_r.enqueue(u,v)$
7:     if $action \neq low$ and $(u,v) \in edgeList$ then
8:         index=edgeList$(u,v)$
9:         if $edgeReplace[index] = 0$ then
10:            $(u,v).color \leftarrow$ setColor
11:        else
12:            $Q_g.enqueue(u,v)$
```

Spanning tree has $|V-1|$ edges, which is typically around 25% of edges in G [49]. The Landmark algorithm is passive in nature due to which an edge $e \in E - E'$ will remain unused. When concurrent transactions are routed, congestion will occur at weak links; and as we move higher up the tree, congestion gets higher since each node forwards the transaction to its parent. Our proposed algorithm replaces the congested edges in the routing tree with edges that are not part of the tree, that is, $(E - E')$. Let N be the total number of nodes in the spanning tree excluding landmark node l and k_{ld} be the degree of the landmark node. Further, let k_{ad} be the average degree of other nodes and l be the level in the spanning tree. Then,

$$N \approx k_{ld} \left(\frac{k_{ad}^{l} - 1}{k_{ad} - 1} \right) \tag{5.2}$$

Assuming that each node can begin only one transaction at a time instant. The total

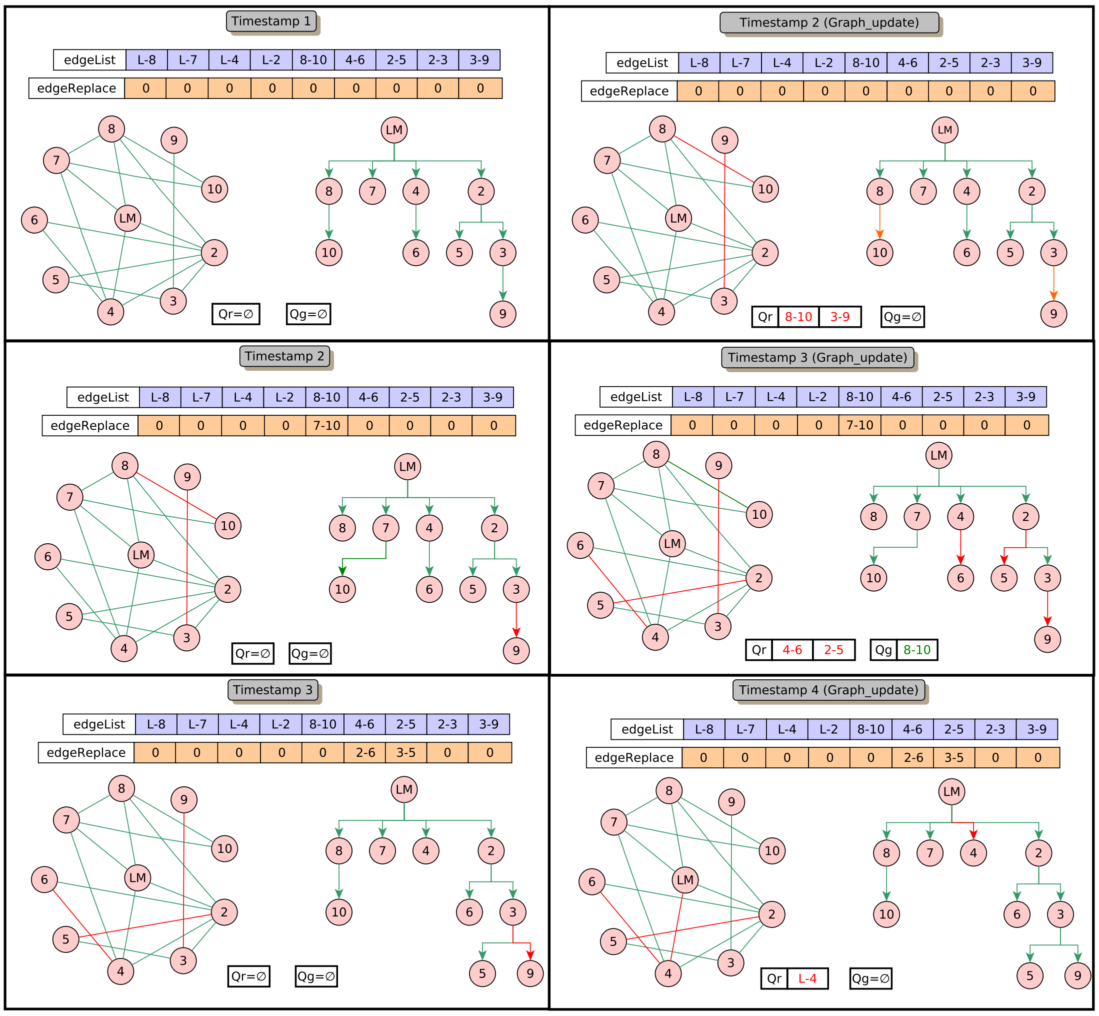


Figure 5.3: *Demonstration of maxREE*

At *timestamp 1*, BFS is used to build a routing algorithm. Initially, Q_r and Q_g are $\emptyset$. Then, at *timestamp 2*, two edges $8-10$, $3-9$ turn red and are added to Q_r. In processing Q_r, $8-10$ is replaced with $7-10$, but there is no alternate for $3-9$, so it remains unaltered. Parallel *edgeList* is updated. At *timestamp 3*, two edges $4-6$, $2-5$ turn red and are added to Q_r, and edge $8-10$ turns green and added to Q_g. Similarly, Q_r is processed in *timestamp 3*. Edge $8-10$ in Q_g is the original edge in *edgeList*, so it is replaced with the corresponding edge in *edgeReplace*, and *edgeReplace* is updated. At *timestamp 4*, edge $9-3$ turns green, but it is not added to Q_g, as its edge replace index is 0.

number of transactions at any timestamp is:

$$n = \sum_{i \in N} \lambda_i \quad \lambda_i \in \{0, 1\} \tag{5.3}$$

5.3.1 Congestion and Load Sharing at Landmark Node

As nodes in the LN network follow the power law, the value of $k_{ld} \approx 10\%|V|$, thus landmark node has the highest betweenness. By counting the paths from each node in the tree to the various first-order subtrees that must travel through the root, one can determine the betweenness B_l of the root.

$$B_l = \frac{1}{2}\frac{N}{k_{ld}}\frac{(k_{ld}-1)N}{k_{ld}}k_{ld} \tag{5.4}$$

$$B_l = \frac{k_{ld}(k_{ad}^l - 1)^2}{2(k_{ad}-1)} \tag{5.5}$$

The number of nodes in one selected first-order subtree is represented by the factor $\frac{N}{k_{ld}}$, while the number of nodes in all other $k_{ld}-1$ first-order subtrees is represented by the factor $\frac{(k_{ld}-1)N}{k_{ld}}$. The product of these two elements yields the total number of shortest paths from the selected first-order subtree to any other first-order subtree. We can choose a first-order subtree in exactly k_{ld} ways. Because each shortest path has been counted twice, the factor 1/2 is included. Total transaction flow, Q_l, towards the landmark node can be referred to as:

$$Q_l = nc\frac{k_{ld}(k_{ad}^l - 1)^2}{2(k_{ad}-1)} \tag{5.6}$$

Here c represents the average transaction capacity. The total volume of transactions is upper bound by:

$$Q_l \leq \sum_{u \in adj(l)} \mathbf{min}(\mathbb{B}(l,u), \mathbb{B}(u,l)) \tag{5.7}$$

Let us suppose that the average channel capacity is w_{avg}

$$Q_l \leq k_{ld} \times w_{avg} \tag{5.8}$$

So, the transaction flow, Q_l, is directly proportional to landmark degree (k_{ld}) and channel capacities (w_{avg}). If the $Q_l > k_{ld}w_{avg}$, then some transactions will be aborted, and all n transactions will not be feasible. The feasibility of transactions also depends upon the value of transaction capacity c, if the value of c is more then the number of feasible transactions will decrease. Our proposed algorithm aims to maximize the number of feasible transactions in time δt and thus maximize (Q_l) at each timestamp, by swapping the congested edges.

Assume that α the number of channels gets congested or exhausted at timestamp one. Then in the next timestamp, the transactions from these α links will be aborted due to these congested links. However, our proposed algorithm will remove the congested edges from the spanning tree and replace them with active edges. So the aborted transaction in the standard landmark spanning tree will be routed through the remaining $k_{ld} - \alpha$ links. Let us suppose S be the set containing nodes from the respective α subtrees. This will be the optimization of our proposed algorithm over the standard landmark spanning tree given by the following expression:

$$\left(\sum_{i \in S} \lambda_i\right) c \frac{\alpha(k_{ad}^l - 1)^2}{2(k_{ad} - 1)} \tag{5.9}$$

However, the above increase in the number of successful transactions may also increase the path length of the transactions optimized by our proposed algorithm. If the sender s of transaction belongs to α subtree and receiver t belongs to $(k_{ld} - \alpha)$, then the increase in path length will be 1. But if both s and t belongs to α subtree then increase in path length will be 2

$$1 < \Delta \mathbf{d}(s,t) \leq 2 \tag{5.10}$$

Further, in the next timestamp, some of the channels may restore channel capacities, and some more channels will be congested. Assuming that the transaction flow is enough that the number of congested channels will increase with the timestamp, then the optimization will increase with the SRF α. Basically, unused links of previous timestamps provide improvement or load sharing to transactions in the current timestamp.

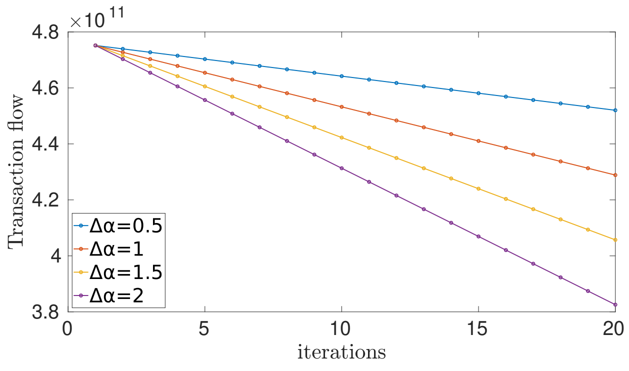
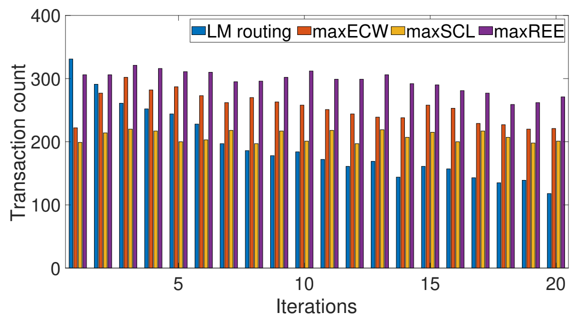
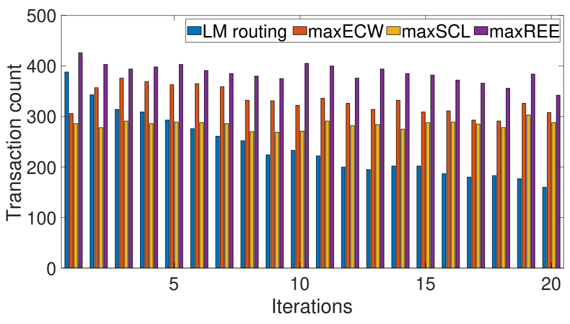


Figure 5.4: *Variation of transaction flow with time*

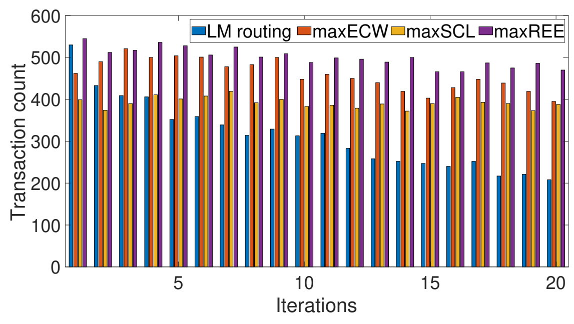
Figure 5.4 shows the decrease in transaction flow/volume, assuming $\alpha = 5$ at the first timestamp, $\Delta\alpha$ refers to an increase in α in succeeding timestamps. So with an increase in α transaction volume decrease in landmark routing, however, our proposed algorithm will try to maintain the transaction volume by rerouting transactions through other links. But in the actual scenario, the congestion links may increase/decrease with the random flow of transactions, so the decrease in transaction flow will not be exactly a straight line as shown in Figure 5.4. More analysis regarding theoretical analysis with actual simulation results is discussed in Section 5.4.



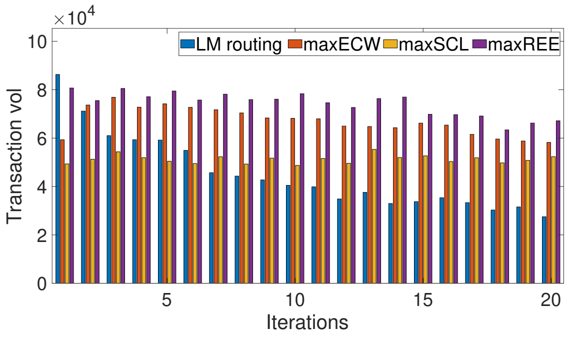
(a) *Nodes=2000 Tx =1000 LM deg=200*



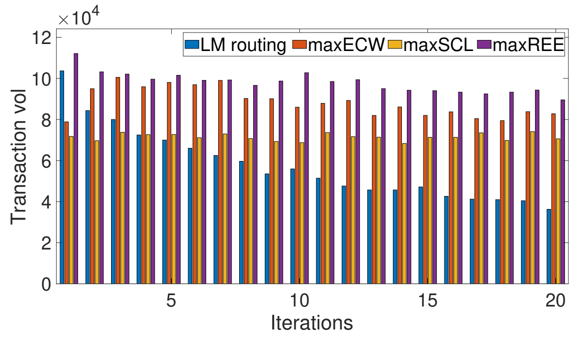
(b) *Nodes=3000 Tx =1000 LM deg=300*



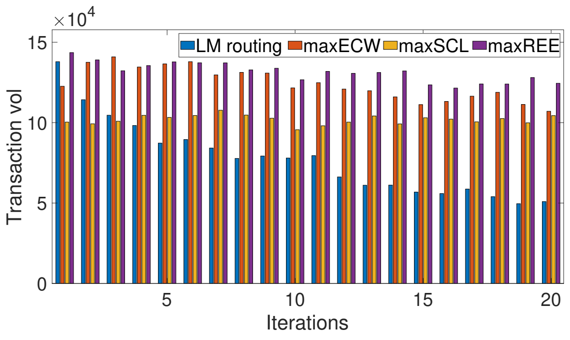
(c) *Nodes=5000 Tx =1000 LM deg=500*



(d) *Nodes=2000 Tx =1000 LM deg=200*



(e) *Nodes=3000 Tx =1000 LM deg=300*

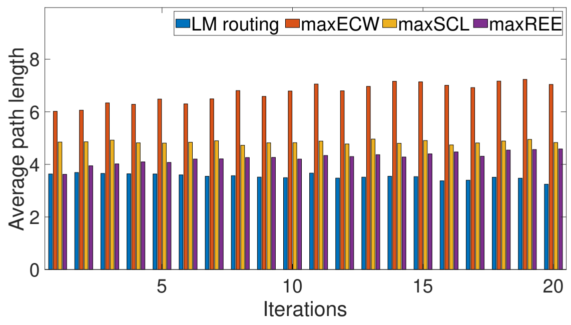


(f) *Nodes=5000 Tx =1000 LM deg=500*

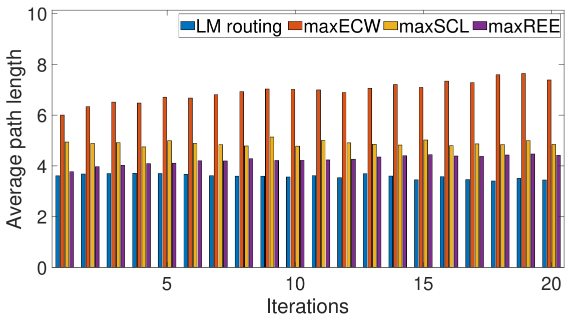
Figure 5.5: *Transaction count, transaction volume, and average path length over iterations*

5.4 Results

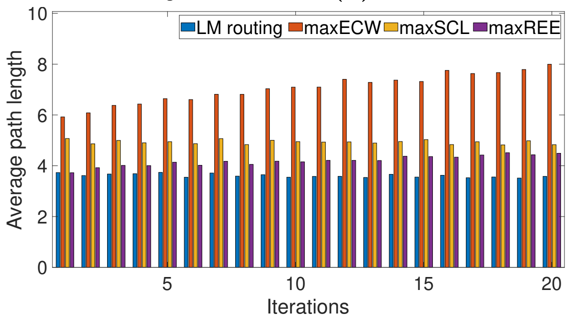
The simulation setup is the same as in Chapter 3. Figures 5.5a, 5.5b, and 5.5c show the transaction count on varying the LN network, LM degree, and number of transactions, while other input metrics are kept fixed. The transaction count in LM routing is acceptable for the first few iterations. However, as iterations continue, the landmark routing protocol's transaction count decreases in every situation since it neither employs any method to balance the channels nor considers weights when building the routing tree. Due to sporadic transactions that will reactivate the MSL channels, the transaction count does not entirely



(g) *Nodes=2000 Tx =1000 LM deg=200*



(h) *Nodes=3000 Tx =1000 LM deg=300*



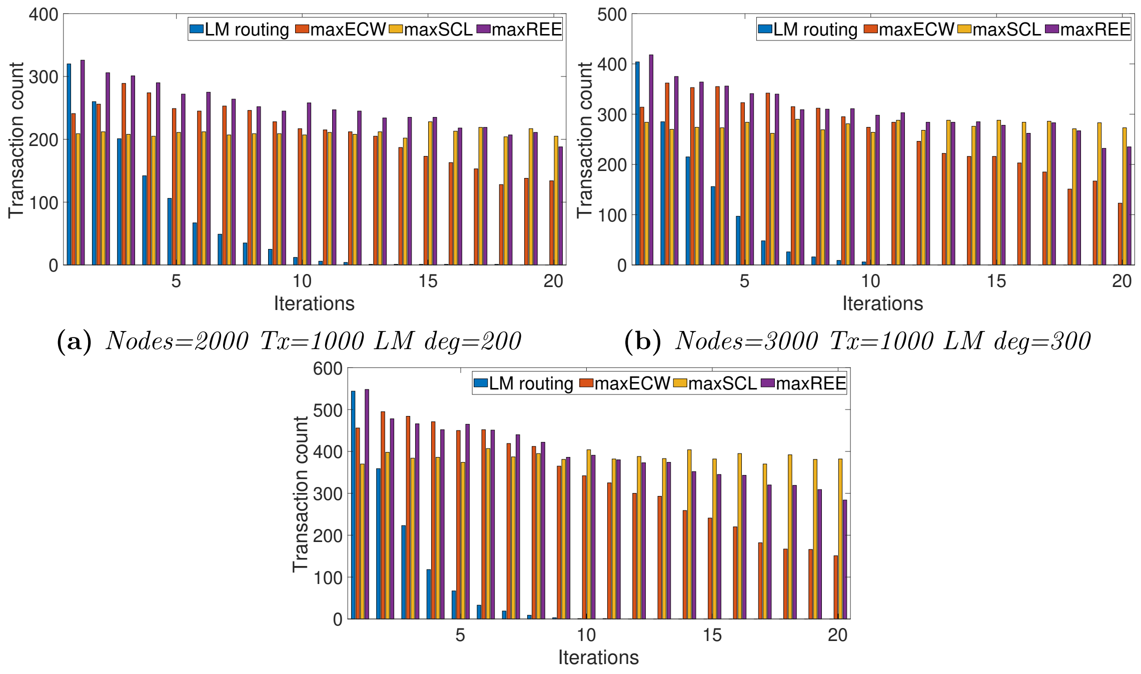
(i) *Nodes=5000 Tx =1000 LM deg=500*

Figure 5.5: *Transaction count, transaction volume, and average path length over iterations*

disappear.

In every situation, $maxECW$ outperforms $maxSCL$ and LM routing because it always selects the best channels based on channel weight. Due to the notion of self-rebalancing, $maxSCL$ performs almost the same throughout all iterations. But our newly proposed algorithm $maxREE$ outperformed the LM routing, $maxECW$, and $maxSCL$ in all iterations. When building the routing tree, $maxECW$ adopts the greedy strategy of only considering excellent channels, which excludes specific decent channels. While $maxSCL$ flips the edge orientation during each iteration to achieve rebalancing, doing so will result in lower channel utilization. Our proposed approach, $maxREE$, selectively replace edges whose values fall below a specific threshold, maintaining both good channels and the self-rebalancing factor, outperforming competing algorithms. Figures 5.5d, 5.5e and 5.5f attest to the fact that in these instances, the transaction count and volume are similar.

Figures 5.5a, 5.5b, and 5.5c show the average path length on varying the LN network. Landmark routing uses BFS, which gives the shortest path from the landmark node. As $maxECW$ exploits the channels' weights, poor channels are not added to the routing tree. As a result, it will lengthen those nodes' paths through the network. $maxSCL$ uses the concept of rebalancing; thus, its average path length is less as compared to $maxECW$. The number of transactions is another factor contributing to the rise in path length; in the suggested methods, transactions that are not viable in a landmark routing tree often



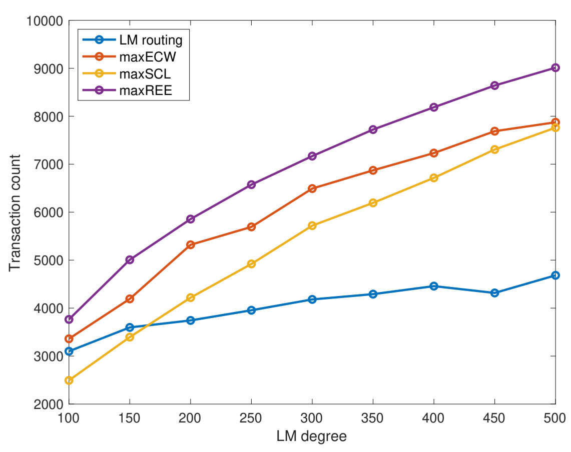
(a) *Nodes=2000 Tx=1000 LM deg=200* **(b)** *Nodes=3000 Tx=1000 LM deg=300*

(c) *Nodes=5000 Tx=1000 LM deg=500*

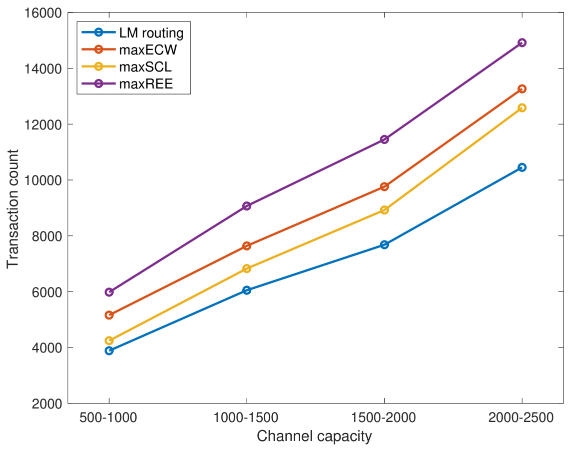
Figure 5.6: *Transaction count in worst case when transaction set is same in all iterations*

take a longer route. Our newly proposed algorithm $maxREE$ has less average path length than $maxECW$ and $maxSCL$, as it only acts on the edge where change is required instead of making changes during all iterations. This improves the transaction count, reduces the average path length, and reduces unnecessary overhead. The improvement of our recently suggested method and previously presented techniques with regard to LM routing is seen in Table 5.2. The negative value of average path length represents the overhead of an increase in average path length in the proposed algorithms compared to LM routing.

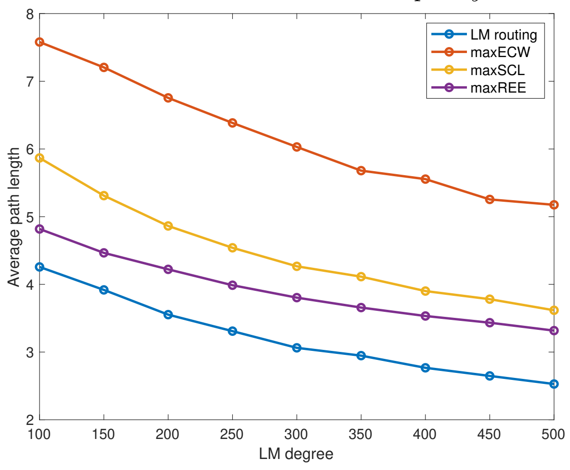
Worst case scenario: The existing landmark routing algorithm is static and does not adapt to changes. To assess how adaptable the suggested routing algorithm is, we took the worst-case scenario into account. In the worst scenario, the rest of the simulation parameters will remain the same, and the transaction set will be the same over all iterations. The transaction count across iterations in the worst-case scenario is depicted in Figure 5.6. The landmark routing technique failed after a few iterations because its MSL channels ran out of capacity, whereas our suggested approach never ran out of the MSL channel's capacity. $maxREE$ considers both link capacity and balance on the channel, but in the worst-case scenario, rebalancing plays a vital role as the same set of links is used. Thus, initial iterations $maxECW$ and $maxREE$ perform better than $maxSCL$. However, toward the end, $maxECW$ performance keeps decreasing, but $maxSCL$ performance remains constant throughout the iterations, and $maxREE$ lies between them.



(a) *Transaction count over δt with change in degree*



(b) *Transaction count over δt with change in channel capacity*



(c) *Average path length over δt with change in degree*

Figure 5.7: *Transaction count and path length with change in parameters*

Change in network parameters: Figure 5.7a shows the transaction count over δt with a change in degree; all other input metrics are the same during the iterations. This simulation is conducted for 2000 nodes, with an average degree, number of iterations, tx_number, and channel capacity as 7, 20, 1000, and 500-1000, respectively; the LM degree is varied from 100 to 500. With the increase in LM degree, the overall flow of transactions will increase, but our proposed algorithm performed better than all existing algorithms. This means that choosing effective channels is necessary to prevent congestion. As a result, *maxECW* consistently outperforms the other algorithms since it always picks the best channels available in the network. Figure 5.7c shows the average path length over δt with a change in LM degree. The average path length of routing trees decreases with an increase in LM degree,

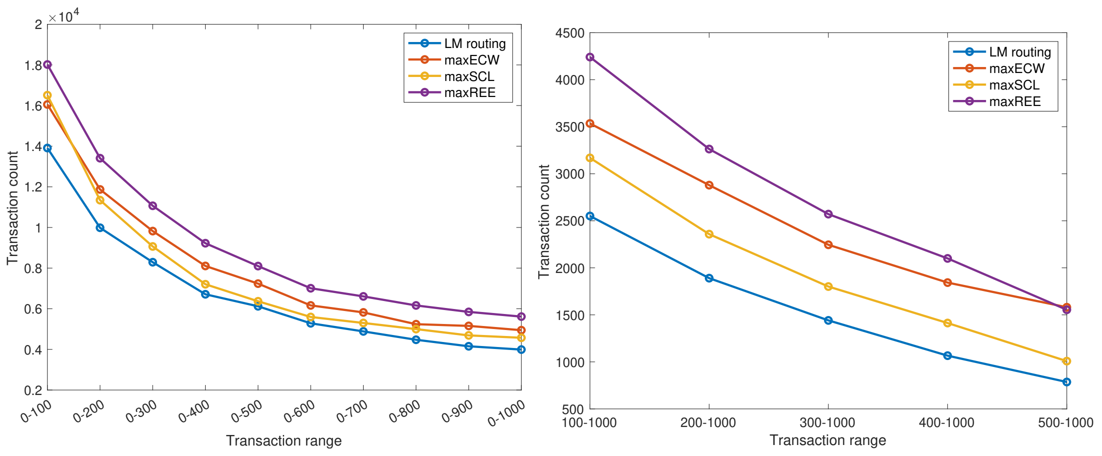
Algo \ $(\lvert V\rvert, \lvert E\rvert)$	$(2000, 7096)$	$(3000, 10646)$	$(5000, 17745)$
maxECW *(TXC)*	31.80%	38.01%	46.28%
maxSCL *(TXC)*	8.15%	18.24%	24.85%
maxREE *(TXC)*	**53.4%**	**60.73%**	**59.22%**
maxECW *(TXV)*	48.51%	54.11%	60.73%
maxSCL *(TXV)*	13.62%	24.55%	31.91%
maxREE *(TXV)*	**64.43%**	**70.77%**	**70.05%**
maxECW *(APL)*	−3.36	−3.3627	−3.44
maxSCL *(APL)*	−1.33	−1.3068	−1.32
maxREE *(APL)*	**−0.71**	**−0.65**	**−0.58**

Table 5.2: *A comprehensive comparison of methods on the custom network over δt*

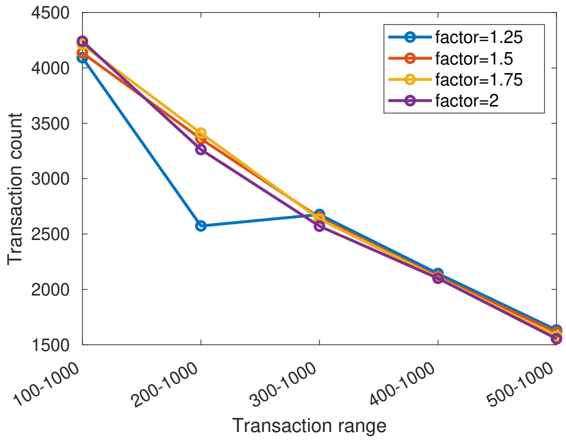
as the depth of the routing tree decreases. However, our proposed algorithm *maxREE* has less average path length than the previously proposed algorithms as it intelligently replaces the edges in the routing tree on each iteration.

The transaction counts over δt with a change in channel capacities is shown in Figure 5.7b. With the increase in channel capacities, the execution of concurrent transactions will increase as an edge has more funds to proceed with multiple transactions at a time. The ability of the routing algorithm to modify the routing tree in accordance with transaction flow will determine the transaction count over δt. It is clear from the plot that our proposed algorithm is more adaptive than existing algorithms.

Change in transaction parameters and SRF: The output of our proposed algorithm depends upon the transaction size, so we varied the transaction size by fixing the channel capacity to 500-1000 and *SRF*. First, the transaction's upper limit is raised till the upper limit of the transaction, as shown in Figure 5.8a. Then, the transaction's lower limit is also raised till the size of channel capacity, as shown in Figure 5.8b. In both cases, the transaction count decreases as the size of the transactions increases due to a decrease in feasible transactions or transaction flow. Due to its adaptive nature in selecting precise edges throughout the development of the routing tree, our suggested method outperformed other approaches. Figure 5.8 describes the performance of *maxREE* with change in *SRF* and transaction size. When transaction size is 100-1000, the algorithm performs best with *SRF*=2 and worst with *SRF*=1.25, but as the transaction size increases, the scenario is reversed, and at the end when transaction size is equal to channel capacity, output is almost identical irrespective of *SRF* value. We can also observe deflection for *SRF*=1.25 for transaction range 200-1000, as *SRF* value is very low, so edges will get exhausted when 80% of their capacities are used, thus transaction flow will be reduced over the iterations due to lack of funds on channels. Here, we have changed the *SRF* from 1.25 to 2 to demonstrate that,



(a) *factor*= 2 **(b)** *factor*= 2

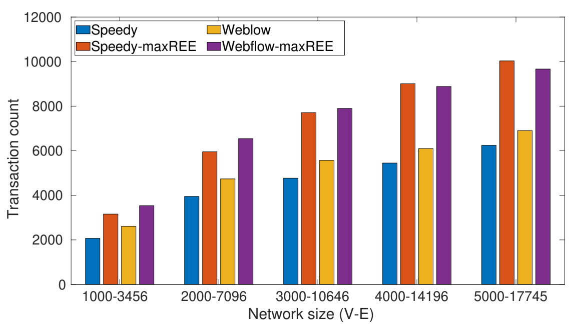


(c) *SRF*= [1.25 : 2]

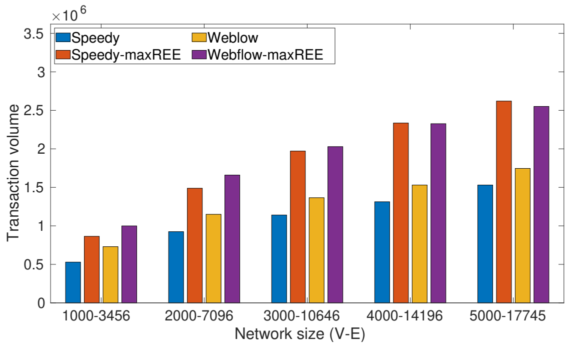
Figure 5.8: *Evaluation of proposed algorithm over the change in transaction range and SRF*

in cases where transaction size and channel capacity are similar, the choice of *SRF* value is inconsequential because, during each iteration, the used channel may run out and be replaced by other active channels. It can be referred to as a toggling effect. Researchers often advise doing small transactions over LN and significant ones through the Bitcoin network. Therefore, setting the *SRF* value to 2 is a practical idea.

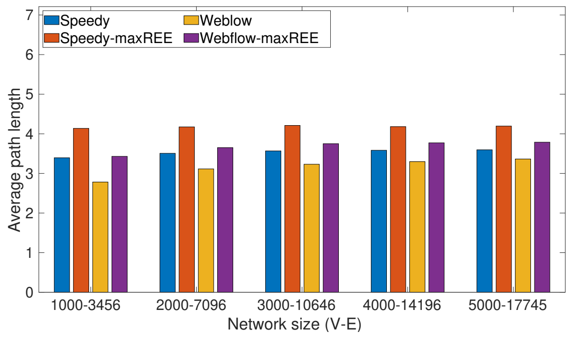
Simulations and theoretical analysis: As seen in Section 5.3, the transaction volume decreases over the iteration in the landmark algorithm due to an increase in congested links. A similar phenomenon is observed in Figure 5.5d, however, the decrease is not linear but eventually decreases as some links regain their capacities due to the random flow of transactions. But our proposed algorithm According to equation Equation 5.8, transaction count is directly proportional to landmark degree and channel capacity, the same results we can observe from Figure 5.7a and 5.7b respectively. Equation 5.10 states that the path



(a) *Transaction count over the change in Network size*



(b) *Transaction volume over the change in Network size*



(c) *Average path length over the change in Network size*

Figure 5.9: *Comparison Speedy and Webflow algorithms with maxREE*

length of optimized transactions through *maxREE* will have an increase in path length in-between 1 to 2, according to Table 5.2 increase in average path length is less than 1. This is because the average path length is calculated with reference to all feasible transactions, not only the optimized transactions; thus, its value is less than 1. This simulation result supports our theoretical result.

Comparison with Speedy and Webflow The analysis above demonstrates that *maxREE* performs better than LM routing. To construct the spanning trees, Speedy [82] and Webflow [111] take advantage of the LM routing. To assess the efficiency of our proposed routing tree algorithm, we compared the results of Speedy and Webflow after substituting the *maxREE* routing algorithm for the default one. We compared results for transaction count, transaction volume and average path length with different network sizes, as shown in Figure 5.9. Figure 5.9a shows that transaction count is improved by exploiting the *maxREE* in Webflow as well as speedy. For transaction volume, Figure 5.9b describes similar results. Figure 5.9c illustrates that the average path length difference using the default and *maxREE* algorithms is less than one in both Speedy and Webflow. As we've just covered, *maxREE* performs better than LM routing, and enhancing Speedy and Webflow's base enhances their effectiveness even more.

In Figure 5.9a and 5.9b, we can observe that the improvement in transaction count and volume with the *maxREE* algorithm is more in Speedy as compared to Webflow, and it increases with an increase in the network size. This is because transactions in the Speedy algorithm can be routed through an LM node or the shortest path in the sub-tree if the receiver belongs to the same sub-tree. As the network size increases, the probability of a receiver being in the same sub-tree will decrease as the LM degree increases; thus, maximum transactions will be routed through the LM node. LM node keeps updating the routing tree, so the path provided by the LM node is congestion-free, which improves the number of successful transactions but may increase the path lengths. We can see in Figure 5.9c that the average path length increase in Speedy is more than Webflow, as the maximum transactions are routed through the LM node, which means the number of updates in the spanning tree is more, increasing the average path length.

Meanwhile, in Webflow, transactions can be routed via a shortcut path even if the receiver belongs to a different sub-tree. So, the chances of routing the transaction through the LM node are lower than that in the Speedy algorithm. The *maxREE* algorithm is more reliable in providing the transaction route than standard LM routing, but the number of transactions routed through the LM node is less, and the non-LM transactions are routed through shortcuts. The non-LM routes are not reliable as the balance information of such routes is not available. Thus, the Webflow's improvement is less than that of the Speedy algorithm. The increase in average path length is also smaller, as the number of transactions opting for LM routes is smaller.

Summary: In the *maxECW* algorithm, the LM node needs to update all channel capacities with time; a suitable gossip protocol is required. Moreover, we need to re-calculate the routing tree over all the iterations. In the *maxSCL* protocol, no channel weights are necessary for calculating the routing tree, and there is no need to re-calculate the routing tree on each iteration. Still, it only rebalances the channel without prior knowledge of channel capacities. *maxECW* provides the transaction concurrency with more overhead in average path length, whereas *maxSCL* provides rebalancing. *maxREE* algorithm provides both features of optimal edge selection and rebalancing with the least overhead. LM routing is static and uses the same routing tree over iterations, thus reducing its efficiency. Our proposed algorithm *maxREE* improves the transaction count by more than 50%, in all network topologies, with less than one hop increase in average path length.

5.5 Discussion

Routing algorithm

The average path length is optimized using the silent whisper algorithm by making use of landmark routing. Moreover, Webflow and Speedy murmurs decrease the average path length by using coordinate assignment shortcuts. However, as landmark routing is the foundation of all of these methods, they try to improve the existing landmark routing by adding shortcuts between sub-trees of the spanning tree generated by the LM node, using the assigned coordinates. Speedy can only exploit shortcuts within a sub-tree, whereas Webflow can exploit inter-sub-tree shortcuts. Our suggested technique aims to construct a dynamic routing tree to enhance concurrent transactions. Because of a continuous flow of transactions, channel balance and its dynamic values are unavailable, which presents a challenge when building a routing tree. Thus, to determine the channel state without compromising privacy, we used *SRF* in place of accurate channel balances. Replacement of the channel is done based on *SRF*. Our suggested algorithm will automatically replace exhausted channels, assisting in self-balancing. Our suggested algorithm outperformed landmark routing because it replaces drained links periodically, preventing congestion and maximizing transaction flow. In contrast, landmark routing uses the same routing tree, which uses congested links and decreases flow. While off-chain and on-chain techniques can be used by the landmark routing node to improve congested links and come with accompanying costs, our suggested approach does not require them because of self-rebalancing.

Simulator and Network Dynamics

Our proposed simulator, DRLNsim, is adaptive with network topologies, and all the input parameters are referred from the actual Lightning Network analysis [53, 64, 85]. A Lightning Network is preferred for transactions with small transaction amounts, transactions with higher amounts are preferred from the main chain(Bitcoin). But channel capacities and transaction amounts are completely decided by users on the network. So we vary channel capacities from values comparable to transaction ranges to higher values to check the credibility of our proposed algorithm. LN network is a scale-free network [85]; new nodes joining the network tend to open a payment channel with LM nodes. In the future, LM degrees may increase, so we compared our results by varying LM degrees to higher values. Our proposed algorithm makes changes to the routing tree, with a change in link weight to retain the transaction concurrency. We may observe that our proposed algorithm works better than all current methods with changes in network parameters. It has also been observed that when network size is raised, Speedy performs better than Webflow. Compared to Webflow, there are more transactions querying the LM node for routes. This further demonstrates that, in comparison to shortcuts employed by existing approaches, the routes offered by the LM node utilizing our suggested dynamic algorithm are more feasible.

Our simulations are conducted assuming that there is no network failure. However, our proposed algorithm adapts to changes in network topology; in case of node failure, it recomputes the routing trees. If a new payment channel is added to the PCN network, the landmark node will update its graph as the channel opening transaction is added to the main chain, which is visible to all full nodes in the network. So, simultaneously, it will update the routing trees. Likewise, channel-closing transactions assist in updating routing trees in the event that a channel is closed. Additionally, the average channel lifetime is 212 days [107], meaning that this update is not particularly often. Because PCN network transactions are atomic in nature, they will be terminated if there is any channel or node fluctuation while the transaction is being carried out from source to destination. This is because HTLC on nodes expires after a certain amount of time. In the future, we will add more functionalities to our simulator to mirror real-world network dynamics.

5.6 Conclusions

We have proposed *maxREE* an algorithm to maximize the number of concurrent transactions in PCNs. In contrast to the existing algorithms, which cannot handle concurrent transactions while maintaining the transaction fee, our algorithm is adaptive and takes into account the dynamic nature of the channel capacities. Only the required edges are pruned from the routing tree, so the overhead of tree construction is reduced. The theoretical analysis supports our experimental findings by demonstrating that the transaction flow remains stable under our suggested method but will decrease in earlier landmark routing due to the congestion of channels. The experimental results demonstrate that the suggested approach enhances concurrency by 50% above the state of the art. Thus far, this thesis has introduced techniques to enhance concurrency, fee minimization, and channel rebalancing; in a subsequent chapter, we will examine how scalability induces deadlock in PCN and the interrelationship among deadlock, channel balance, and scalability.

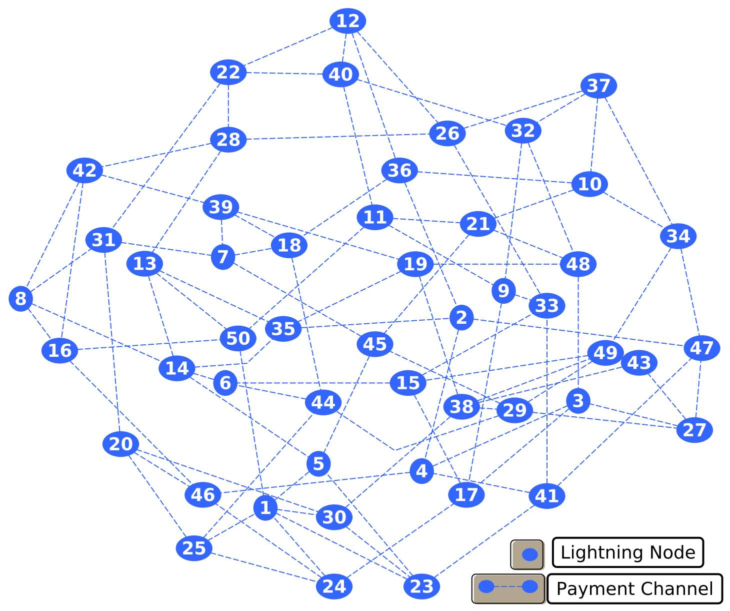
6
Deadlock Prevention in PCNs

In the preceding chapters, we focused on routing methods to enhance concurrency and reduce transaction fees. However, routing transactions through numerous choices will impact their feasibility. However, while the scalability of distributed routing algorithms is enhanced compared to static landmark routing, these algorithms are more afflicted by the problem of deadlock. This chapter aims to examine the presence of deadlocks in current distributed algorithms and propose potential solutions for addressing this issue. In this chapter, we study deadlocks in spanning tree algorithms that take use of resource allocation graphs and how the dynamic behavior of resources makes the situation worse. We define the deadlock trilemma and demonstrate that the D2TIR issue is NP-complete. We provide a method for improving upon current routing algorithms that is called DEadlock PRevention (DEPR) for avoiding deadlocks. Our results show that the *DEPR* method outperforms the current LN routing tree methodology. The *DRLNsim* simulator is used to carry out the simulations.

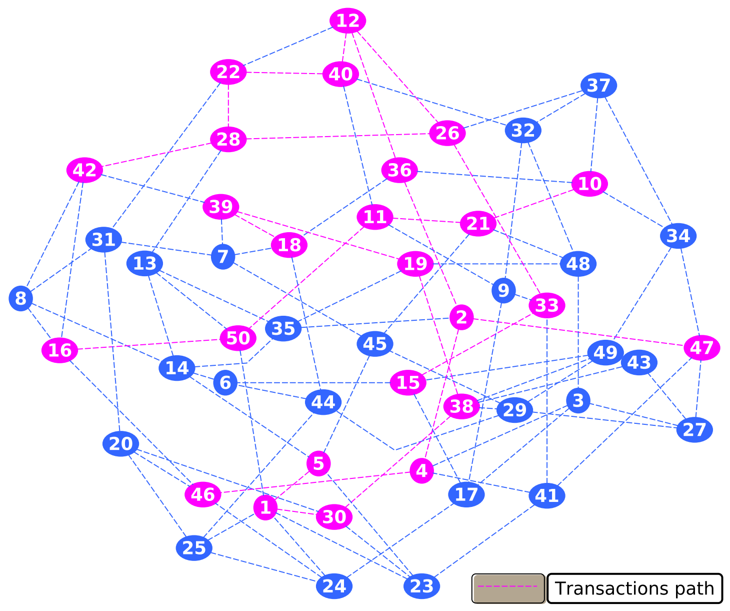
The rest of the chapter is organized as follows. The illustration of deadlocks in routing tree algorithms is explained in Section 6.1. theoretical insights are described in Section 6.2, The proposed approach is described in Section 6.3. Section 6.4 presents the simulation setup; and the performance evaluation of the proposed approach, deadlock dependencies are described in Section 6.5. We finally conclude in Section 6.6.

6.1 Deadlocks in Distributed Routing

When concurrent payments are being processed and have shared edges in their paths, such that no payment is successful, this is known as a *deadlock* in the PCN [103]. Since there is a



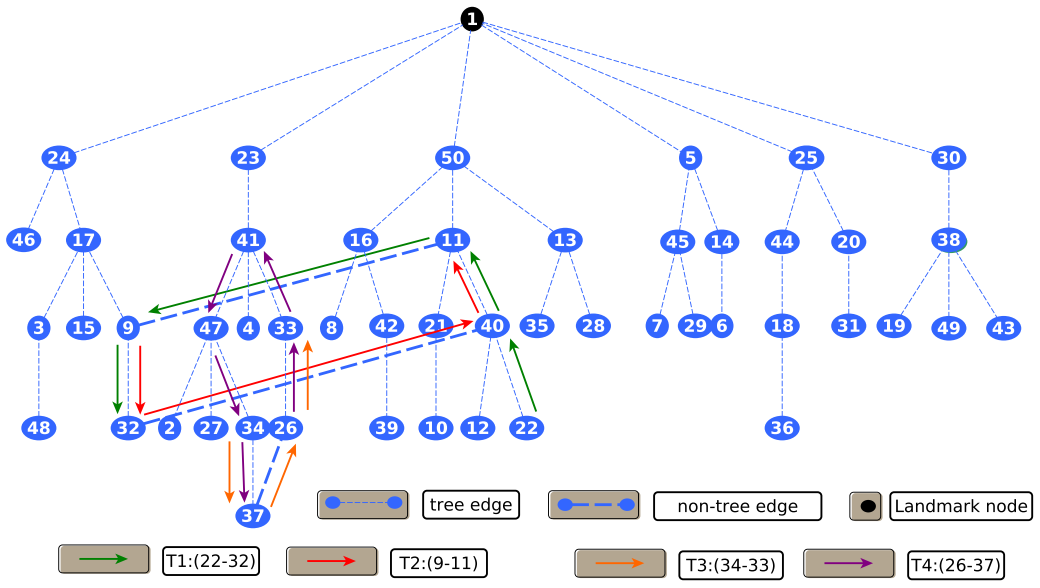
(a) *Lighting graph with 50 nodes*



(b) *Embedded routing with 10 Txs*

Figure 6.1: *Deadlock situations in routing tree*

channel capacity restriction in PCNs, it is possible that two payment routes with the same edge will not be able to travel through at the same time. In addition, pathways are made up of several edges obtained sequentially (using the HTLC configuration). Once an edge



(c) *Deadlock with two Txs*

Figure 6.1: *Deadlock situations in routing tree*

has an established HTLC, a payment can block it and wait for an edge that is being held up by another transfer. This is possible only if there is a cycle in the paths of the transactions in deadlock.

Deadlock Illustration in a Routing Tree Consider the random graph shown in Figure 6.1a generated with 50 nodes. The graph in Figure 6.1c is a landmark routing tree formed by applying BFS on graph 6.1a. As we know, Webflow [82] uses virtual coordinates to route a transaction. Consider two transactions $T_1 : (s_1 = 22, d_1 = 32)$ and $T_2 : (s_2 = 9, d_2 = 11)$; where s_i and d_i represents sender and receiver, respectively. Each payment is of 1 unit, and each edge has a capacity of 1. The edge's capacity in the opposite direction is 0. The order of execution is $([22, 40]_1, [9, 32]_2, [40, 11]_1, [32, 40]_2, [11, 9]_1, [40, 11]_2, [9, 32]_1)$, where each pair represents an edge and the subscript represent transaction number. The green and red edges in Figure 6.1c are the routes followed by transactions T_1 and T_2, respectively. In this situation, T_1 is unable to acquire edge $[9, 32]$ and T_2 is unable to acquire $[40, 11]$, thus they are deadlocked. Similarly, transactions T_3 and T_4 will be deadlocked with execution sequence $([27, 37]_3, [26, 33]_4, [37, 26]_3, [33, 41]_4, [41, 47]_4, [26, 33]_3, [47, 27]_4, [27, 37]_4)$.

The next question arises: what is the possibility of a deadlock situation? In the above illustration, edge capacities are either zero or one; it represents that edges are unbalanced, which means they can forward transactions in only one direction. Silent Whispers and Speedy Murmurs take a greedy approach to get the shortest path but do not follow any procedure to rebalance the channels. When concurrent transactions are routed at the same time, there will be a higher probability of a cycle. In the above example, when 10 trans-

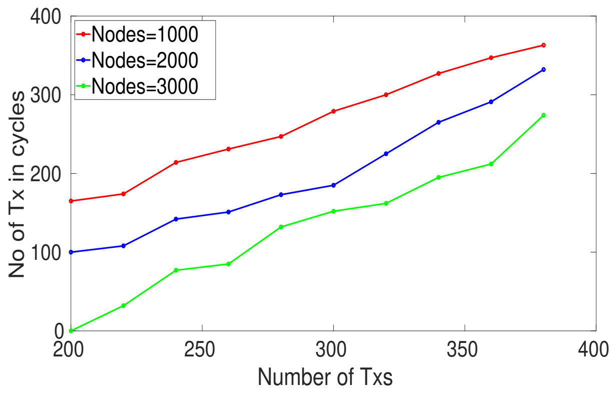


Figure 6.2: *Txs involved in a cycle with respect to number of Txs*

actions are routed using embedded routing, there exist three cycles, as shown in Figure 6.1b.

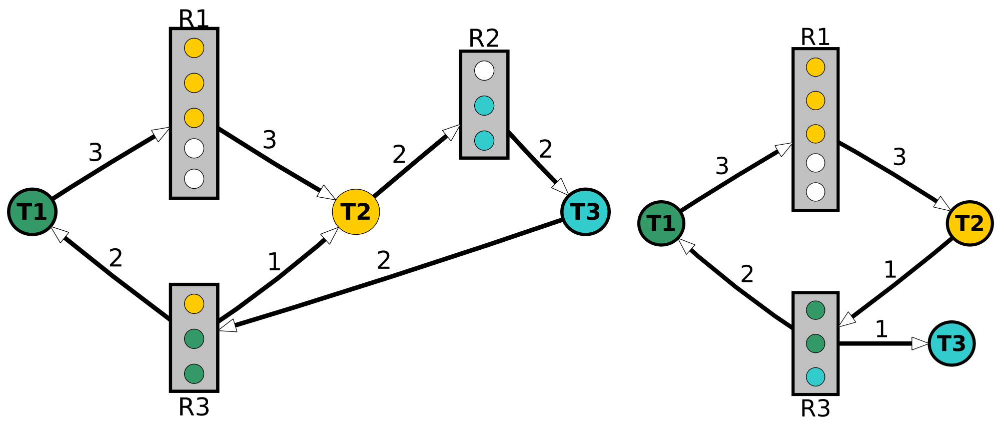
For a more general scenario, we find transactions involved in cycles with the increase in the number of transactions with different graphs, as shown in Figure 6.2. It can be observed from the chart that with the increase in the number of transactions, the possible number of cycles also increases. Although the existence of a cycle does not necessarily result in a deadlock, the possibility of having a deadlock depends upon cycles, so the number of cycles is the possibility of deadlock. Moreover, assigning routes to a group of transactions without deadlock is NP-complete [103].

Is it good to ignore deadlocks? Consider the same example illustrated in Figure 6.1a. If we ignore the transactions involved in deadlock and assume no deadlock has occurred, then both transactions will fail and again try for the path according to embedded routing. The deadlock will occur again if the same path is allocated to them. Alternatively, the longest path, that is, the path through the landmark node, will be assigned to them. Figure 6.2 shows a situation where many transactions are involved in a deadlock situation, which will delay transaction completion as path selection will take time. Probing for paths can take more than 5% of the attempted payments [74].

6.2 Network Model

Definition 6.2.1 (Resource Allocation Graph)**.** Let $G(V, E, W)$ be a PCN graph and set of transactions, T, with feasible solution P. We construct resource allocation graph $G^{'}(V^{'}, E^{'}, W^{'})$ where,

(a) $V' = T \cup R$ with $T = \{T_1, T_2, \ldots, T_n\}$ and $R = \{R_1, R_2, \ldots, R_m\}$ representing transactions and resources, respectively. Here a resource R_j refers to an edge $(u, v) \in P$, $P = \bigcup_{i=1}^{n} p_i$. Each resource instance of R_j is an indivisible value of $W(u, v)$.



(a) *Case 1* **(b)** *Case 2*

Figure 6.3: *Resource allocation graph with cycles*

(b) $E^{'}$ is a subset of $((T \times R) \cup (R \times T))$, and each element (x, y) represents a directed edge from x to y, where $x, y \in V^{'}$. Each transaction $T_i \in T$ needs some instances of resource R_j for completion. If a resource R_j is acquired by a transaction T_i, then it has an inward edge from resource R_j. If T_i requires some instances of resource R_j but is unavailable, then T_i has an outward direction toward resource R_j.

(c) $W^{'} : E' \rightarrow \mathbf{Z}$, represents weight on edge (x, y), $W^{'}(x, y)$ is the number of instances acquired by transaction y of resource x or number of instances required of resource y by transaction x.

Lemma 6.2.1. *A deadlock in PCN can occur under some ordering of T, if and only if resource allocation graph contains a simple cycle and for each transaction x in the cycle, it is waiting for resource y such that $W^{'}(x, y)$ on edge (x, y) is less than available instances in y.*

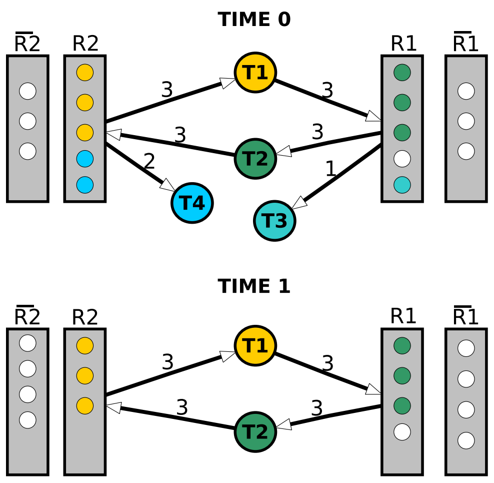
Proof. If there is a deadlock in the graph for a certain ordering of transaction, then there is a transaction T_1 which is holding resource R_1 and waiting for resource R_n, T_2 is holding up R_2 and waiting for R_1, etc. So the resource allocation graph contains a cycle $(T_1, R_1, T_2, R_2, \ldots, T_n, R_n, T_1)$.

Let S be set to represent the summation of all instances of resource R_j and A be set to represent the number of allocated instances of resource R_j. For each edge (T_i, R_{i-1}) in cycle

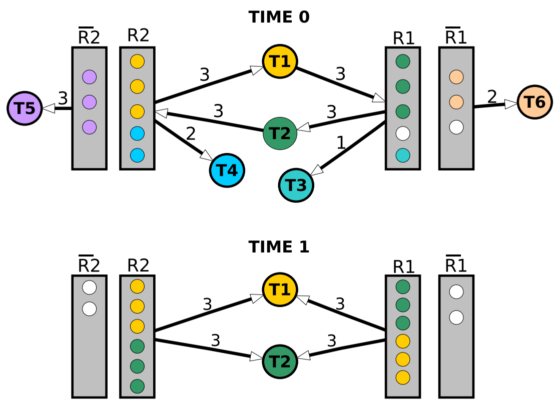
$$W^{'}(T_i, R_{i-1}) < S_{R_{i-1}} - A_{R_{i-1}} \tag{6.1}$$

So under this ordering, there is a deadlock between transactions $T_1, T_2, \ldots, T_n$ in G.

(a) Figure 6.3a shows the resource allocation graph with three transactions and three resources. In this CASE, transaction T_1 is waiting for resource R_1, which is assigned



(a) *Case for Lemma 6.2.2*



(b) *Case for Lemma 6.2.3*

Figure 6.4: *Dynamic resource allocation graph*

to transaction T_2, transaction T_2 is waiting for resource R_2, which is assigned to T_3, transaction T_3 is waiting for resource R_3 which is assigned to transactions T_1, T_2; and thus the cycle is there satisfying the Equation 6.1, which confirms the deadlock.

(b) Figure 6.3b has cycle between transactions T_1 and T_2, but deadlock depends upon the execution of transaction T_3. In a standard deadlock situation, no deadlock will occur as transaction T_3 after completion will free resource R_3, and the cycle will be broken. But resource here refers to edge (u, v), when an instance of the resource is consumed by a transaction T_3, it is no longer available on edge (u, v), it is shifted to resource (v, u). Further, if transaction T_3 has failed due to some reason, then the resource is not consumed by it; thus, the cycle is broken, so there is no deadlock. But we assume the system to be fault-tolerant, and no transaction will fail if it has all the required resources.

□

6.2.1 Dynamic Behavior of Resources in PCN

Here we will discuss the dynamism in the resource allocation graph, which will affect the deadlock state. Consider the resources R_i and $\overline{R_i}$ representing the edges (u, v) and (v, u), respectively. If a transaction uses resource R_i, then instances from R_i will move to $\overline{R_i}$ and vice-versa. Let T_d be a set of transactions involved in the deadlock, $T_r = T - T_d$. R_d is the set of resources involved in the deadlock. In the following discussion, we consider a timeline starting with time 0 where at each time instant we have a set of transactions to route and execute.

Lemma 6.2.2. *In a given PCN graph G and transaction set T, if there exists a cycle at time* 0 *in resource allocation graph $G^{'}$, then the deadlock will continue at time 1 if T_r in $G^{'}$ only utilizes R_d.*

Proof. If transactions in T_r only utilize the resource R_d, then instances from R_d will shift to $\overline{R_d}$, but no instance from $\overline{R_d}$ will shift to R_d. Thus

$$S_{R_j(Time0)} \leq S_{R_j(Time1)} \qquad (R_j \in R_d)$$

$$A_{R_j} > 0 \qquad (R_j \in R_d)$$

and from the equation 6.1, we have

$$W_0'(T_i, R_j) \leq W_1'(T_i, R_j) \qquad (T_i \in T_d)$$

Therefore, the deadlock state will continue at Time 1.

In Figure 6.4a, transactions T_3, $T_4 \in T_r$ and transaction T_1, $T_2 \in T_d$. At Time 0, T_3 and T_4 acquired instances of resource R_1 and R_2 respectively, after completion instances used by transaction T_3 and T_4 will be shifted to resource $\overline{R_1}$, $\overline{R_2}$ respectively. But transactions T_1, T_2 continue deadlock state in Time 1, as insufficient instances are available. □

Lemma 6.2.3. *Given a PCN graph G and transaction set T, if at time* 0 *there exists a cycle in resource allocation graph $G^{'}$, then deadlock will continue at time 1 if and only if*

$$\forall R_j \in R_d(Time1)$$
$$S_{R_j} < \sum (W'(T_i, R_j) + W'(R_j, T_i)) \quad (T_i \in T_d)$$

Proof. Transactions in T_r can utilize resources from R_d as well as $\overline{R_d}$. Thus,

$$S_{R_j(Time1)} = S_{R_j(Time0)} - \sum_{T_k} A_{R_j} + \sum_{T_k} A_{\overline{R_j}}$$

$$(T_k \in T_r), (R_j \in R_d)$$

If resource R_j is more utilized by the transactions $T_k \in T_r$, then R_j will have fewer instances at Time 1 compared to Time 0. Thus, R_j will not be able to satisfy the requirement of transaction in T_d and will satisfy the Eqn 6.2. Hence deadlock will continue at Time 1.

In Figure 6.4b transactions T_3, T_4, T_5, $T_6 \in T_r$ and transactions T_1, $T_2 \in T_d$. After the resource shift at Time 1, $R_j \in R_d$ opposes the Eqn 6.2. Thus deadlock is broken, and all transactions in deadlock (T_1, T_2) get their required resources and are completed. □

6.2.2 The Two-transaction Routing in PCN is NP-complete

In [103] authors have shown that the safe routing problem in PCN is NP-complete. The safe routing problem is the problem of determining whether there are individually viable routes for a set of transactions that are safe from deadlock. Our aim is to prove that routing two transactions in PCN is NP-complete.

In [26] authors had shown that two-commodity Integral Flow in Directed graphs (D2CIF) is NP-complete. The authors reduced the satisfiability problem to D2CIF, by constructing the lobe for each clause using literals. We present a reduction of the (D2CIF) to Two-Transaction Integral routing in Directed graphs (D2TIR).

Definition 6.2.2. Given a graph $G(V, E, W)$, $(u_1, v_1), c_1$ and $(u_2, v_2), c_2$ are two transactions T_1 and T_2 respectively. The problem is to determine whether there exist two paths P_1 and P_2 for the completion of T_1 and T_2, such that,

$$P_i = (W_{(u_i,1)}, W_{(1,2)}, \ldots, W_{(l,v_i)}) \quad \{\ 1, 2, \ldots, l\} \in V,\ i = \{1, 2\}$$

$$c_i \leq min(W_{(u_i,1)}, W_{(1,2)}, \ldots, W_{(l,v_i)})$$

$$\sum T_i(s,t) - \sum T_i(t,s) \leq W_{s,t} + W_{t,s} \quad \{s,t\} \in V - \{u_i, v_i\}$$

$$T_i(u_i, s) - T_i(s, u_i) = c_i$$

$$T_i(s, v_i) - T_i(v_i, s) = c_i$$

In D2CIF problem $G''(V'', E'')$ is to determine whether there exist two functions f_1 and f_2, such that two commodities will reach from source (s_1, s_2) to destination (t_1, t_2), $(s, t \in V'')$, by utilizing the weight on the directed edges E''. In D2TIR, transactions can be considered commodities, and edge weight is referred to as channel balance. In D2CIF, if edge weight is utilized by one commodity, it can't be used by other commodities. D2TIR, HTLC locks the channel funds on the channel once utilized by any transaction. Thus, D2TIR is a similar problem to D2CIF; thus, D2TIR is NP-complete.

6.3 Deadlock Free Distributed Routing

In distributed PCN, keeping all of the channels balance is a challenge. Landmark nodes know the whole topology of LN. Channel capacities are also known to landmark nodes, as these values are included in transactions used to open the channel and are recorded on the blockchain. In the event of Webflow and a Speedy algorithm, landmark nodes broadcast coordinates and offer pathways to receive requests. However, the request does not share the desired transmission amount. So the channel funds/weights keep on updating as transactions use those channels, these updated funds are known as channel balance. The nodes with higher degrees, also referred to as the most influential nodes in the network, make up the landmark node-set that is employed in distributed routing. Since the set of landmark nodes is dynamic, the system is expected to remain decentralized. This is because the new nodes take the place of many influential nodes that had previously lost the opportunity to be a landmark node [108]. As, the landmark nodes are those that have a greater network degree and, as a result, they are the source of the majority of transactions. Thus, the transactions that are occurring through landmark nodes are really helpful. Landmark has the ability to maintain track of the transaction amount and continuously update the Transaction threshold.

Definition 6.3.1 (Transaction threshold (λ))**.** The average value of the transaction amount passing through landmark nodes is known as the transaction threshold. The first transaction amount is the initial value, and it will update subsequently.

As demonstrated in section 6.1, there is a higher probability of deadlock when the transaction size equals the channel balance. This λ value will be utilized by our suggested method DEPR to maintain channel balance and avoid deadlock.

6.3.1 DEadlock PRevention (DEPR)

The algorithm workflow is described in detail in Figure 6.6. The Landmark node initially sets the value of λ to zero, and the spanning trees are constructed applying BFS on PCN graph G. Upon receiving multi-hop transactions from its neighbors, the landmark node will propagate them and update λ. A non-landmark node sends a path request to the landmark node in order to complete the transaction. In response to that request, the Landmark node sends the path and λ value after determining the path using spanning trees. In parallel, it updates the channel balance, CB, of the routing trees' edges. If an edge drops below λ, it uses the revised graph to reconstruct the spanning trees. The initial CB of edge (u, v) is equal to $W(u, v)$; the landmark node updates the CB of routing trees and weight on the graph G depending on λ; each time a path request is made. All these steps are described in Algorithm 3.1.

The steps taken by the non-landmark node to route the transaction after receiving the path and λ from the landmark node are given in Algorithm 3.3. First, the coordinates of the point s are compared to determine if r is reachable through the shortcut. The transaction will only be routed through the shortcut if the balance of the path is greater than the λ. Alternatively, the sender routes the transaction through a path shared by the landmark node.

After the selection of the path, node s will commence the process of onion routing [40] by transmitting the onion message, which includes both the chosen path and the payment that needs to be sent. Upon receipt of the onion message, each intermediate node will sequentially unwrap the layers of encryption to unveil the subsequent destination point of the message. Subsequently, the node will establish the HTLC with the next node in the

Algorithm 6.1 Tree construction

```
 1: $Q \leftarrow \emptyset$
 2: $\lambda \leftarrow 0$
 3: $t_1, t_2 \leftarrow \emptyset$                                  ▷ spanning trees constructed at $l$
 4: procedure SpanningTree($G$, $l$ )                                 ▷ $l \in LM$
 5:     if $\lambda = 0$ then
 6:         $t_1 \leftarrow t_2 \leftarrow BFS(G, l)$
 7:         $Coordinate\ Assignment(t_1, t_2)$
 8:     else
 9:         $Q.enqueue(l)$
10:         while $Q \neq \emptyset$ do
11:             $u \leftarrow Q.dequeue()$
12:             for each $v \in InDegree(G, w)$ do
13:                 if $v \notin t_1$ & $CB(u, v) > \lambda$ then
14:                     add edge $(u, v)$ to tree $t_1$
15:                     $Q.enqueue(v)$
16:         $Q.enqueue(l)$
17:         while $Q \neq \emptyset$ do
18:             $u \leftarrow Q.dequeue()$
19:             for each $v \in OutDegree(G, w)$ do
20:                 if $v \notin t_2$ & $CB(u, v) > \lambda$ then
21:                     add edge $(u, v)$ to tree $t_2$
22:                     $Q.enqueue(v)$
23:         $Coordinate\ Assignment(t_1, t_2)$
24:     return $t_1, t_2$ to $l$
 1: procedure PathRequest($l$, tx(s,d))
 2:     path $\leftarrow$ $findPath(G, t_1, t_2, s, d)$
 3:     for each $(u, v)$ in path do
 4:         CB($u$,$v$) $\leftarrow$ CB($u$,$v$) - $\lambda$              ▷ update edges of $t_1, t_2, G$
 5:         if CB($u$,$v$)< $\lambda$ then
 6:             Chk $\leftarrow$ 1
 7:     return (path, $\lambda$) to $s$
 8:     if Chk = 1 then
 9:         $SpanningTree(G, l)$
10:     else
11:         return ($G$, $t_1, t_2$) to $l$
```

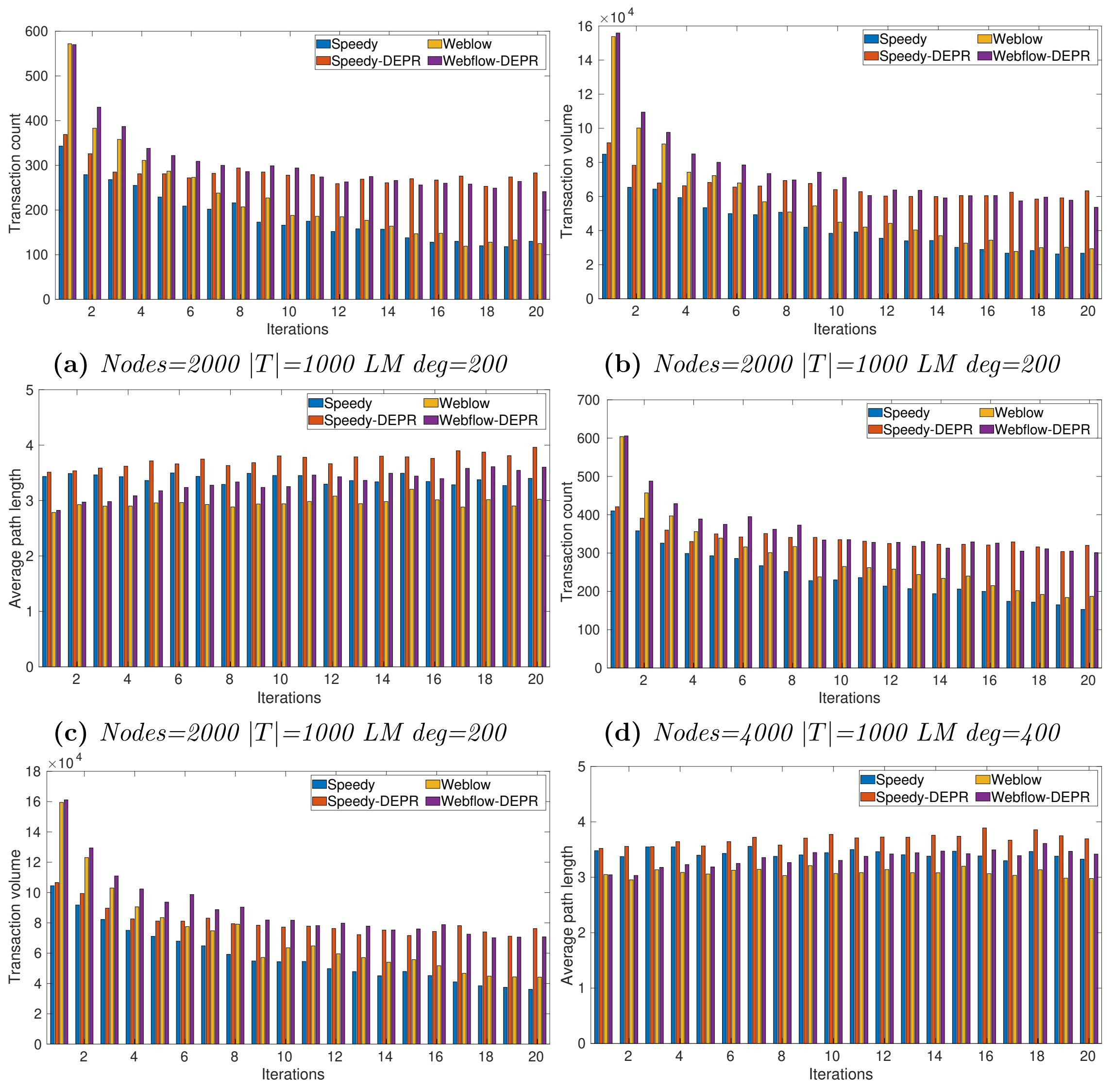
routing path and proceed to transmit the message. The aforementioned procedure will iterate until the destination d successfully receives the transaction.

Algorithm 6.2 Transaction routing

```
1: procedure RouteTransaction(s,d)
2:     if $d.coordinate$ $reacheable$ then                    ▷ Speedy/Webflow
3:         $path \leftarrow$ ComputePath($s, d$)
4:         if pathBalance> $\lambda$ then
5:             SendOnion($path$, $amount$)
6:     else
7:         $lpath \leftarrow$ LMPath($l, s, d$)
8:         SendOnion($path$, $amount$)
1: procedure ReceiveOnion($path$, $amount$)
2:     OnionPeel()
3:     $next \in$ Neighbours($i$)
4:     Build HTLC with the next node in path with $amount$
5:     SendOnion($path$ $-\{i\}$, $amount$)
```

Our DEPR algorithm exploits the transactions flowing through the landmark node to improve the routing tree construction as well as to improve the decision of the sender to optimally choose the route. Routing trees are reconstructed if the channel balance drops its value below the λ. Such updation with the progress of time avoids saturated links. So, the path provided by the landmark node will decrease the probability of transactions being in a deadlock situation. Existing distributed routing protocols, Webflow and Speedy, use static routing trees constructed with BFS on all iterations. This static routing tree is not able to bear the concurrent transaction and will saturate its links over time. Our proposed algorithm uses two different trees for sending and receiving the transactions; there is no compulsion to use the same channel for sending and receiving the transaction. When the CB of a few channels falls below λ in t_1, this means the edge in the opposite direction will have sufficient funds and is highly probable to be in t_2. So, the routing trees will complement each other to maintain sufficient CB on channels depending upon the flow of transactions.

We have discussed in Chapter 2 that deadlock preventive measures taken by existing techniques [62, 103] try to avoid the deadlock situation occurring due to saturated links. Our proposed work takes one step further to tackle the saturated links by updating the routing trees. Moreover, the techniques used by existing methods use locking mechanisms or transaction priorities, which leads to starvation. However, in our proposed technique, if there is an edge that is saturated due to concurrent transactions, the path provided by the landmark node will change on another request from a non-landmark node due to the dynamic nature of spanning trees. This will resolve the dependency of transactions on the same edge and avoid starvation.



(a) *Nodes=2000 |T|=1000 LM deg=200*

(b) *Nodes=2000 |T|=1000 LM deg=200*

(c) *Nodes=2000 |T|=1000 LM deg=200*

(d) *Nodes=4000 |T|=1000 LM deg=400*

(e) *Nodes=4000 |T|=1000 LM deg=400*

(f) *Nodes=4000 |T|=1000 LM deg=400*

Figure 6.5: *Transaction count, transaction volume, and average path length over iterations*

6.4 Simulations

We evaluated our suggested approach with the well-known Speedy algorithm used by [82], and Webflow [111]. The LM route is utilized by Webflow and Speedy. We examined the outcomes of Speedy and Webflow after replacing the default routing algorithm with the $DEPR$ routing algorithm in order to evaluate the effectiveness of our suggested routing tree algorithm. The simulation setup and results are described here.

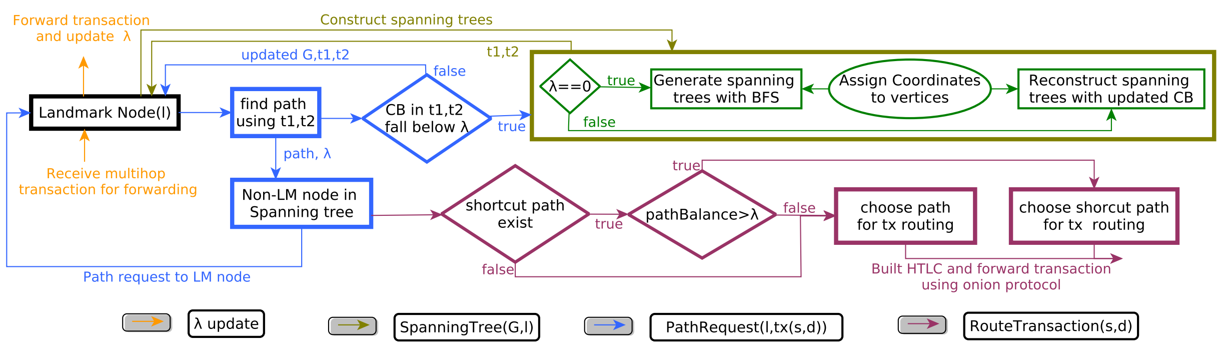

Forward transaction and update λ
Construct spanning trees
updated G,t1,t2
t1,t2
false
Landmark Node(l)
find path using t1,t2
CB in t1,t2 fall below λ
true
λ==0
true
Generate spanning trees with BFS
Assign Coordinates to vertices
Reconstruct spanning trees with updated CB
false
path, λ
Receive multihop transaction for forwarding
Non-LM node in Spanning tree
true
shortcut path exist
true
pathBalance>λ
false
choose path for tx routing
choose shorcut path for tx routing
false
Built HTLC and forward transaction using onion protocol
Path request to LM node
λ update
SpanningTree(G,l)
PathRequest(l,tx(s,d))
RouteTransaction(s,d)

Figure 6.6: *Workflow of DEPR*

Network Specifications

Various simulations were conducted by altering the input variables using the *DRLNsim* simulator. The number of nodes ranges from 1000 to 5000. The capacity of each channel is randomly set within the range of $500 - 1000$. While the number of iterations is set at 20, its value may be changed, but 20 iterations are sufficient to determine how our algorithm behaves over time. A landmark node has around 20% payment channels in network [17]. Consequently, the degree of each landmark node has been computed, amounting to around 10% of the total nodes in the network; and is varied till 20%. Non-LM nodes have an average degree of 7 [85] and is varied up to 25. During each iteration, the landmark node will be assigned a set of 1000 transactions that are randomly generated. For every transaction, a random pair is selected as the sender and recipient. The fund for each transaction is randomly chosen between 100 and 500; it is subsequently altered up to the channel capacity in $[500 - 100]$.

Results

Transaction count and volume: Figures 6.5a and 6.5d display the transaction count of the Speedy and Webflow algorithms compared to the DEPR approach. In both scenarios, the DEPR technique enhances the efficacy of the preexisting routing algorithm. Then we varied the size of the network to verify the adaptability of our approach, as shown in Figure 6.7. Figure 6.7a shows the transaction count of overall iterations with different G, other parameters are fixed as specified above. In every scenario, our method outperforms the current approaches since our suggested algorithm avoids channels with low capacity, which reduces the likelihood of transaction deadlock. The evidence shown by Figures 6.5b, 6.5e and 6.7c supports the claim that in the given scenarios, there is a similarity in both transaction count and volume.

Average Path length: The average path length of the Speedy and Webflow algorithm compared to $DEPR$, on each iteration is depicted in Figures 6.5c and 6.5f, and Figure 6.7c

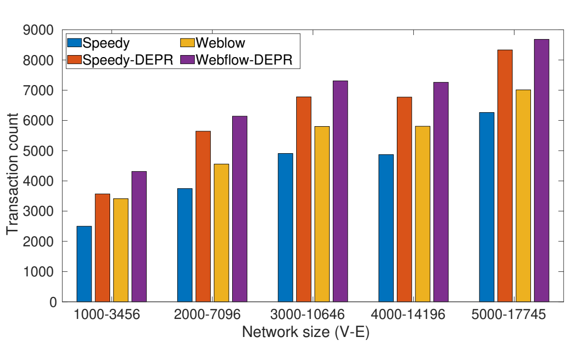
$G(V,E)$	Algorithm	Transaction count	Transaction Volume	Average path length
(1000,3456)	**Speedy-DEPR**	42.81%	43.44%	-0.34
	Webflow-DEPR	26.40%	24.25%	-0.45
(2000,7096)	**Speedy-DEPR**	50.66%	51.25%	-0.33
	Webflow-DEPR	34.78%	33.84%	-30.35
(3000,10646)	**Speedy-DEPR**	38.14%	37.00%	-0.28
	Webflow-DEPR	26.01%	24.71%	-0.30
(4000,14196)	**Speedy-DEPR**	39.05%	37.30%	-0.25
	Webflow-DEPR	25.03%	24.68%	-0.25
(4000,17745)	**Speedy-DEPR**	33.03%	31.94%	-0.22
	Webflow-DEPR	23.83%	22.12%	-0.20

Table 6.1: *A comprehensive comparison of DEPR with Speedy and Webflow on the custom network*

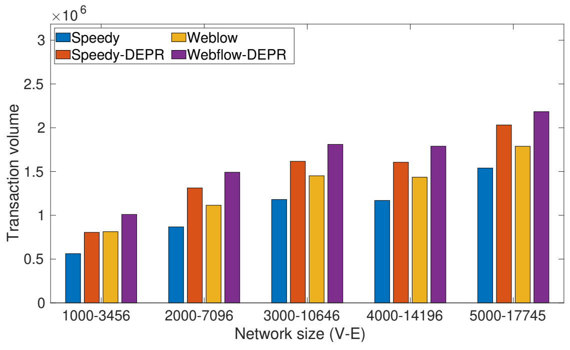
shows the average path length with change in network parameters. The analysis of the figure reveals that the path length remains rather consistent across all situations, with just a marginal increase of less than one hop observed in certain instances. The increase in path length is due to variations in the routing tree when CB falls below the λ. As usual, BFS will give the shortest path, but $DEPR$ will alter the routing tree according to Algorithm 6.1. Thus, the DEPR technique enhances performance by mitigating deadlock occurrences with a negligible increase in path length.

The improvement of $DEPR$ with regard to the Speedy and Webflow algorithms is shown in Table 6.1. The overhead of an increase in average path length in the suggested approach above SOTA is shown by the average path length's negative value. With the increase in the size of G, there is little drop in performance, as the probability of sending transactions through shortcut paths will increase and λ, will not be updated accurately. The DEPR technique effectively mitigates the occurrence of deadlock by employing the λ as a means to sustain an appropriate state of channel balances.

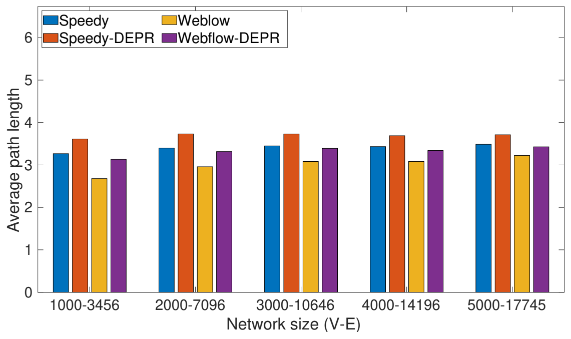
Same transaction set: The current approach takes advantage of landmark routing, which is inflexible. In order to evaluate the adaptiveness of the proposed routing method, we considered the worst situation. In the worst case, the transaction set won't change over all iterations, and the other simulation parameters won't change. In the worst-case scenario, Figure 6.8 shows the transaction count over iterations. Both Speedy and Webflow only survive for a few iterations, but with our proposal, both techniques are working fine. Speedy-DEPR is performing more consistently as compared to Webflow-DEPR, as the Speedy algorithm only uses a shortcut path within a subtree, whereas Webflow can also route transactions



(a) *Transaction count over the change in Network size*



(b) *Transaction volume over the change in Network size*

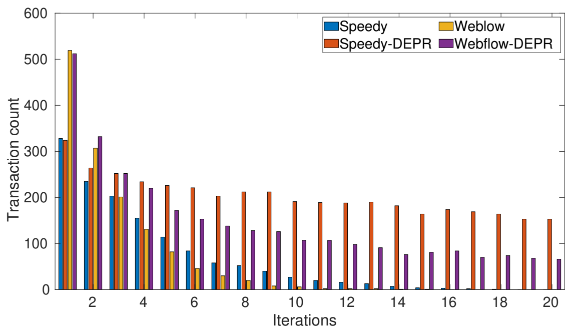


(c) *Average path length over the change in Network size*

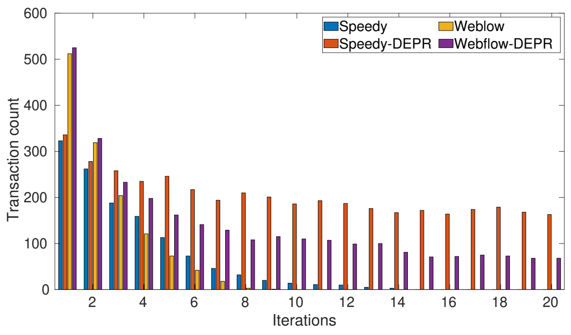
Figure 6.7: *Comparison Speedy and Webflow algorithms with DEPR*

inter-subtree. Thus, maximum transactions in speedy routed through the landmark node, which will provide updated paths with the help of $DEPR$, whereas Webflow will keep on trying shortcuts and is unaware of the channel balance.

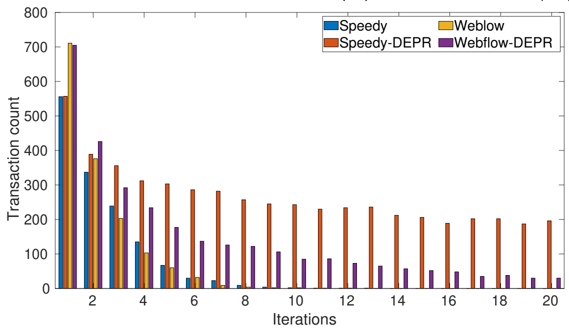
Change in LM degree: The transaction count during δt is displayed in Figure 6.9a with a change in degree; other input metrics remain constant throughout the iterations. The average degree, number of iterations, transaction number, transaction range and channel capacity for this 2000 node simulation are 7, 20, 1000, $100 - 500$, and 500-1000, respectively; the LM degree is adjusted from 50 to 400. The overall flow of transactions will grow as the LM degree increases, however, our suggested algorithm outperformed all other algorithms. Because $DEPR$ always selects the best channels in the network, it continuously performs better than the other algorithms. The average path length over δt with a change in LM degree is displayed in Figure 6.9b. As the depth of the routing tree reduces, the average path length of the routing tree also falls as the LM degree increases. However, because our suggested approach $DEPR$ replaces the routing tree's edges, it has a greater average path length than the Speedy and Webflow algorithms. But it is always less than 1, which is negligible.



(a) *Nodes=2000 |T|=1000 LM deg=200*



(b) *Nodes=3000 |T|=1000 LM deg=300*

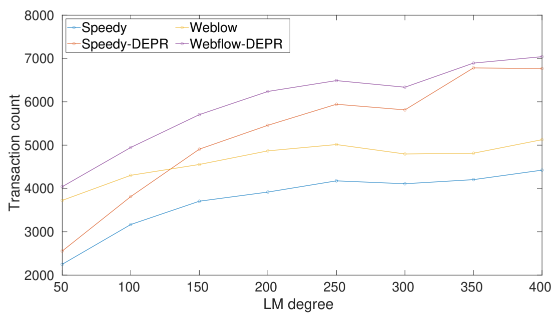


(c) *Nodes=5000 |T|=1000 LM deg=500*

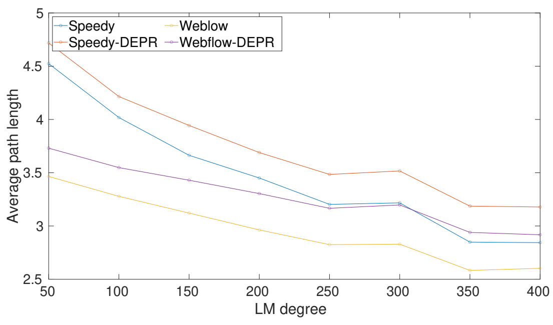
Figure 6.8: *Transaction count in worst case when transaction set is same in all iterations*

Change in average degree: Figure 6.10a shows the transaction count during δt with a change in the average degree of non-LM nodes; other input metrics remain constant throughout the iterations. The LM degree, number of iterations, transaction number, transaction range and channel capacity for this 2000 node simulation are 200, 20, 1000, $100 - 500$, and 500-1000, respectively; the average degree is adjusted from 7 to 25. Speedy and Webflow have no effect on transaction count with change in average degree, as same routing trees are used throughout the lifetime of the channel. However, our proposed algorithm improves with an increase in the average degree, as it exploits other existing links in the network while modifying the routing tree. As the average degree rises, there is a decreasing difference between the default algorithm and $DEPR$ since there is a greater possibility to replace edges without increasing the hop count, as shown in Figure 6.10b.

Change in transaction range: Figure 6.11a shows the transaction count during δt with a change in the transaction range; other input metrics remain constant throughout the iterations. The LM degree, average degree, number of iterations, transaction number, and channel capacity for this 2000 node simulation are 200, 7, 20, 1000, and 500-1000, respectively; the average degree is adjusted from 7 to 25. As the transaction range increases, the chances of deadlock will increase, as discussed in Section 6.1, thus the transaction count will decrease with an increase in the transaction range. But $DEPR$ is performing better as

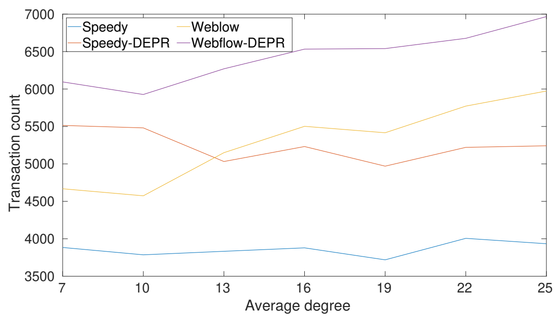


(a) *Transaction count*

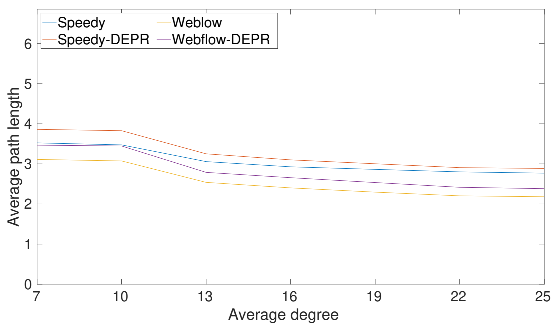


(b) *Average path length*

Figure 6.9: *Performance metrics with change in LM degree*



(a) *Transaction count*



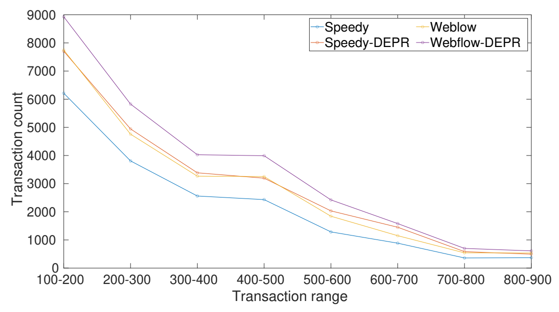
(b) *Average path length*

Figure 6.10: *Performance metrics with change in average degree*

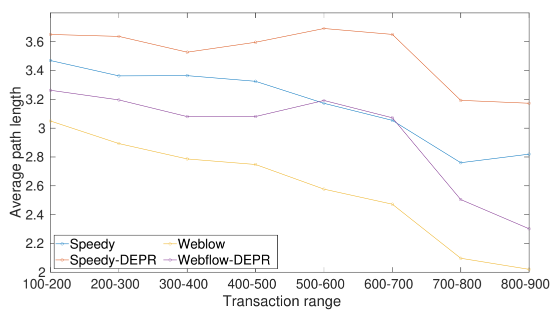
compared to existing techniques, but towards the end, all techniques almost become equal as the transaction range becomes comparable to channel capacity. The average path length in this instance is less than one, making it negligible.

Transactions and deadlock cycles: As we have discussed in Section 4.1, the increase in the number of transactions involved in cycles increases the possibility of a deadlock. We have also noted that deadlocks in cycles are caused by saturated links. To prevent deadlocks, the authors of [103] employed a locking technique to lock the edges prior to transaction routing. Figures 6.12 and 6.13 show how our suggested method and a locking mechanism utilizing the Webflow and Speedy algorithms are contrasted. For this 3000 nodes simulation, the LM degree, average degree, number of iterations, transaction number, and channel capacities are 7, 20, 300, and 500-1000, in that order. Cycle-related transactions in Webflow and Speedy show the same pattern in both locking and DEPR methodology. However, our suggested approach outperforms in terms of transaction count in comparison to the locking mechanism in both scenarios.

As our algorithm refreshes the routing trees during iterations, it reduces the possibility of having saturated links. It is, therefore, more effective. In contrast, the locking mechanism

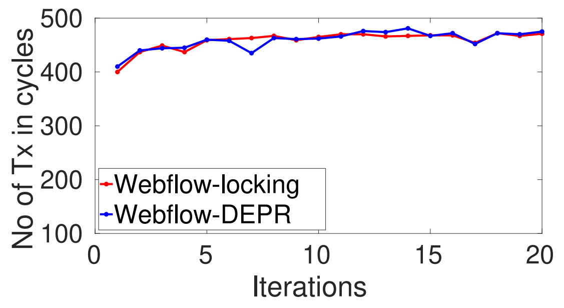


(a) *Transaction count*

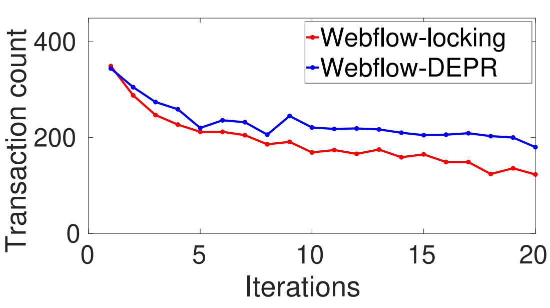


(b) *Average path length*

Figure 6.11: *Performance metrics with change in transaction range*



(a) *Txs in cycles*



(b) *Transaction count*

Figure 6.12: *Performance of DEPR verses locking mechanism in Webflow*

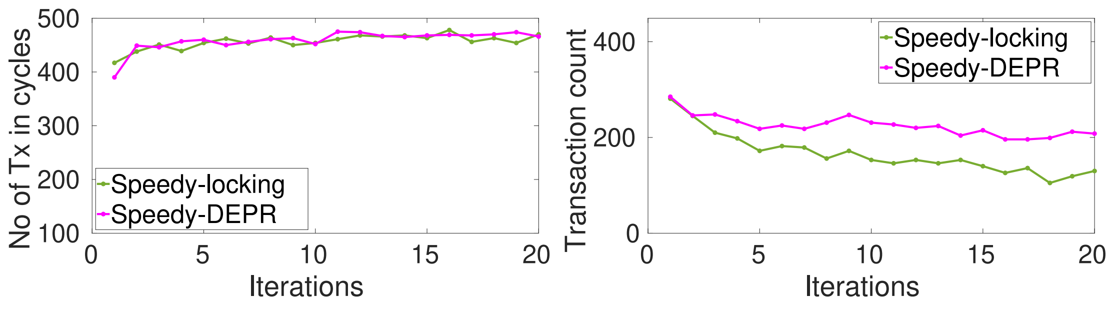
locks edges to prevent deadlocks; nonetheless, the efficiency is decreased overall due to starvation conditions caused by saturated links. As can be observed in Figure 6.13b, DEPR attempts to preserve transaction success across the iterations when combined with Speedy. Nevertheless, Webflow experiences less improvement than Speedy. This is because Speedy's transaction flow is primarily through landmark nodes, while shortcut paths are only utilized within subtrees. As a result, a more precise calculation of the λ value yields more efficient channel balancing. However, in Weblow, shortcut paths are not limited to subtrees; as a result, there is less flow via the landmark node than in Speedy, which will have an impact on the best way to calculate channel balances.

6.5 Discussion

We have shown the dynamic nature of resources, which will create a critical situation. This situation arises due to three reasons, viz.

(a) Multiple transactions using the same resource at the instance,

(b) resource consumption by substantial transaction, and



(a) *Txs in cycles* **(b)** *Transaction count*

Figure 6.13: *Performance of DEPR verses locking mechanism in Speedy*

(c) Improper utilization of resources during routing.

However, transactions in LN are considerably small compared to transactions on the main chain. So the occurrence of deadlock depends upon the transaction concurrency and edge resources. Concurrency and edge resources also depend on each other.

Deadlock Trilemma It is impossible to avoid deadlock in case of concurrency and limited resources. If a deadlock exists, it will affect the concurrency and resources. Concurrency also depends on resources and vice-versa, so simultaneously working on all three factors is impossible (shown in Figure 6.14).

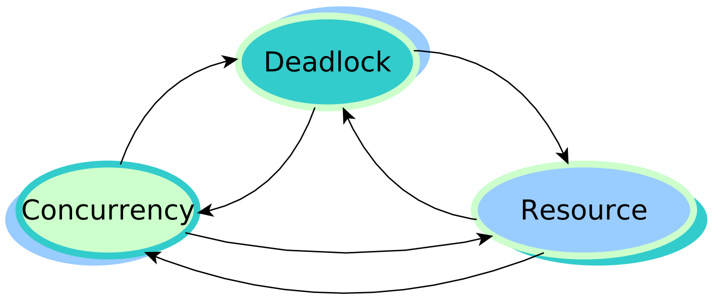


Figure 6.14: *Deadlock trilemma*

Concurrency Irrespective of our approach, whether it is centralized or distributed, concurrency will affect the deadlock as well as resources. All transactions are feasible if the resource has infinite instances and no deadlock will occur. But resources are finite; to make all transactions feasible, there must exist a path for each transaction. If for instance, there is only one transaction, it is equivalent to the max-flow problem; there will be no deadlock issue. The capacity of the minimum cut separation between source and destination sets an upper constraint on the amount of flow that may be delivered from source to destination [69]. In case there are multiple transactions, this can be referred to as a concurrent multi-commodity flow problem. Maximum multi-commodity flow problem is solved by LP, by maximizing the fraction of feasible commodities. But transactions are atomic; feasibility is an NP-complete problem even for two transactions. The possibility of cycles will increase with the increase in concurrency, as shown in Figure 6.2. So if concurrency rises, it increases resource utilization and the possibility of deadlocks.

Resource Resource instances will be used by parallel transactions, leading to two possibilities. Either transaction will get aborted due to insufficient resource instance, or deadlock will occur between multiple transactions as shown in CASE *a* and *b*. If the transaction is aborted due to an inadequate resource instance, it can be routed through another path next time. But if it is due to deadlock, the resource will still have available instances but insufficient for all transactions. Resources will satisfy the condition for each transaction in deadlock, individually, so resources will be underutilized due to deadlock. Some transactions in a deadlock can be delayed to make some transactions feasible; however, it is a tough decision which transactions should be delayed or how to order them to resolve the deadlock.

Deadlock The deadlock possibility escalates with increased concurrency as resource instances are confined. Assigning routes for concurrent transactions in the centralized scenario is an NP-complete issue such that the system is not in a deadlock [103]. Because no node has complete knowledge of the system's current state and because every inter-site connection requires a limited and unexpected latency, handling deadlock in distributed systems becomes extremely difficult. Existing distributed routing algorithms suffer from deadlocks, as we discussed in Section 6.1. Deadlocks can be avoided by having control over concurrency; concurrency depends upon the resources, and deadlock will affect resource utilization and concurrency.

6.6 Conclusions

The constraints of current distributed routing methods for PCNs and the presence of a deadlock scenario during the execution of concurrent transactions have been thoroughly examined. The dynamic nature of edges has been demonstrated, revealing its potential to worsen the deadlock situation. Theoretically, we conduct an analysis of the deadlock circumstances, the deadlock trilemma, and the NP-completeness of the D2TIR. In this study, we have put forth a novel strategy known as the $DEPR$ to mitigate the occurrence of deadlocks in currently employed distributed routing algorithms. To assess the effectiveness of our suggested approach, we conducted an evaluation utilizing the *DRLsim* simulator. The strategy we suggest leverages the transaction capacities that pass through prominent nodes in order to uphold channel balance. The utilization of channel balancing facilitates the development of an enhanced routing tree, hence aiding the sender in achieving improved routing capabilities. The proposed approach results in an increase in transaction count 41% in the Speedy algorithm and 27% in the Webflow algorithm. So far, the thesis has addressed factors related to scalability, including concurrency, fees, and deadlock. The other factor that will influence scalability is attacks; in the subsequent chapter, we will explore attacks that hinder the performance of PCN.

7 Attacks in PCNs

In previous chapters, we have covered various aspects related to routing. All of our proposed algorithms and existing algorithms exploit spanning trees to route the transactions. Attacks in PCN can be classified into three categories: congestion attacks, balance attacks, and theft attacks, as discussed in Chapter 2. In the third type of attack, an attacker wants to steal the incentives of other nodes in the network. The potential theft attacks on spanning tree-based routing protocols will be covered in this chapter. We investigated the effects of two types of theft attacks—wormhole and flood-loot—on a spanning tree-based routing algorithm.

The rest of the chapter is organized as follows: Wormhole and flood-loot attacks are explained in Section 7.1. Threat Model is described in Section 7.2, Attack implementation is described in Section 7.3. Section 7.4 presents the simulation setup; and the performance evaluation of the proposed attacks. We finally conclude in Section 7.5.

7.1 Wormhole and Flood-loot Attacks

Wormhole attack: In PCN, payment is usually sent across multiple hops because there might not be a direct link connecting the payer and the payee. In this case, the intermediates know who their near neighbors are, but they do not know about other nodes along the path or, in certain cases, even about the payment and payee. An attacker can launch wormhole attacks with just two malicious nodes on the path from the payer to the payee. The term “wormhole” refers to the method by which the money is transferred between the attacker’s malicious nodes via a wormhole channel [63]. The value of the random number r generated by the receiver is shared with neighboring nodes in the path in the backward

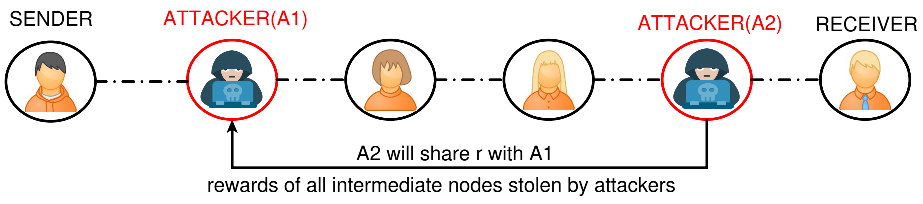


Figure 7.1: *Wormhmole Attack*

direction, as shown in step 3 in Figure 1.3. A malicious node, however, can bypass the r to another malicious node, omitting any nodes and channels in between. Because of this, the benign nodes believe the transaction has failed and return to their initial state, allowing the malicious nodes to take advantage of them and steal their reward. A wormhole attack scenario is shown in Figure 7.1. The attack needs the malicious nodes to be closer to the sender and recipient to have a greater impact.

Flood-loot Attack: During the assault, the assailant is capable of appropriating funds from victims who consent to establish channels with him. The assailant guarantees that numerous channels, burdened with various outstanding HTLC payments, will be terminated concurrently, resulting in a substantial influx of transactions simultaneously pending confirmation on the blockchain. Consequently, this congestion will cause some of the HTLC-claiming transactions submitted by the victims to fail to confirm prior to their expiration. The assailant can assert any funds linked to HTLC payments that have expired in this manner.

We will now outline the specific actions undertaken by the assailant from the initiation of the assault to the theft of the HTLCs, together with the responses of the victims. The processes are also delineated in Figure 7.2, with one victim node. After all channels are established, the sender attacker node (A1) initiates the transmission of the greatest feasible quantity of HTLC payments to the receiver attacker node (A2) via the victims' channels. A1 is directly linked to the victim node, however A2 is not subject to such a requirement. At that juncture, the A2 has not yet provided the preimages to settle the HTLC payments, retaining them all in a pending status.

At the conclusion of this phase, all compromised channels are inundated with outstanding HTLCs. Upon receiving the preimages, each victim requests the source node to settle the HTLCs and transfer their amounts to the victim's side of the channel. The source node ceases all communication with its victims and declines to settle the HTLCs. Subsequent communications from the victims are disregarded. Every compromised channel is now saturated with unresolved HTLC payments, and the victims possess all the necessary preimages to assert their claims. Due to the attacker's lack of cooperation, they are unable to assert these payments via Lightning interactions; instead, they must terminate their channels, publish their commitments, and claim the HTLCs on the blockchain. A sufficiently high quantity of compromised channels ensures that certain victims' transactions will remain un-

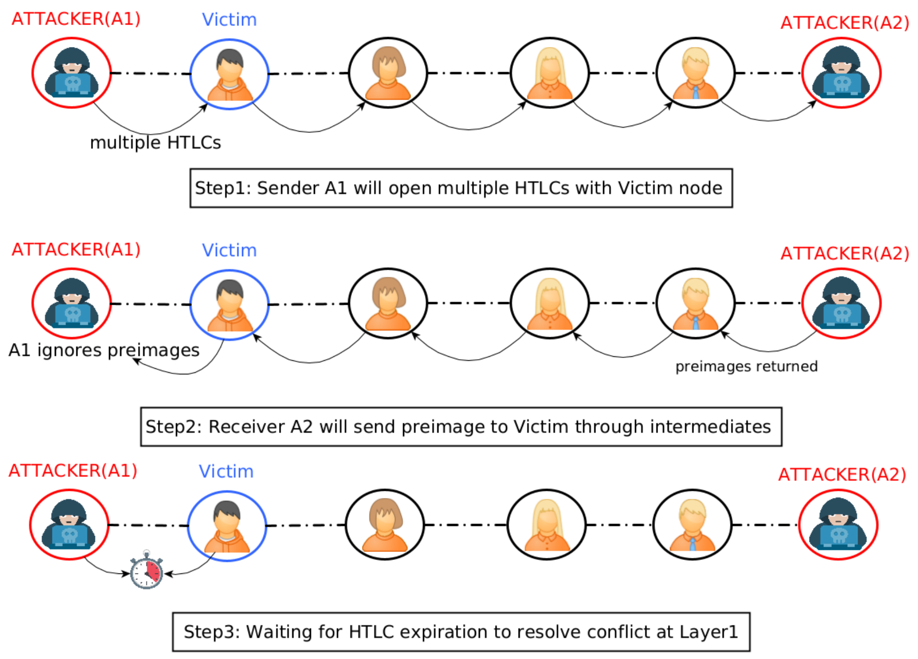

ATTACKER(A1)
Victim
ATTACKER(A2)
multiple HTLCs
Step1: Sender A1 will open multiple HTLCs with Victim node
A1 ignores preimages
preimages returned
Step2: Receiver A2 will send preimage to Victim through intermediates
Step3: Waiting for HTLC expiration to resolve conflict at Layer1

Figure 7.2: *Flood and loot Attack*

confirmed prior to the expiration of the HTLCs, just due to limitations in block space. Upon reaching the expiration height, the attacker can additionally utilize any unspent HTLC output using HTLC-timeout transactions. These transactions contradict any HTLC success disseminated by the victims as they attempt to utilize the same outputs. The assailant can substitute any victim's transactions present in mempools, as all are interchangeable through the Replace-By-Fee method. Each HTLC output that is successfully claimed by the source node is appropriated from the recipient of that HTLC, as this HTLC has previously been transmitted and claimed by the attacker's receiving node.

LM routing and Theft attacks

Wormhole: All distributed routing protocols exploit the LM routing to route the transaction. If attackers are placed near the LM nodes, the transactions passing through the LM node will pass through attackers, and the attacker will steal the reward of the corresponding intermediate nodes in the routing path. As shown in Figure 7.3, transactions $T1$ and $T2$ are routed through path $(S1, A1, 1, A2, R1)$ and $(S2, A4, 6, 1, A3, R2)$, respectively. Attacker pair $(A1, A2)$ and $(A3, A4)$ can steal the rewards of $T1$ and $T2$, respectively. Here, the placement of attackers and the properties of the PCN network will play a crucial role, as discussed in the next section.

Flood-loot: Attackers can seize the rewards of the relevant victim nodes in the routing

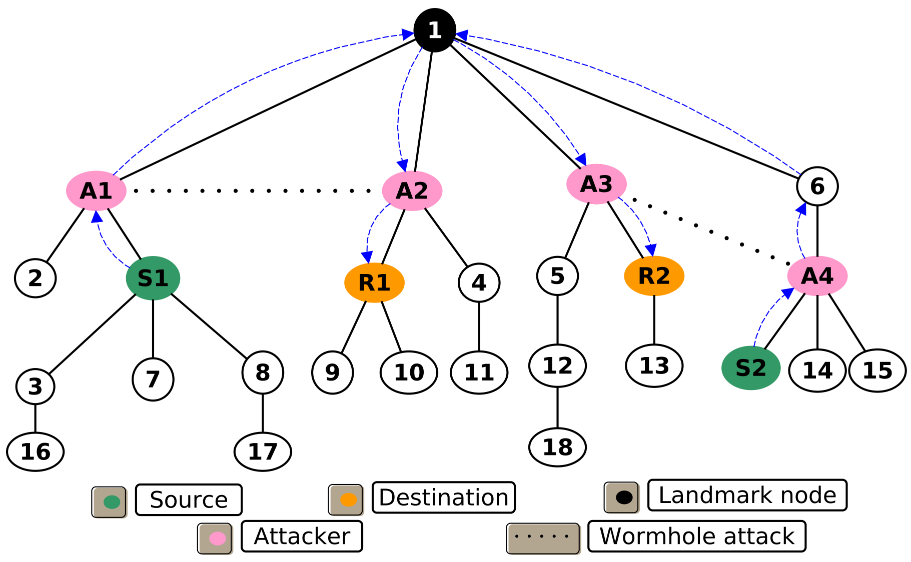


Figure 7.3: *Wormhmole Attack in LM routing*

path by positioning themselves within the network. The maximum amount an attacker can appropriate from the victim's nodes is the channel balance between the attacking node and the victim, excluding the transaction fees associated with the attacks. Figure 7.4 illustrates many potential attack trajectories based on the assailants' placement. If the attacker pair is in a distinct sub-tree, the source attacker will endeavor to pursue the shortest path, if available, to the destination attacker, as a shorter path entails reduced transaction fees and increased potential for theft. This indicates that attacker $A1$ will favor the route $A1, V1, 4, A2$ over $A1, V1, 1, 4, A2$. However, in a different scenario, if both attackers reside within the same sub-tree, they are constrained to a singular alternative, as illustrated by the attacker pair $A3, A4$ in Figure 7.4.

7.2 Threat Model

This section outlines the threat model and attacks that this study takes into consideration. Our focus is towards malicious internal attackers who seek to compromise the payment service's availability. Stated differently, the attacker's goal is to exploit protocol primitives to maximize the monetary benefit.

Incentives and computational capabilities: We investigate a computationally bounded, active, internal colluding attacker. The attacker can't break cryptographic primitives but has strong computational capabilities. It is presumed that malicious entities might receive financial incentives without causing harm to the system.

An attacker can establish completely controlled nodes and use techniques like social

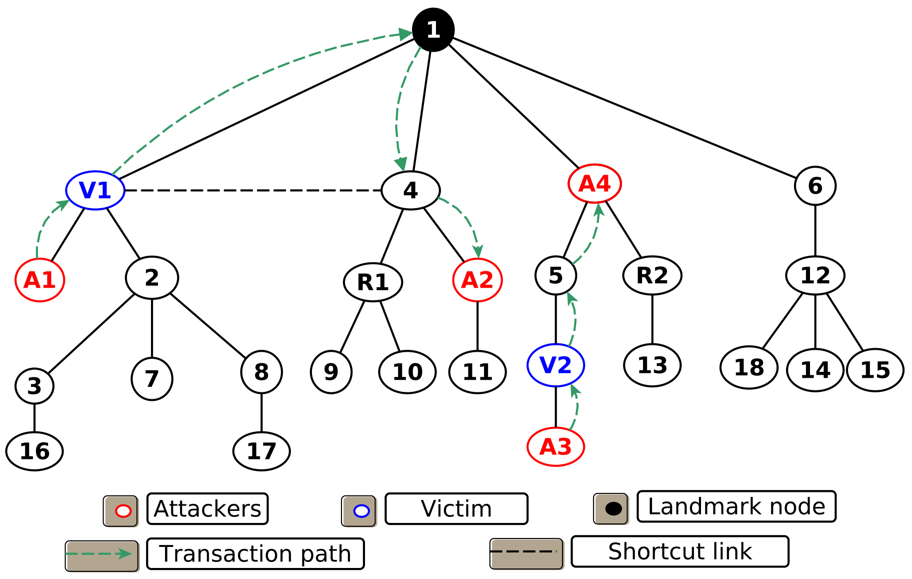


Figure 7.4: *Flood-loot Attack in LM routing*

engineering to taint previously truthful nodes. Attacker nodes cooperate and are dispersed geographically in random places. Out-of-band communication between colluding nodes can be quicker than PCN information. As a result, we presume that any information obtained by other hostile parties is known to an attacker node. Attacker-created nodes can be arbitrarily connected to other nodes in the network, even those under the authority of honest players, as demonstrated by earlier research [6, 25]. They cannot access the locally stored data on an honest node, though.

Network topology known to Attacker: An attacker can access the whole network topology because, as is standard with PCNs, all channel-opening transactions are visible on the main chain. Channel balances and Layer-2 transaction data, however, are not accessible. The attacker may not know the most recent balances on channels it is not directly linked, even when it is aware of the initial capacity. This is because, although a channel's opening balance is made available to the public, transactions involving two connected parties can alter the balance of a channel without requiring updates to be disclosed.

The attacker is also aware of the features of the routing mechanism. Nodes near the spanning tree root/LM forward more traffic. In the case of Silent Whisper, all transactions are routed through the LM node. However, the LM node is absent in the selected path if the source and recipient share a subtree, in the case of Speedy Murmurs, and if the transaction is routed through the cloud, in the case of Swift. The attacker can only attack if the transactions are routed through the LM node.

7.2.1 Attack Design

Since the Bitcoin Lightning Network is an open P2P network that anybody can connect to, an adversary can carry out the suggested attack with just the capacity to establish a payment channel with some node. Naturally, a node could decline to establish a payment channel with the attacker, as this process requires consent from both sides, preventing the initiation of an attack. To enable improved communication, nodes in the Lightning Network are anticipated to be receptive to opening additional channels. If the adversary finances the financing transaction as a whole, the node should not have to supply any liquidity and is, therefore, more likely to grant the adversary's request to open the channel.

We can take advantage of the fact that distributed routing algorithms [61, 82] rely on spanning trees. Damaging the root node is not the best course of action. Moreover, defenses against attacks on root nodes are in place [15]. Alternatively, it is simple for the attacker to get a position near the root. Positioning the malicious nodes will increase the chances of attack as more transactions will be attacked. However, if the malicious nodes are placed away from the root node or towards the leaf node in the tree, the reward earned by the wormhole node will be greater, as the number of intermediate nodes will increase, but the possibility of attacking a transaction will reduce.

Let G be a PCN graph and S be a spanning tree the LM node generates for G. The malicious nodes are distributed randomly in S at different depths defined by the Depth Ratio.

Definition 7.2.1 (Depth Ratio ($\mathbb{D}_r$)). Let m be the total number of malicious nodes. The depth ratio, D_r, is a proportion $(r_1 : r_2 : \ldots : r_k)$, that specifies that $\frac{r_i}{R} * m$, where $R = \sum_{j=1}^{k} r_j$, number of malicious nodes are placed at i^{th} depth from an LM.

In an attack, these malicious nodes share the secret value of random numbers generated by receivers among themselves to skip the intermediate node. In distributed routing algorithms, nodes do not know the network topology. Therefore, a node may not know the complete path or destination node due to the use of onion protocol [40]. The skipped intermediate nodes will assume the transaction could fail due to network failure or other reasons. When it is on-path, an attacker has several options. Unlike [80], who completely remove their chosen nodes from the topology, our chosen nodes carry out malevolent acts that are more difficult to identify. Moreover, our attack does not generate fake or intentional drop or grief transactions.

7.3 Attack Implementation

We implemented our attack using the DRLNsim simulator. Using the simulator, we create various LN topologies to observe how attacks affect routing tree strategies.

Network Parameters

The DRLNsim simulator was used to run several simulations by changing the input variables. The nodes vary between 2000 and 5000. Every channel has a capacity that ranges from 0 to 16777215 Satoshis; only 10% of channels have larger capacities, and 90% of channels have values that are not greater than the maximum transaction value [37]. Although the number of iterations is fixed at 50, we can alter it; 50 iterations is enough to see how our method performs over time. In a network [17], a landmark node has about 20% payment channels. As a result, the degree of every landmark node is roughly 10% of all the nodes in the network. The average degree of non-LM nodes is 7 [85].

A set of randomly generated 1000 transactions will be assigned to the landmark node every iteration. A random pair is chosen as the sender and recipient for each transaction. Every transaction involves a fund selected between 0 and 4294967 Satoshis [37]; its distribution is akin to channel capacities. We have used the default maximum values of transaction and channel capacities, which are generally used by the majority of users in PCN Networks. In a flood-loot attack, the source attacker node will generate 100 transactions, with values contingent upon the channel balance of the victim nodes, and the number of iterations is 1, as all the transactions are generated once with the same HTLC duration.

7.3.1 Evaluation Parameters

Three assessment metrics are used to evaluate and contrast the impact of wormhole attacks on different routing protocols. Attack gain, attack cost, and attack transaction ratio are the main measures used to assess an attack on a system.

Attack Gain Let n be the number of successful transactions in each iteration. Attacked transactions have two malicious nodes in their path; let A_1 and A_2 be such nodes having $\mathbf{d}(A_1, A_2)$ distance between them (defined in Chapter 4). Then, the gain from the wormhole attack is given by

$$\text{Attack gain}^{\text{wormhole}} = \sum_{j=1}^{n} T_j * \mathbf{d}(A_1, A_2) * (b_f + p_f * T_x^{fund})$$

Let r be the number of transactions generated by source attackers in a flood-loot attack. Then, the gain from the flood-loot attack is given by

$$\text{Attack gain}^{\text{flood-loot}} = \sum_{j=1}^{r} T_j * T_j^{fund} - \mathbf{d}(A_1, A_2) * (b_f + p_f * T_j^{fund})$$

where,

T_j	is 1 if j^{th} transaction is attacked, otherwise 0
b_f	base fee, kept as 1 Satoshi

p_f	proportional fee $= \frac{1}{10^6}$ Satoshis
T_j^{fund}	funds to be transacted

The default values of base and proportional fee [96] are used in the experiments.

Attack Cost A corrupt participant can be introduced into a PCN through one of two methods: infiltrating new malicious nodes or corrupting already existing nodes. Both are financially expensive, but we can only determine the malicious node cost inserted in a spanning tree using D_r. The inserted nodes have to replicate the node's properties at a particular depth. Let x be the number of malicious nodes inserted according to distribution D_r. Each malicious node x_i has to open y_i payment channels according to the position; for example, in Figure 7.3, the value of y_i for node A_4 is 4. The cost of attack is given by

$$\text{Attack cost}^{\text{wormhole}} = \sum_{i=1}^{x} y_i * T^{fee}$$

In case of flood and loot, one attacker will open only one channel so the attack will be given by

$$\text{Attack cost}^{\text{flood-loot}} = x * T^{fee}$$

where, $T^{fee} = 21335000$ Satoshis, is the fee to open the payment channel on the main chain [37].

Attack Transaction Ratio (ATR) Let p_i be the number of transactions attacked out of n_i successful transactions in i^{th} iteration and m be the total number of iterations. ATR will be the ratio of the total number of transactions attacked over the successful transactions during m iterations.

$$\text{ATR}^{\text{wormhole}} = \frac{\sum_{i=1}^{m} p_i}{\sum_{i=1}^{m} n_i}$$

ATR for flood and loot will be the ratio of the total number of successful transactions (q) generated by source attacker over the total number of transactions generated by source attackers(r).

7.4 Evaluation

Wormhole

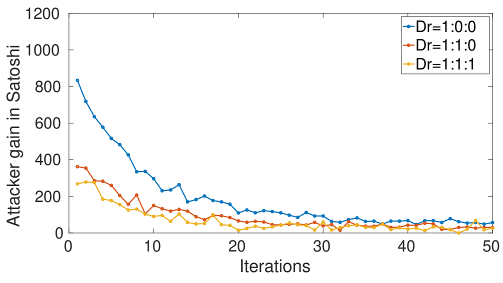
Attack gain and $\mathbf{D_r}$ The attack gain of Silent, Speedy, and Webflow algorithms over different D_r is shown in Figure 7.5. In all cases, the gain for $(1 : 0 : 0)$ is greater than other values. The probability of a transaction attack is higher when the malicious nodes are near the LM node. If we place malicious nodes even one depth down in the spanning tree, the

$G(V, E)$	D_r	**Silent**	**Speedy**	**Swift**
(1000, 3456)	**1:0:0**	64.75%	66.89%	58.38%
	1:1:0	19.51%	21.29%	17.88%
	1:1:1	11.93%	11.79%	10.31%
(2000,7096)	**1:0:0**	80.89%	82.29%	76.13%
	1:1:0	26.43%	27.57%	25.22%
	1:1:1	17.77%	15.77%	14.26%
(3000,10646)	**1:0:0**	85.67%	87.58%	83.41%
	1:1:0	29.50%	30.00%	28.15%
	1:1:1	13.50%	13.52%	13.19%
(4000,14196)	**1:0:0**	89.19%	88.93%	85.52%
	1:1:0	33.71%	32.68%	31.24%
	1:1:1	17.81%	19.24%	18.75%
(5000,17745)	**1:0:0**	89.62%	89.87%	86.64%
	1:1:0	35.16%	33.73%	32.28%
	1:1:1	16.62%	16.76%	15.95%

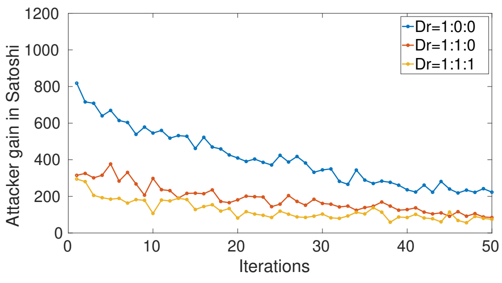
Table 7.1: *A comprehensive evaluation of wormhole attack on Silent, Speedy, and Swift with ATR on the custom network*

probability of transaction attack is reduced by the reciprocal of the average degree of non-LM node for each malicious node. However, the gain for each transaction with $(1:1:0)$ and $(1:1:1)$ is more as compared to for $(1:0:0)$, but the number of transactions attacked is more, which compensates for the gain. The malicious nodes are placed randomly, but their value is kept below the LM degree to avoid the possibility of placing malicious nodes at each sub-tree of the spanning tree with depth=1. We increase the number of malicious nodes with the increase in number of nodes in Figure 7.5, but the ratio is maintained. Due to the wide distribution of malicious nodes in larger spanning trees, the attack gain is greater, as we can observe in Figures 7.5g, 7.5h and 7.5i that with 5000 nodes, the attack gain is maximum for all routing algorithms.

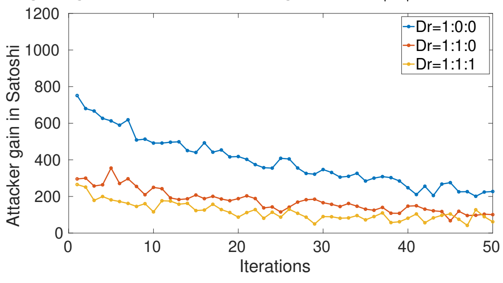
Attack gain and routing algorithms As we know, all routing algorithms exploit landmark routing. However, the Silent Whisper only routes transactions through the landmark node, whereas the other two routing algorithms, Speedy and Swift, use voute coordinates and cloud to route the transactions. This means that attack gain must be more in the case of Silent than the other two routing algorithms, but the results are the opposite, as we observe in Figure 7.5. Figure 7.6, shows the number of transactions attacked for $nodes = 2000$, same pattern is followed for different network parameters. We can observe that all three routing algorithms attack almost the same number of transactions in the initial few iterations, but



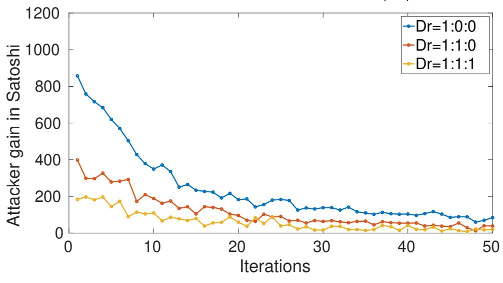
(a) *Silent, Nodes*= 2000, $|T| = 1000$, *LM deg*= 200, $|x| = 180$, *Avg deg*= 7



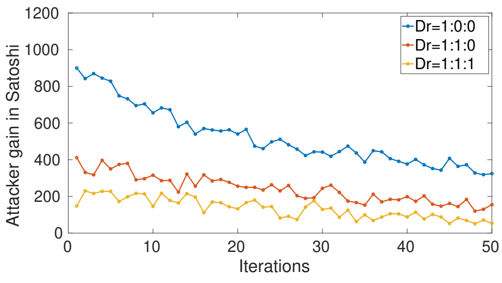
(b) *Speedy, Nodes*= 2000, $|T| = 1000$, *LM deg*= 200, $|x| = 180$, *Avg deg*= 7



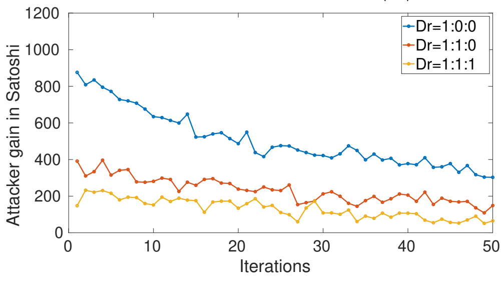
(c) *Swift, Nodes*= 2000, $|T| = 1000$, *LM deg*= 200, $|x| = 180$, *Avg deg*= 7



(d) *Silent, Nodes*= 3000, $|T| = 1000$, *LM deg*= 300, $|x| = 280$, *Avg deg*= 7

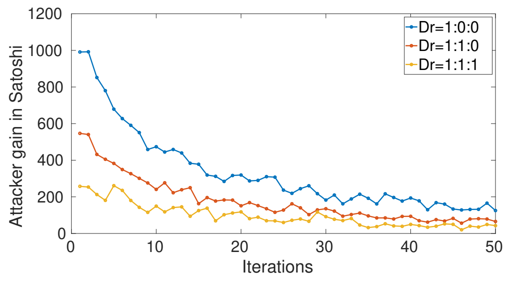


(e) *Speedy, Nodes*= 3000, $|T| = 1000$, *LM deg*= 300, $|x| = 280$, *Avg deg*= 7

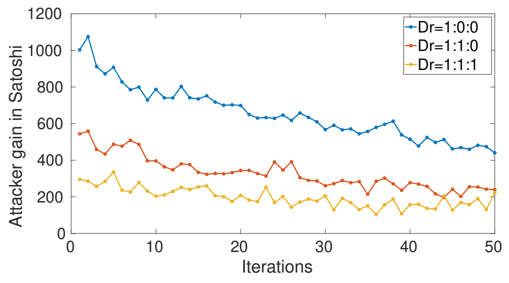


(f) *Swift, Nodes*= 3000, $|T| = 1000$, *LM deg*= 300, $|x| = 280$, *Avg deg*= 7

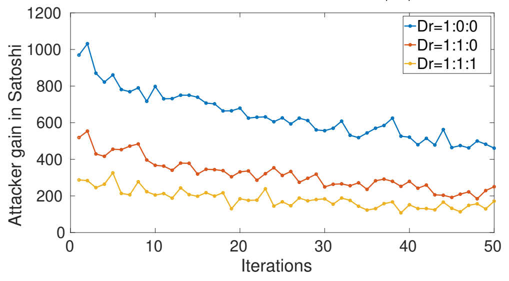
Figure 7.5: *Monetary gain over 50 iterations with different network parameters and routing algorithm*



(g) *Silent, Nodes*= 5000, $|T| = 1000$, *LM deg*= 500, $|x| = 480$, *Avg deg*= 7



(h) *Speedy, Nodes*= 5000, $|T| = 1000$, *LM deg*= 500, $|x| = 480$, *Avg deg*= 7



(i) *Swift, Nodes*= 5000, $|T| = 1000$, *LM deg*= 500, $|x| = 480$, *Avg deg*= 7

Figure 7.5: *Monetary gain over* 50 *iterations with different network parameters and routing algorithm*

after a few iterations, the Silent value decreases. The reason behind this behavior is congestion; in Silent, after a few iterations, nodes near the LM node become congested due to the concurrent flow of transactions, whereas in other routing protocols, all transactions are not routed through the LM node, so the possibility of congestion is less.

We reduced the number of transactions from 1000 to 200 to check the result of the attack gain for routing protocols. In Figure 7.7, we can observe that all three algorithms perform almost the same; this time, Silent doesn't show different variations as shown in Figure 7.5. As the number of transactions decreases, the concurrency will decrease, and congestion near the LM node will decrease; the Silent algorithm has the same attack gain as other routing algorithms.

Attack gain and malicious nodes Figure 7.8 shows that attack gains with an increase in the number of malicious nodes; it is obvious that with an increase in the number of attackers, attack gain will increase. If we compare transactions attacked in Figure 7.8aand attack gain in Figure 7.8b, we can observe that the number of transactions attacked with D_r of $(1 : 0 : 0)$ is comparatively very high as compared to other D_r values. However, the difference between curves in attack gain is smaller. This means that more transactions have to be attacked to have some gain with $(1 : 0 : 0)$, but the same gain can achieved by attacking fewer transactions.

Attack cost Figure 7.9 shows the attack cost with different numbers of malicious nodes

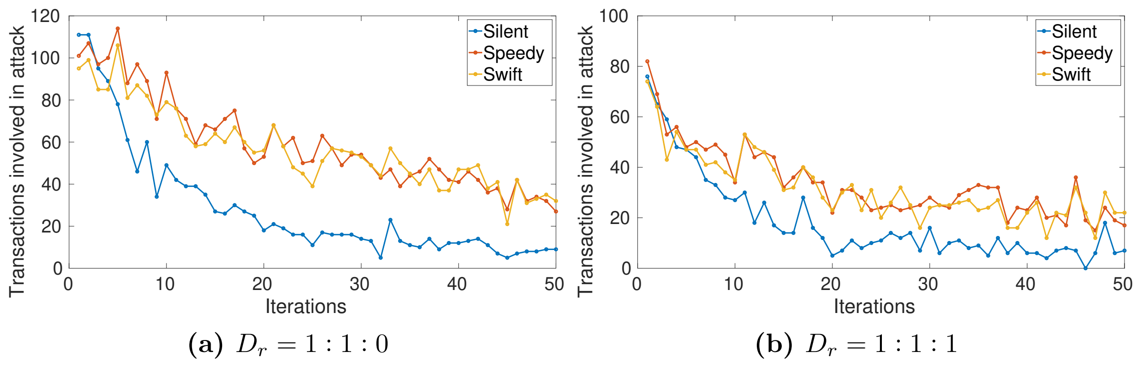
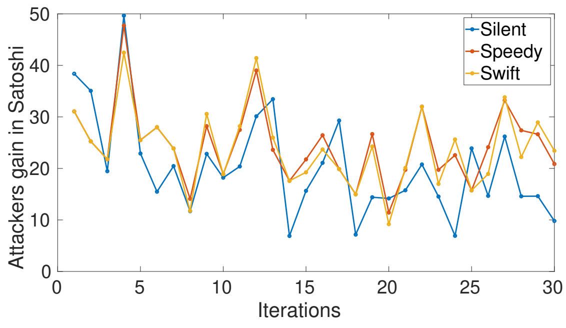


Figure 7.6: *Number of transactions attacked in 7.5a, 7.5b and 7.5c*

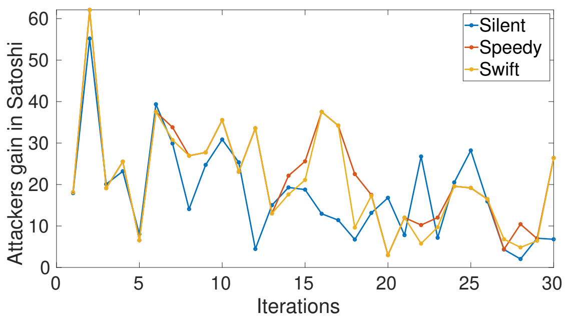
and different D_r with 5000 nodes. If all nodes are placed at depth 1, the attack cost is very high, and the degree of nodes at depth 1 is high, so more payment channels must be established to locate the malicious node. All the routing algorithms use BFS for spanning tree construction; as we move down towards the leaf nodes in the BFS spanning tree, the degree of intermediate nodes will decrease. Thus, the attack cost will reduce with a change in D_r values.

If we compare the attack cost in Figure 7.9 with the attack gain in Figure 7.5, the attack gain is a small fraction of the attack cost. We run very few iterations for attack gain, which cannot justify the attack cost. According to [107], the average channel lifetime is 30522 blocks or 212 days; the attacker will gain enough profit during its lifetime. Moreover, it is the attacker's choice whether to leave or join the network. Attackers can also increase or decrease the number of malicious nodes depending on the network topology. Here, we can consider that all the malicious nodes are placed at different positions for attack. The attacker can also communicate with existing true nodes to become malicious, which will cost less than opening payment channels as it is already in the network.

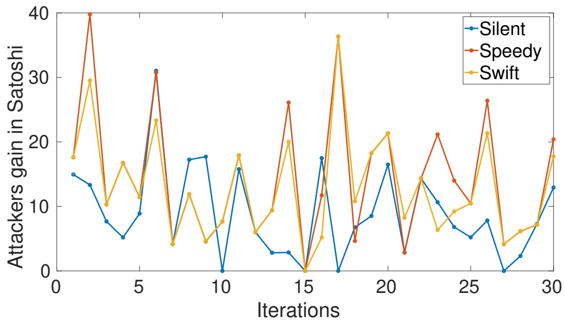
ATR Table 7.1 shows ATR values for all routing algorithms with different PCN network $G(V, E)$. It is very clear that the number of transactions attacked when D_r of $(1 : 0 : 0)$ is very high compared to other D_r values. The ATR value for $D_r = (1 : 1 : 0)$ is less than half compared to D_r of $(1 : 0 : 0)$, but if we observe attack gain in Figure 7.5, the difference between curves doesn't justify this difference. This means that with a lesser number of attacked transactions, we can achieve more gain with D_r of $(1 : 1 : 0)$ compared to $(1 : 0 : 0)$. A similar trend can be seen for D_rs, $(1 : 1 : 0)$ and $(1 : 1 : 1)$. This reflects the importance of the placement of malicious nodes in depth to the theft rewards of more nodes. Now, if we compare the attack cost in Figure 7.9, placing the nodes for $D_r = (1 : 0 : 0)$ is slightly costly compared to attack gain; observing values of evaluation parameters, it is better for the attacker to place malicious nodes at $D_r = (1 : 1 : 0)$. So if an attacker will implant 9.6% malicious nodes in the PCN network, on an average with different placements, they can attack 42.13%, 42.46%, 39.82% of transactions in Silent, Speedy, and swift routing



(a) $D_r = 1 : 0 : 0$



(b) $D_r = 1 : 1 : 0$



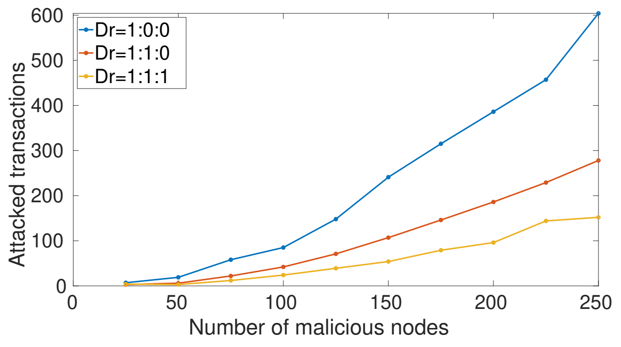
(c) $D_r = 1 : 1 : 1$

Figure 7.7: *Monetary gain over* 30 *iterations with Nodes* $= 2000$, $|T| = 200$, *LM degree* $= 200$, $|x| = 180$, *Average degree* $= 7$

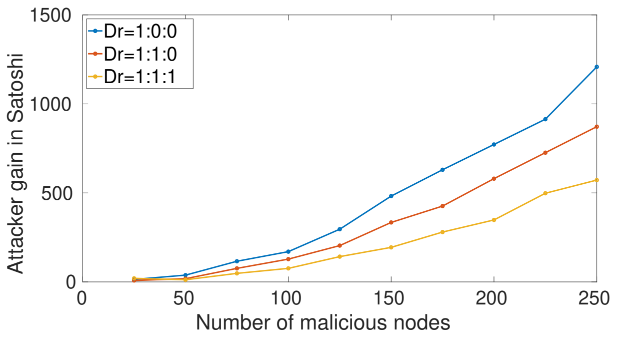
algorithms, respectively.

Flood-loot

Attack gain: The attack gain for Silent, Speedy, and Swift is illustrated in Figure 7.10a, depicting several attackers ranging from $6 - 60$; similarly, Figure 7.10b displays the number of transactions successfully affected by these attackers. Attackers are randomly positioned at the leaf nodes of the spanning tree, as these nodes incur the least costs. In the instance of Silent, nearly the same trend is seen between the transactions associated with the attack and the resultant attack gain; as the transactions related to the attack escalate, the attack gain correspondingly rises, and vice versa. Nonetheless, in the instance of Speedy and Swift, the trend is inconsistent. In Silent, all transactions are directed through the landmark node, whereas Speedy and Swift utilize intra-tree paths and the cloud to enhance routing efficiency. Consequently, if the number of transactions associated with attacks increases but no route optimization is implemented, the cost of the attack will diminish due to the higher transaction costs incurred. Nevertheless, with fewer transactions and more efficient routes, the profit from the assault increases due to reduced transaction fees for optimized routes. In all instances, attack gain is superior for Speedy and Swift compared to Silent due



(a) *Transactions attacked*



(b) *Attack gain*

Figure 7.8: *Number of transactions attacked vs. attack gain by varying malicious nodes with Nodes* = 4000, $|T| = 2000$ *and LM degree* = 300

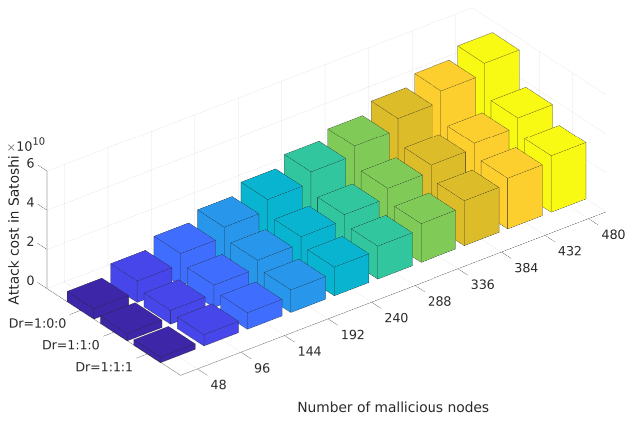
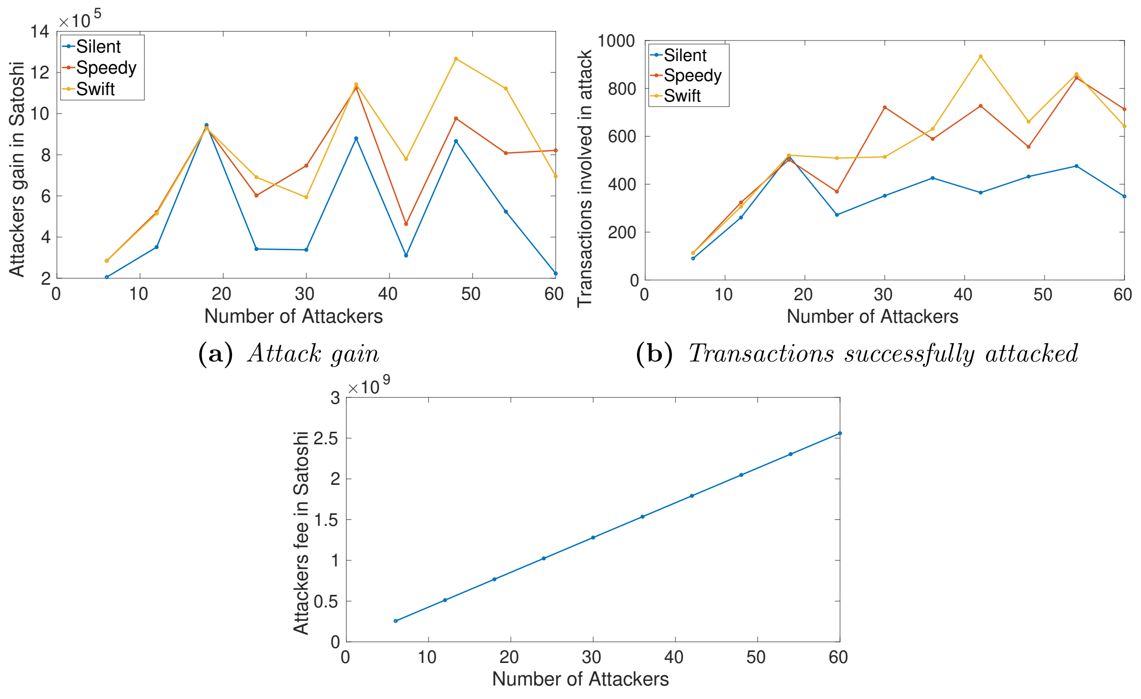


Figure 7.9: *Attack cost*

to enhanced routing efficiency. Figure 7.10b illustrates that when the number of attackers reaches 54, the transactions implicated in the attack are nearly identical for both Speedy and Swift approaches. However, the attack gain depicted in Figure 7.10a is greater for Swift than for Speedy, as Swift facilitates inter-tree optimization, whereas Speedy is confined to intra-tree optimization. A consistent trend is evident across various network sizes, with the attack gains for these sizes presented in Table 7.2. As the network expands, the average attack gain rises in correlation with the degree of landmark nodes, hence enhancing the overall transaction flow.

Attack transaction ratio: The transactions initiated by the source attacker will be influenced by other transactions occurring within the network. The aggregate ATR for all network sizes is below 20% while the number of attackers varies from $(6 - 60)$. To analyze the transactions associated with the attack, we adjust the number of attackers from 50 to 190, maintaining 2000 nodes; all other parameters remain constant as depicted in Figure



(a) *Attack gain* **(b)** *Transactions successfully attacked*

(c) *Attack cost*

Figure 7.10: *Flood-loot attack with Nodes* $= 4000$, $|n| = 100$, *LM degree* $= 400$, $|x| = (6 - 60)$), *Average degree* $= 7$

$G(V, E)$		**Silent**	**Speedy**	**Swift**
(1000, 3456)	**AG**	$1.6557e + 05$	$2.9873e + 05$	$3.0126e + 05$
	ATR	12.19%	18.21%	19.79%
(2000,7096)	**AG**	$4.7082e + 05$	$7.1018e + 05$	$7.0002e + 05$
	ATR	16.01%	21.98%	22.15%
(3000,10646)	**AG**	$4.3471 + 05$	$6.9981e + 05$	$6.9420e + 05$
	ATR	9.51%	14.40%	14.59%
(4000,14196)	**AG**	$4.9855e + 05$	$7.2829e + 05$	$8.0192e + 05$
	ATR	10.73%	16.53%	17.24%
(5000,17745)	**AG**	$1.0451e + 06$	$1.5106e + 06$	$1.5610e + 06$
	ATR	12.19%	18.76%	19.23%

Table 7.2: *A comprehensive evaluation of flood-loot attack on Silent, Speedy, and Swift on the custom network*

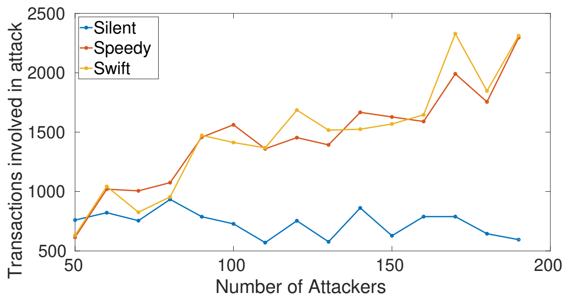


Figure 7.11: *Number of transactions attacked verses attackers*

7.11. It is noted that as the number of attackers rises, the transactions associated with attacks increase for Speedy and web flow, while remaining constant for Silent. This occurs because, in Silent, the majority of critical links become depleted, whereas other algorithms include other pathways for routing transactions.

Attack Cost: The attack cost for varying numbers of attackers is illustrated in Figure 7.10c. The outcome will remain consistent regardless of the network's size. The attack gain in all instances is lower than the attack cost. Does this imply that attackers will incur a disadvantage? No, as stated in [107], the average channel lifetime is 30,522 blocks or 212 days. No victim node will initiate numerous HTLCs with any unreliable nodes. The attacking nodes will persist within the network for reliability assessments and to generate sufficient revenue through transaction forwarding until they have accrued enough to avoid incurring losses upon initiating an attack.

7.5 Conclusions

We examined how successful assaults are against PCNs that route transactions using distributed routing technologies. Such routing algorithms' performance gradually deteriorates as the quantity of randomly positioned attackers rises. Wormhole attacks are highly susceptible, nevertheless, if the attacker manages nodes close to LM nodes and steals a significant number of transactions. We have also observed how different distributed algorithms, viz. Silent, Speedy, and Swift have behaved in the presence of theft attacks. When concurrent transaction flow is high, the Silent routing algorithm chokes the links near the LM node, which reduces the attack gain. In a wormhole attack, to get more gain while placing malicious near the LM node, more transactions need to be attacked; however, when malicious nodes are placed far from the LM node, more gain can be earned by attacking a lesser number of transactions. In a flood-loot attack, if the malicious node pair is positioned at a considerable distance from one another, the profit diminishes due to the increased transaction fees incurred. In this chapter, we evaluated the theft attacks on different distributed algorithms. In the future, we will devise solutions for attack detection and prevention.

8 Conclusions and Future Work

In this thesis, we have addressed the issues that arise when we attempt to scale transactions in the blockchain. We reviewed an extensive amount of literature and put it into groups based on Layer-1 and Layer-2 scaling solutions. Payment channel networks appeared to be the most promising Layer-2 solution to improve scalability. On beyond that, we talked about the strategic analysis of routing protocols in the payment channel network. The thesis encompasses several aspects of routing, including fee optimization, concurrency, deadlocks, channel balancing, and security.

We present a *Swift* routing algorithm, which prioritizes the fee optimization aspect of routing protocols. Inter- and intra-cluster mechanisms utilize the non-tree edges of the cloud to optimally route transactions. Intra- and inter-tree transactions will enable a node to remain cognizant of its neighbors across multiple hops. We theoretically evaluate our proposed technique by establishing that it is adequate to examine the cycle topology and k-cycle graph concerning junctions and junction cycles. The anticipated enhancement per transacand cycle graphs exponentially as the neighborhood hop count escalates until the complete cycle is within the neighborhood. Swift routing attains fee and throughput optimizations of up to 21% and 63%, respectively. The simulation results align with the trends established in the theoretical analysis.

We present $maxECW$ and $maxSCL$ methods for concurrency and channel balancing. Our algorithms consider channel weights and rebalancing to avoid directional capacity saturation. $maxECW$ optimally modifies the routing tree according to revised channel weights to use the best channels in every iteration, avoiding MSL congestions, while $maxSCL$ changing channel flow direction to avoid link saturation. We have also developed a simulator, $DRLNsim$, to compare our proposed algorithms with the existing ones. The simulator evaluates routing strategies by adjusting parameters. Overall, our algorithms handled concurrent transactions more effectively than existing methods by 50%.

The implementation of $maxECW$ and $maxSCl$ enhances concurrency; however, it also results in an increased path length. We propose a method that employs dynamic programming to reduce costs and enhance transaction concurrency. This approach combines the advantages of both $maxECW$ and $maxSCL$, while mitigating their respective drawbacks. This study assesses the substitution of critical edges with enhanced alternatives to prevent the saturation of a channel's directional capacity while maintaining the integrity of the underlying routing tree. The proposed method enhanced transaction flow through self-rebalancing and link load-sharing capabilities. A theoretical analysis of load balancing and congestion at the landmark node is conducted. A theoretical analysis of transaction volume is performed by rerouting transactions through alternative routes. The simulator assesses routing methodologies through various custom network dimensions. The proposed technology enabled concurrent transactions with an average efficiency increase of 58% compared to existing techniques. $maxREE$ enhances throughput by 8% relative to $maxECW$ and $maxSCl$, utilizing identical network parameters while significantly decreasing the average path length. Our simulations are conducted under the assumption of no network failure. The proposed approach recalibrates routing trees in response to node failure and adapts to changes in network topology. The simulation results align with the patterns observed in the theoretical analysis.

We demonstrate the potential for deadlock in distributed routing algorithms and provide the DEPR technique, which effectively reduces deadlock incidents. The suggested methodology utilizes the capabilities of transactions traversing critical nodes and guarantees that the channel sustains adequate balance in both directions. We demonstrate that the D2TIR problem is NP-complete. The task of identifying individually valid paths for a set of transactions that are impervious to deadlock is referred to as the "safe routing problem," which is classified as NP-complete. The D2TIR problem demonstrates that routing two transactions in a PCN is NP-complete. The experimental assessment indicates that the proposed methodology enhances transaction count in the Speedy and Webflow algorithms by 41% and 27%, respectively.

Finally, we analyze the impact of theft assaults on PCNs using local knowledge-based routing algorithms. This study focuses exclusively on theft attacks where the assailant takes over the incentives or channel balance of the victim nodes. In such scenarios, malicious nodes can exploit the benefits of cooperative nodes by exchanging sensitive information among themselves. Our research introduced an attack that utilizes a depth metric to strategically position malicious nodes in various locations. We assessed the impact of the assault on routing algorithms through metrics of attack gain, cost, and transaction ratio. In the wormhole attack, with 9.6% of malicious nodes distributed across different locations, the average attack transaction ratio for all distributive routing algorithms is 41%. In the flood-loot attack, with a relatively small percentage of malicious nodes (ranging from 0.66% to 3.3%), the average attack transaction ratio across all evaluated routing algorithms is a concerning 16.23%.

Future Work

1. Our proposed method improved throughput and concurrency at the tradeoff of higher transaction costs. When a landmark node receives a transaction route request, it does not use any scheduling algorithms. We already proved that the scheduling of even two transactions is NP-complete in PCN. However, sequencing and delay can help with concurrency. In the event of concurrency, all transactions are not viable.
    (a) However, we can use past transaction information routed through the landmark node to determine which group of transactions has a higher probability of feasibility, allowing us to improve concurrency. We need to add a learning model to our existing DRLNsim simulator. We can validate our model by running various simulations with all possible scenarios we have already done in our previous works.
    (b) With the provided PCN network and transaction parameters, we may devise an approximation algorithm that will provide the upper bound of the maximum number of viable transactions.
2. We have also gone through other Layer-2 solutions, such as plasma, polygon, and *zk*-rollups. While there is a lot of research on *zk*-Rollups, none of them provides a clear understanding of the technique's throughput. We will analyze how factors like proof creation time, transaction aggregation, data availability, and network circumstances impact throughput. We will explore key components for high throughput, such as *zk*-SNARKs/STARKs and Layer-1 blockchain interfaces.
3. Since PCN combines scalability and security, it can be utilized as a solution in real-world applications in addition to currency exchange. PCN has been used by certain scholars to improve transaction efficiency in electronic toll collection and green electricity trade. We will attempt to apply our suggested algorithms to more recent real-world uses in the future.

A
Scalability Solutions for Blockchain

A.1 Layer-1 Protocols

Chain-based PoS

Peercoin and Nxt

In Peercoin [45], the stake value deposited at the beginning by the node will appreciate linear with time. When a node mines a block, its stake value is reset to the base value. To avoid deliberate waiting, the stake value only appreciates for 90 days, then reset to the base value. In Peercoin, the chances of an attack are higher, as the malicious node will wait for a maximum of time and then mine a block. Nxt protocol [105] reduces this attack chances; it appreciates stake value only during one block cycle. The incentive is also divided among all stakeholders as compared to Peercoin; only miners get the incentive.

Bentov's PoA

It is a combination of both PoW and PoS, also known as Proof of Activity (PoA). An empty Block (with only a block header) is generated according to PoW and then broadcast to the network. Some pseudo-random stakeholders are selected to check the validity of the Block. The last stakeholder from the selected ones will add as many transactions to the Block and broadcast the wrapped Block. The transaction fee is shared among all selected stakeholders.

Committee-based PoS

In committee-based PoS, miners need not compete for mining a block. No hashing puzzle is required; one particular node is selected to create a block. Some committee-based protocols are described below:

Bentov's CoA

In Bentov's Chain of Activity (CoA) [10], when a node joins the network, it has to deposit a stake. The committee selects a node to generate a block then the Block is added to the blockchain. Other rules, such as the longest chain rule will remain the same as in the Nakamoto consensus.

Ouroboros

Ouroboros [44] divides physical time into n slots. Each slot is associated with a slot leader to propose a block. Only nodes with enough stake can become electors for leader selection. As there is one leader per slot, it will not cooperate with a partially synchronous system. So this protocol is modified to Ouroboros Praos [21] with multiple slot leaders. Ouroboros has one more issue: all nodes in the network know leaders of all time slots. An attacker may try to corrupt the slot leader by adding unwanted transactions to his Block while proposing a block. So Ouroboro Proas uses a verifiable random function, so only the elector knows its blocking slot.

Snow-white

Snow-white [20] is the modified version of the sleeping consensus protocol [72]. This protocol is designed for a particular situation where nodes are not reliable. Nodes may switch to online and offline modes arbitrarily, and message delays are volatile. A checkpointing scheme is used in this protocol to protect the blockchain from attacks.

Hybrid Consensus protocols

A hybrid consensus protocol is a combination of SMR-based (traditional) consensus protocols and blockchain-based protocols. As we have studied, the throughput of SMR-based protocol is very high as compared to blockchain-based protocol. SMR-based protocols are non-scalable as compared to blockchain-based protocols. So, researchers tried to combine the benefits of both technologies. We have discussed some hybrid protocols below.

SCP (Scalable Consensus Protocol)

SCP[58] combines the SMR and PoW protocols. In this approach, network nodes are divided into subcommittees. Each sub-committee runs classical BFT to process a different set of transactions. The division of the committee is done with the PoW mechanism. A dedicated final committee is used to combine outputs from sub-committees to make a block and add it to the blockchain. The key idea behind the sub-committee is to limit computational power to the sub-committee and to increase the throughput.

ByzCoin

This protocol combines the Bitcoin NG protocol with traditional PBFT. As we have studied Bitcoin NG, it consists of two blockchains, the key blockchain and the micro blockchain. In ByzCoin protocol, the key blockchain is mined using the PoW mechanism, and micro blockchain adopts PBFT protocol based on collective signing (CoSi) [91]. Cosi reduces the message complexity to $O(N \log N)$. Pass, and Shi's extended the Byzcoin work to reduce the susceptibility to fork. They used fruit-chain [71] as the PoW mechanism. The fruit-chain mechanism reduces the variance of mining reward, thus reducing the need for a mining pool.

There exist many other hybrid protocols which combine PoS with BFT-based protocols, such as Tedermint [105], Algorang [32], and Casper FFG [14]. These protocols add an extra layer of BFT to improve block finalization.

Sharding

In a sharded blockchain system, the nodes in the network are dynamically partitioned into subsets known as shards. Each shard is responsible for performing storage, communication, and computation tasks individually, without requiring fine-grained synchronization with each other [54]. Scalability is achieved through sharding, as the addition of newer nodes and the creation of additional fragments will result in increased task processing and improved parallelism. In monolithic blockchain networks, the complete state is stored on each node, and each block is shared with all nodes prior to reaching a consensus. Therefore, the transmission is reduced as the number of nodes in the network increases, which is due to the increase in overhead. The overhead is reduced by sharding, which divides the network into smaller fragments, resulting in efficient computation, storage, and data transmission. ELASTICO [59] was the first to implement sharding to blockchains in order to increase the speed of transactions. OmniLedger [46], CycLedger [110], RapidChain [109], and Monoxide [101] are among the numerous sharding-based protocols that have been proposed in the literature since ELASTICO. ELASTICO [59] developed a comprehensive protocol that outlines the use of sharding to achieve a performance improvement in the network's block throughput and latency. The local set of transactions that have been agreed upon by a shard is established in this protocol. A consensus committee, which is a specialized body, combines and validates the sets of each shard. The consensus committee has completed the processing of transactions, and the valid transaction blocks are broadcasted to the entire network. OmniLedger [46] extended ELASTICO by utilizing an Atomic Commit Protocol known as Atomix to resolve cross-shard transactions. It enhanced efficacy and resilience by employing ByzCoinX, a variant of the BFT consensus protocol. In addition to these, OmniLedger implemented sharding techniques to reduce blockchain storage. CycLedger[110] enhanced the sharded blockchain systems by incorporating incentive mechanisms to encourage nodes to maintain integrity and by introducing techniques to enhance resilience in the event that the leaders of fragments (or committees) are malicious.

A.2 Layer-2 Protocols

Commit Chains

Channel-based solutions, like PCN, need users to create specific channels where money is locked and cannot be removed. Users also need to stay online to receive money, among other requirements. To solve these problems with channel-based scaling solutions, commit chains [42, 43] were introduced. Commit chains use a non-custodial operator to initiate and maintain a committed chain, while a smart contract stops the operator from acting inappropriately. To obtain an account ID on the commit chain, participants interested in joining must first register with the operator. Through the smart contract, participants lock up money, and the operator reads and updates the associated account IDs on the commit chain. Fund deposits are not required of the recipients. The sender gives the operator permission to debit its account to transfer money. The transactions between the participants are handled off-chain by the operator. Also, the operator uses constant-sized checkpoints in the smart contract to periodically commit the most recent balances of participants' accounts to the main chain. In the event of a disagreement, participants use the smart contract to challenge the operator while keeping an eye on the checkpoints. In addition to penalizing the operator for misbehaving, the smart contract stops the commit chain from getting the balances from the most recent stable checkpoint. Lastly, a participant can either force an exit by using the smart contract to close and return all of their account IDs, or they can withdraw money by submitting a withdrawal request to the operator.

Plasma

The Plasma [75] chain is another associated idea. Comparing plasma chains to commit chains reveals some important drawbacks and problems. Thus, we will quickly go over its main idea and issues. For a thorough explanation, we direct interested readers to the study [75]. Systems that use accounts, like NOCUST [42], are utilized in commit chains. A UTXO-based ledger system over an account-based blockchain, like Ethereum, is what Plasma Chain offers. Plasma uses a set of smart contracts to allow many blockchains to coexist as branches of a tree. Every branch, or blockchain, is capable of having child chains or sub-branches. Every Plasma chain has the ability to function independently of its parent chain by maintaining its own block validation process. Nonetheless, every calculation within the chain hierarchy is globally mandated or reliant on a solitary root chain. Several problems plague plasma chains, including rapidly rising data storage expenses, high processing demands, and a lack of native support for instantaneous finality.

Rollups

Non-custodial side chain solutions called rollups are designed to lighten the load on the main chain. Rollups use smart contract and data compression techniques to scale Layer-1 chains. The idea is similar to a plasma chain, with the exception that rollups only store a small amount of state update-related data (a Merkle root) on the chain. These data make faster withdrawals and on-chain verification possible. The transactions are packaged together once they execute off-chain in batches for on-chain verification. Specifically, the smart contract keeps the Merkle root on-chain from the Rollup's current state. With the data that is available on-chain, the same root can also be calculated or confirmed. To save space, the Merkle tree is not kept on-chain. After a series of transactions causes balance updates, a new state root is calculated. By including the transactions in a compressed format, the old state root, and the freshly computed state root, anyone can publish the batch. The contract updates its state root to the new state root if the current state root matches the prior state root specified in the new batch. If a batch needs outside inputs or outputs, the necessary money is provided to the contract before the transactions are processed, or it is sent to the outputs after the transactions are processed. Two forms of rollups result from the process of preventing fraud and confirming the new state root: Optimistic and Zero-Knowledge (zk) Rollups. This is because anyone can broadcast the batch of transactions.

(i) Rollups that adopt an optimistic stance and presume that transactions are legitimate unless contested are known as optimistic rollups. Therefore, to greatly increase scalability, no calculation for verification is done by default. On the other hand, a history of state root modifications and associated batch hashes are retained by the contract. A fraud-proof, or a proof of faulty computation, must be published on-chain and confirmed by the contract to challenge a batch. The contract reverses the erroneous batch and all the following batches after verification.

(ii) zk-Rollups: These rollups distrust each transaction, in contrast to Optimistic Rollups. Each batch includes a cryptographic validation proof (also known as a proof of validity) demonstrating that the output of executing the batch of transactions indeed matches the new state root. PLONK protocol and zk-SNARK are used in the construction of such proofs [27]. Although computing validity proofs are difficult, on-chain verification proceeds swiftly.

Hybrid Solutions

Hybrid solutions improve the scalability of off-chain protocols. Hybrid solutions differ from off-chain solutions by changing certain essential features. We identify two types of solutions: one tries to reduce on-chain reliance on dispute resolution processes, while the other employs safe execution to eliminate peer trust needs.

Bisection Protocols

Hybrid solutions improve the scalability of off-chain protocols. Hybrid solutions differ from off-chain solutions by changing certain essential features. We identify two types of solutions: one tries to reduce on-chain reliance on dispute resolution processes, while the other employs safe execution to eliminate peer trust needs.

Truebit [95] is a blockchain innovation that improves computational efficiency and reduces costs for smart contracts. It transfers computations from the main chain to the off-chain. Truebit relies on judges, who have limited computational capabilities. All participants widely trust the judges. When a computational problem is solved, the solver publishes both the answer and the sub-problems that led to it. A challenge period allows other users, known as challengers, to challenge the published solution. When a challenge is made, judges use binary search to answer sub-problems recursively, reducing the problem size by half with each repetition. All participants agree that the judges' solution is correct. The judges compare their right solutions to those provided by the solver and challenger to identify and penalize cheating participants.

Arbitrum employs a Virtual Machine (VM) to implement smart contracts. A user can establish a VM and designate other users as managers. An honest manager forces the VM to obey its coded functionality. Any changes to the VM require approval from all management. Managers may disagree on the state of a virtual machine (VM). Disagreements should be raised during the challenge period after a new state is committed. If there is a conflict among managers, the verifiers/miners use a bisection procedure to execute a single instruction. Managers now present the outcome of executing a single instruction.

TEE-based Solutions

A trusted execution environment, such as Intel SGX [19], isolates and protects the integrity and confidentiality of loaded data within a CPU. TEE-based blockchain scalability solutions minimize the need for on-chain collateral to establish confidence among participants by leveraging its integrity protection. TEE is a trusted entity in such solutions, providing enhanced application security.

Teechan [51] utilizes TEEs to allow two distrusting nodes to communicate with one another. The two nodes establish a channel by sharing secrets through their TEEs. Nodes can exchange funds peer-to-peer utilizing TEE-supported activities as long as the channel is available, without relying on the parent Bitcoin blockchain. TEEs are responsible for securely updating the channel's state over its lifespan. When a channel ends, the TEEs generate a Bitcoin transaction to be added to the parent chain. During the channel's lifetime, only two transactions are reflected on-chain: one for channel setup with a 2-of-2 multisig.

Teechan minimizes the burden on the parent chain while increasing transaction throughput among distrusting nodes. Teechain [52] is a solution that conducts off-chain transactions asynchronously with the main chain. Teechain uses TEE-protected treasury to maintain the proper channel state. Teechain creates a chain of committees to duplicate treasuries and handle failures.

A.3 DAG based Solutions

The directed acyclic graph (DAG) structure in graph theory was added to the original blockchain topology to increase its scalability. This structure allows a block to reference more than one previous block, and it also allows two or more blocks to reference the same prior block. As a result, in a DAG, blocks can be formed concurrently and may contain transactions that clash. The DAG structure shortens confirmation times by decreasing latency and increasing blockchain throughput (TPS). However, the main security problems of the existing DAG-based blockchains are double-spending and fear of centralization. The following are the main DAG-based blockchain scaling proposals:

IOTA

IOTA [92] introduces the term tangle. A tangle is a directed graph in which a transaction is a vertex. When new transactions are added to the graph, it has to choose two existing transactions in the tangle. As shown in Fig A.1, transaction 5 approves transaction numbers 2 and 3. Unapproved transactions in the graph are known as tips.

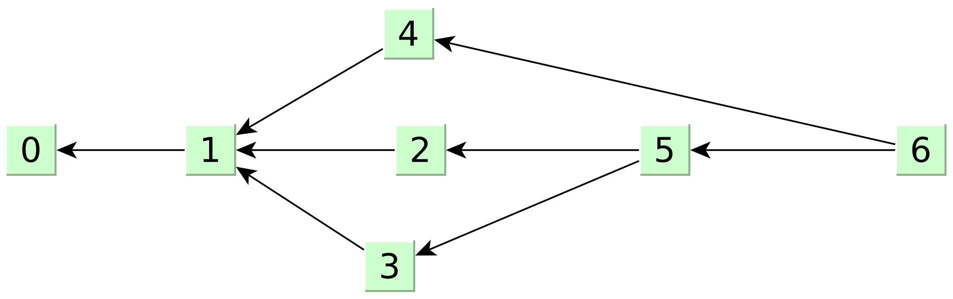


Figure A.1: *Tangle*

Strategy for choosing a tip is a challenging task; we get to know the arrival time of the transaction. Authors considered the distribution of transactions as Poison distribution. But if the transaction frequency is very low, then it will become linear, and if the frequency is very high, then there will be only one tip, which is genesis. The authors also discussed the advanced tip selection algorithm; this algorithm uses weighted random walks and cumulative weights on transactions. Advanced tip selection algorithms avoid lazy transactions to select old transactions in a tangle. This algorithm is still not the optimal solution for the tip selection; researchers are still working on it to improve it. All IOTAs are created at the genesis transactions; afterward, these IOTAs are sold to others, and a trading network is eventually established. Confirmation confidence is a measure to check tip acceptance by tangle to avoid double-spent. Its threshold is kept very high, up to 95%. But if a node has very high computational power, then it can double-spent. To avoid this coordinator is used to verify transactions till the network has some scalability level. The coordinator is used to verify each transaction and is removed from the network if the network reaches a safe state.

G-IOTA [12] modifies the weighted random walk tip selection algorithm to approve three ancestor nodes in tangle instead of two nodes to improve the fairness of transaction confidence for honest transactions. G-IOTA also proposed a mutual supervision mechanism instead of a coordinator to remove the centralization.

E-IOTA IOTA proposed a weighted random walk for tip selection. E-IOTA [13] converted this algorithm into the parameterized algorithm by defining two parameters, P1 and P2. A tip will select a random number to decide upon these parameters, and further, these parameterize are used to decide upon the tip selection algorithms. The tip selection algorithm varies from uniform random selection to high-weighted selection.

Graphchain

Graphchain [11] has a similar design to IOTA. A new transaction has to confirm its ancestor transaction to be part of DAG. It differs from IOTA in terms of incentive mechanisms. Transactions are required to post transaction fees as the offering for collection. Each transaction must refer to a group of ancestor transactions to deplete their fee when verifying the validity. Fees are depleted by an incoming transaction from the oldest ancestors in the path to reach a prescribed total amount.

Avalanche

Avalanche [79] protocol works in three stages: 1)Slush, 2)Snowflake, and 3)Snowball. In the first stage of slush, every node has three states: neutral, true, and false, differentiated by a different color. Initially, each node is neutral; then, if the node changes its color, other nearby nodes will adopt the same color. If they are neutral, the majority will have priority. Multiple rounds will be conducted with different random nodes until the majority of the network is on the same color. In the second stage, snowflake uses a counter for each node on the network. Every time counter will be increment if the color is unchanged after slush. If the counter reaches a very high number, it will be locked, and no more slush will affect its color. The third stage, snowball, adds memory elements to nodes to add a state of confidence, which is calculated based on past queries. When new transactions are validated by validator nodes, these transactions are added to the transaction pool, and a set of k random nodes(weighted by stake) are selected to approve the transaction using the above three stages, and the transaction is finally accepted if it reaches the certain threshold confidence.

Spectre

Spectre [88] protocol exploits the topological sorting property of DAG. PoW is used for mining, similar to the Nakamoto consensus. Pairwise ordering of blocks is done using a voting system to get the correct order of blocks. Pairwise ordering is required only when two blocks contain a conflicting transaction. If too many conflicting transactions are there, then the DAG will become a linear blockchain. Accordingly, the everyday number of transactions is increasing, which will lead to more block conflicts; thus, there is a scope for improvement in this protocol to decrease the number of conflicting blocks. Spectre is only suited to support cryptocurrency networks where strict ordering among transactions is unnecessary.

Phantom

Phantom [89] is the upgraded version of spectra. It enforces the strict ordering of blocks using a greedy approximation algorithm. Mining is the same as spectra; phantom protocol identifies the well-connected nodes and excludes the other blocks created by dishonest nodes. Parameter k is used for adjusting the tolerance level of concurrent transactions. The protocol aims to find maximum sub-DAGs with non-conflicting blocks.

Conflux

Conflux [48] is the improvement of ghost protocol. ghost protocol is used in Ethereum with the concept of uncle blocks. In these types of protocols, the first chain is allowed to diverge, but in the end, only one main chain is followed, and other miners of pruned blocks are provided with shared incentives. In conflux protocol, the adaptive weighted mechanism is used to assign weights on the edges based on the past subgraph. Sharding protocol also lies in this category. In sharding, the number of nodes in the network is divided into shards such that inter-shard communication is minimized and parallel; all shards will produce/mine new blocks. Mining protocols such as PoW and PoS can be used for intra-shard transactions, but the main challenge in sharding is to handle the inter-shard transaction. A separate committee-based protocol like PBFT is used to agree upon valid transactions.

Prism

Prism [8] is a protocol with parallel chains. This protocol consists of three types of blocks: proposal blocks, voter blocks, and transaction blocks. Voter blocks are used to elect the proposer block and specify the leader according to height. Proposer block will pack the transactions Block and extend the chain according to the longest chain rule.

References

[1] Ademar T. Akabane, Roger Immich, Richard W. Pazzi, Edmundo R. M. Madeira, and Leandro A. Villas. Distributed egocentric betweenness measure as a vehicle selection mechanism in VANETs: A performance evaluation study. *Sensors*, 18(8), 2018. [Pg.36]

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, Jan 2002. [Pg.61]

[3] Ahmed Alrehaili, Martin White, and N. Beloff. Systematic review of usage patterns and acceptance of blockchain-based cryptocurrencies across diverse domains. *Discover Analytics*, 2:11, Aug 2024. [Pg.16]

[4] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition, 2014. [Pg.2]

[5] Vadim Arasev. POA network whitepaper. `https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper`, Sep 2018. [Pg.1]

[6] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the Lightning: The game theory of payment channels. In *Financial Cryptography and Data Security*, pages 264–283. Springer, 2020. [Pg.112]

[7] Adam Back. Hashcash-a denial of service counter-measure. `http://www.hashcash.org/hashcash.pdf`, 2002. [Pg.11]

[8] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *ACM SIGSAC Conference on Computer and Communications Security*, page 585–602. ACM, 2019. [Pg.136]

[9] M. Basile, G. Nardini, P. Perazzo, and G. Dini. SegWit extension and improvement of the blocksim bitcoin simulator. In *IEEE International Conference on Blockchain*, pages 115–123, 2022. [Pg.3]

[10] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work, 2014. [Pg.15], [Pg.128]

[11] Xavier Boyen, Christopher Carr, and Thomas Haines. Graphchain: A blockchain-free scalable decentralised ledger. In 2nd *Workshop on Blockchains, Cryptocurrencies, and Contracts*, pages 21–33. ACM, 2018. [Pg.134]

[12] Gewu Bu, Önder Gürcan, and Maria Potop-Butucaru. G-IOTA: Fair and confidence aware tangle. In *IEEE Conference on Computer Communications Workshops (INFOCOM)*, pages 644–649. IEEE, 2019. [Pg.134]

[13] Gewu Bu, Wassim Hana, and Maria Potop-Butucaru. Metamorphic IOTA. `https://arxiv.org/abs/1907.03628`, 2019. [Pg.134]

[14] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. `https://arxiv.org/abs/1710.09437`, 2017. [Pg.15], [Pg.129]

[15] Martin Byrenheid, Thorsten Strufe, and Stefanie Roos. Attack Resistant Leader Election in Social Overlay Networks by Leveraging Local Voting. In 21st *International Conference on Distributed Computing and Networking*. ACM, 2020. [Pg.113]

[16] Yanjiao Chen, Yuyang Ran, Jingyue Zhou, Jian Zhang, and Xueluan Gong. MPCN-RP: A routing protocol for blockchain-based multi-charge payment channel networks. *IEEE Transactions on Network and Service Management*, 19(2):1229–1242, jun 2022. [Pg.24], [Pg.28], [Pg.31]

[17] Marco Conoscenti, Antonio Vetrò, and Juan Carlos De Martin. Hubs, rebalancing and service providers in the lightning network. *IEEE Access*, 7:132828–132840, 2019. [Pg.46], [Pg.62], [Pg.99], [Pg.114]

[18] Marco Conoscenti, Antonio Vetrò, and Juan Carlos De Martin. CLoTH: A lightning network simulator. *SoftwareX*, 15:100717, 2021. [Pg.58]

[19] Victor Costan. Intel SGX explained. IACR Cryptol, EPrint Archive `https://eprint.iacr.org/2016/086.pdf`, 2016. [Pg.132]

[20] Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security*, pages 23–41. Springer, 2019. [Pg.128]

[21] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology – EUROCRYPT*, pages 66–98. Springer International Publishing, 2018. [Pg.128]

[22] Raynor de Best. Number of cryptocurrencies worldwide. `https://www.statista.com/statistics/863917/number-crypto-coins-tokens/`, Dec 2024. [Pg.16]

[23] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *12th Annual International Cryptology Conference on Advances in Cryptology*, page 139–147. Springer-Verlag, 1992. [Pg.11]

[24] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19(2):248–264, April 1972. [Pg.22]

[25] Oğuzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to Profit from Payments Channels. In *Financial Cryptography and Data Security*, pages 284–303. Springer, 2020. [Pg.112]

[26] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5:691–703, 12 1976. [Pg.94]

[27] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. `https://eprint.iacr.org/2019/953`, 2019. [Pg.16], [Pg.131]

[28] Ankit Gangwal, Haripriya Ravali Gangavalli, and Apoorva Thirupathi. A survey of layer-two blockchain protocols. *Journal of Network and Computer Applications*, 209:103539, 2023. [Pg.17], [Pg.21]

[29] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015. [Pg.13]

[30] geeksforgeeks. What is ghost protocol for ethereum. `https://www.geeksforgeeks.org/what-is-ghost-protocol-for-ethereum/`, Sep 2024. [Pg.14]

[31] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. In *International Conference on Financial Cryptography and Data Security*, pages 439–457. Springer, 2018. [Pg.1]

[32] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *26th symposium on operating systems principles*, pages 51–68. Association for Computing Machinery, 2017. [Pg.15], [Pg.129]

[33] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. [Pg.21]

[34] Cyril Grunspan, Gabriel Lehéricy, and Ricardo Pérez-Marco. Ant routing scalability for the lightning network. `https://arxiv.org/abs/2002.01374`, 2020. [Pg.24], [Pg.29]

[35] Timo Hanke. AsicBoost - a speedup for bitcoin mining. `https://arxiv.org/pdf/1604.00575`, 2016. [Pg.3]

[36] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods*. University of California, 2005. Last Accessed: 02 Jul 2023. [Pg.36]

[37] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. On the Difficulty of Hiding the Balance of Lightning Network Channels. In *Conference on Computer and Communications Security*, page 602–612. ACM, 2019. [Pg.32], [Pg.114], [Pg.115]

[38] Tim Hornyak. Bitcoin xt debate overshadowing growth opportunities. `https://www.pcworld.com/article/423179/bitcoin-xt-debate-overshadowing-growth-opportunities.html`, Aug 2015. [Pg.14]

[39] Rein Houthooft, Sahel Sahhaf, Wouter Tavernier, Filip De Turck, Didier Colle, and Mario Pickavet. Robust geometric forest routing with tunable load balancing. In *IEEE Conference on Computer Communications*, pages 1382–1390, 2015. [Pg.72]

[40] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based onion routing. In 7^{th} *International Privacy Enhancing Technologies Symposium*, pages 95–112. Springer, 2007. [Pg.5], [Pg.39], [Pg.95], [Pg.113]

[41] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *ACM SIGSAC Conference on Computer and Communications Security*, page 439–453. ACM, 2017. [Pg.31]

[42] Rami Khalil, Arthur Gervais, and Guillaume Felley. Nocust-a securely scalable commit-chain. `https://api.semanticscholar.org/CorpusID:202619420`, 2018. [Pg.16], [Pg.130]

[43] Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. `https://eprint.iacr.org/2018/642`, 2018. [Pg.16], [Pg.130]

[44] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017. [Pg.128]

[45] Sunny King and Scott Nadal. Ppcoin peer-to-peer crypto-currency with proof-of-stake. `https://api.semanticscholar.org/CorpusID:42319203`, 2012. [Pg.15], [Pg.127]

[46] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *IEEE Symposium on Security and Privacy*, pages 583–598, 2018. [Pg.129], [Pg.130]

[47] Simon S. Lam and Chen Qian. Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. *IEEE/ACM Transactions on Networking*, 21(2):663–677, 2013. [Pg.27]

[48] Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. Scaling nakamoto consensus to thousands of transactions per second. `https://arxiv.org/abs/1805.03870`, 2018. [Pg.135]

[49] Lightning network statistics. `https://bitcoinvisuals.com/lightning`, March 2020. [Pg.6], [Pg.36], [Pg.73]

[50] Lightning-Network-Tools. Channel fees. `https://docs.lightning.engineering/lightning-network-tools/lnd/channel-fees`, Feb 2022. [Pg.66]

[51] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using trusted execution environments. `http://arxiv.org/abs/1612.07766`, 2016. [Pg.132]

[52] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In 27th *ACM Symposium on Operating Systems Principles*, page 63–79. ACM, 2019. [Pg.133]

[53] A. Lisi, D. Francesco Maesa, P. Mori, and L. Ricci. Lightnings over rose bouquets: an analysis of the topology of the bitcoin lightning network. In 45th *Annual Computers, Software, and Applications Conference*, pages 324–331. IEEE Computer Society, jul 2021. [Pg.85]

[54] Yizhong Liu, Jianwei Liu, Marcos Antonio Vaz Salles, Zongyang Zhang, Tong Li, Bin Hu, Fritz Henglein, and Rongxing Lu. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *Computer Science Review*, 46:100513, 2022. [Pg.129]

[55] David Lobmaier, Rafael Konlechner, Stefan Schulte, and Ingo Weber. Assessing routing algorithms for payment channel networks. *Distrib. Ledger Technol.*, 3(1), March 2024. [Pg.22]

[56] Zhichun Lu, Runchao Han, and Jiangshan Yu. General Congestion Attack on HTLC-Based Payment Channel Networks. `https://eprint.iacr.org/2020/456`, 2020. [Pg.32]

[57] Xiaofei Luo and Peng Li. Learning-based off-chain transaction scheduling in prioritized payment channel networks. *IEEE Journal on Selected Areas in Communications*, 40(12):3589–3599, 2022. [Pg.6], [Pg.28], [Pg.31]

[58] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. `https://eprint.iacr.org/2015/1168`, 2015. [Pg.15], [Pg.129]

[59] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *ACM SIGSAC Conference on Computer and Communications Security*, page 17–30. ACM, 2016. [Pg.129]

[60] Nancy A Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers Inc., 1996. [Pg.36]

[61] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. `https://eprint.iacr.org/2016/1054.pdf`, 2016. [Pg.6], [Pg.25], [Pg.28], [Pg.30], [Pg.34], [Pg.50], [Pg.54], [Pg.58], [Pg.64], [Pg.72], [Pg.113]

[62] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *ACM SIGSAC Conference on Computer and Communications Security*, page 455–471. Association for Computing Machinery, 2017. [Pg.30], [Pg.97]

[63] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. `https://eprint.iacr.org/2018/472`, 2018. [Pg.32], [Pg.108]

[64] Stefano Martinazzi. The evolution of lightning network's topology during its first year and the influence over its core values. `http://arxiv.org/abs/1902.07307`, 2019. [Pg.46], [Pg.48], [Pg.58], [Pg.59], [Pg.85]

[65] Ayelet Mizrahi and Aviv Zohar. Congestion Attacks in Payment Channel Networks. In *Financial Cryptography and Data Security*, pages 170–188. Springer, 2021. [Pg.32]

[66] Daniel Platt Moritz Platt and Peter McBurney. Sybil attack vulnerability trilemma. *International Journal of Parallel, Emergent and Distributed Systems*, 39(4):446–460, 2024. [Pg.13]

[67] Satoshi Nakamoto. Bitcoin a peer-to-peer electronic cash system whitepaper. `https://bitcoin.org/bitcoin.pdf`, 2009. [Pg.1], [Pg.2], [Pg.11]

[68] Jianyu Niu, Ziyu Wang, Fangyu Gai, and Chen Feng. Incentive analysis of bitcoin-ng, revisited. *SIGMETRICS Performance Evaluation Review*, 48(3):59–60, March 2021. [Pg.14]

[69] Asuman E. Ozdaglar and Dimitri P. Bertsekas. Optimal solution of integer multicommodity flow problems with application in optical networks. In *Frontiers in Global Optimization*, pages 411–435. Springer, 2004. [Pg.106]

[70] Nikolaos Papadis and Leandros Tassiulas. Blockchain-based payment channel networks: Challenges and recent advances. *IEEE Access*, 8:227596–227609, 2020. [Pg.3]

[71] Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In *ACM Symposium on Principles of Distributed Computing*, page 315–324. Association for Computing Machinery, 2017. [Pg.15], [Pg.129]

[72] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *23rd International Conference on the Theory and Applications of Cryptology and Information*, pages 380–409. Springer, 2017. [Pg.128]

[73] Cristina Pérez-Solà, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. LockDown: Balance Availability Attack Against Lightning Network Channels. In 24th *International Conference Financial Cryptography and Data Security*, page 245–263. Springer-Verlag, 2020. [Pg.32]

[74] Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. `https://arxiv.org/abs/1912.09555`, 2019. [Pg.23], [Pg.31], [Pg.90]

[75] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts white paper. `https://plasma.io/plasma.pdf`, 2017. [Pg.16], [Pg.130]

[76] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. [Pg.4], [Pg.5], [Pg.6], [Pg.17], [Pg.20]

[77] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2016. [Pg.5], [Pg.31]

[78] Gabriel Antonio F. Rebello, Gustavo F. Camilo, Maria Potop-Butucaru, Miguel Elias M. Campista, Marcelo Dias de Amorim, and Luís Henrique M. K. Costa. PCNsim: A flexible and modular simulator for payment channel networks. In *IEEE Conference on Computer Communications Workshops*, pages 1–2, Seoul, South Korea, 2022. [Pg.58]

[79] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. `https://arxiv.org/abs/1906.08936`, 2019. [Pg.134]

[80] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. `http://arxiv.org/abs/1904.10253`, 2019. [Pg.113]

[81] Stefanie Roos, Martin Beck, and Thorsten Strufe. Voute-virtual overlays using tree embeddings, 2016. [Pg.29], [Pg.37]

[82] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. `https://arxiv.org/pdf/1709.05748`, 2017. [Pg.6], [Pg.26], [Pg.28], [Pg.29], [Pg.30], [Pg.34], [Pg.50], [Pg.54], [Pg.64], [Pg.72], [Pg.83], [Pg.89], [Pg.98], [Pg.113]

[83] Abdurrashid Ibrahim Sanka and Ray C.C. Cheung. A systematic review of blockchain scalability: Issues, solutions, analysis and future research. *Journal of Network and Computer Applications*, 195:103232, 2021. [Pg.6]

[84] Sawtooth. Hyperledger sawtooth project. `https://github.com/hyperledger/sawtooth-core`, Dec 2018. [Pg.1]

[85] István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. Topological analysis of bitcoin's lightning network. In *Mathematical Research for Blockchain Economy*, pages 1–12. Springer, 2020. [Pg.36], [Pg.48], [Pg.58], [Pg.62], [Pg.85], [Pg.99], [Pg.114]

[86] 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 17^{th} *USENIX Symposium on Networked Systems Design and Implementation*, pages 777–796. USENIX Association, February 2020. [Pg.23]

[87] Slickcharts. Cryptocurrency market data. `https://www.slickcharts.com/currency`, Dec 2024. [Pg.16]

[88] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. `https://ia.cr/2016/1159`, 2016. [Pg.135]

[89] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom ghostdag: A scalable generalization of nakamoto consensus: September 2, 2021. In 3^{rd} *ACM Conference on Advances in Financial Technologies*, page 57–70. ACM, 2021. [Pg.135]

[90] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015. [Pg.14]

[91] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy*, pages 526–545. IEEE Computer Society, 2016. [Pg.129]

[92] Shuyang Tang. Bracing a transaction dag with a backbone chain. `https://ia.cr/2020/472`, 2020. [Pg.133]

[93] CryptoEQ CORE Team. Bitcoin cash report. `https://www.cryptoeq.io/corereports/bitcoin-cash-abridged`, May 2021. [Pg.2], [Pg.14]

[94] The Raiden Team. Raiden network. `https://raiden.network/`, March 2021. [Pg.4], [Pg.18]

[95] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. `http://arxiv.org/abs/1908.04756`, 2019. [Pg.132]

[96] Saar Tochner, Stefan Schmid, and Aviv Zohar. Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff. `http://arxiv.org/abs/1909.06890`, 2019. [Pg.32], [Pg.115]

[97] Gijs van Dam, Rabiah Abdul Kadir, Puteri N. E. Nohuddin, and Halimah Badioze Zaman. Improvements of the Balance Discovery Attack on Lightning Network Payment Channels. `https://eprint.iacr.org/2019/1385`, 2019. [Pg.32]

[98] Jan van den Brand and Daniel Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. `https://api.semanticscholar.org/CorpusID:258309586`, 2023. [Pg.21]

[99] Sushil Mahavir Varma and Siva Theja Maguluri. Throughput optimal routing in blockchain-based payment systems. *IEEE Transactions on Control of Network Systems*, 8(4):1859–1868, 2021. [Pg.6], [Pg.28], [Pg.31]

[100] Bimal Viswanath, Mainack Mondal, Krishna P. Gummadi, Alan Mislove, and Ansley Post. Canal: scaling social network-based sybil tolerance schemes. In 7th *ACM European Conference on Computer Systems*, page 309–322. ACM, 2012. [Pg.25]

[101] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 95–112. USENIX Association, February 2019. [Pg.129]

[102] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: Efficient dynamic routing for offchain networks. `http://arxiv.org/abs/1902.05260`, 2019. [Pg.22]

[103] Shira Werman and Aviv Zohar. Avoiding deadlocks in payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 175–187. Springer, 2018. [Pg.30], [Pg.87], [Pg.90], [Pg.94], [Pg.97], [Pg.103], [Pg.106]

[104] Jie Wu and Suhan Jiang. On increasing scalability and liquidation of lightning networks for blockchains. *IEEE Transactions on Network Science and Engineering*, 9(4):2589–2600, 2022. [Pg.6], [Pg.28]

[105] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020. [Pg.2], [Pg.15], [Pg.66], [Pg.127], [Pg.129]

[106] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. Coin-Express: A fast payment routing mechanism in blockchain-based payment channel networks. In 27^{th} *International Conference on Computer Communication and Networks*, pages 1–9, 2018. [Pg.27]

[107] Philipp Zabka, Klaus-T. Foerster, Stefan Schmid, and Christian Decker. Empirical evaluation of nodes and channels of the lightning network. *Pervasive and Mobile Computing*, 83:101584, 2022. [Pg.53], [Pg.54], [Pg.58], [Pg.86], [Pg.119], [Pg.123]

[108] Philipp Zabka, Klaus-Tycho Foerster, Christian Decker, and Stefan Schmid. *A Centrality Analysis of the Lightning Network*, pages 374–385. Springer-Verlag, 10 2022. [Pg.35], [Pg.59], [Pg.95]

[109] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *ACM SIGSAC Conference on Computer and Communications Security*, page 931–948. ACM, 2018. [Pg.129]

[110] Mengqian Zhang, Jichen Li, Zhaohua Chen, Hongyin Chen, and Xiaotie Deng. Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding. In *IEEE International Parallel and Distributed Processing Symposium*, pages 358–367, 2020. [Pg.129], [Pg.130]

[111] Xiaoxue Zhang, Shouqian Shi, and Chen Qian. WebFlow: Scalable and decentralized routing for payment channel networks with high resource utilization. `https://arxiv.org/pdf/2109.11665`, 2021. [Pg.6], [Pg.27], [Pg.28], [Pg.83], [Pg.98]

[112] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *IEEE International Conference on Communications*, pages 1–6, 2019. [Pg.23]

Dissertation Publications

Journals

1. Neeraj Sharma and Kalpesh Kapoor, maxREE: maximizing flow by replacing exhausted edges in lightning network, Transactions on Network Science and Engineering (TNSE), 2025 (accepted)

2. Neeraj Sharma, Kalpesh Kapoor, Deadlock Prevention in Payment Channel Networks, IEEE Transactions on Network and Service Management (TNSM), July 2024, IEEE.

3. Neeraj Sharma, Kalpesh Kapoor, and V Anirudh, Design and Evaluation of Swift Routing for Payment Channel Network, Blockchain: Research and Applications (BCRA), Dec 2023, Elsevier.

Conferences

1. Neeraj Sharma and Kalpesh Kapoor, Distributed Routing Algorithms for Concurrent Execution of Transactions in PCNs, IEEE International Conference on Fog and Edge Computing (ICFEC), May 2023.

2. Neeraj Sharma and Kalpesh Kapoor, Attacks in Distributed Routing in PCNs, IEEE International Conference on Blockchain (ICB)" Aug 2024

3. Neeraj Sharma and Kalpesh Kapoor, Flood and Loot Attack in Distributed Routing Protocols in PCNs, 17th International Conference on COMmunication Systems & NETworkS (COMSNETS)" Jan 2025 (accepted)

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati 781039, India