BLN: A LAYER-2 FRAMEWORK FOR FAST BITCOIN TRANSACTION PROCESSING

by

Yahu Wang

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2023

Table of Contents

List of Tables

List of Figures

Abstract

Over the past few years, cryptocurrency has evolved from a virtual concept to a digital currency that can be used for payments. Bitcoin, the world's largest cryptocurrency, was created in 2009. It is expected that an increasing number of business transactions will involve Bitcoin. Technically, Bitcoin uses blockchain as its distributed ledger. Periodically, a group of business transactions is packed into a block, which is added to the blockchain for verification purposes. However, Bitcoin has two inherent limitations that affect its scalability. First, the original size of a block is at most 1 megabyte. Second, on average, a new block is generated once every ten minutes. With these two limitations, Bitcoin cannot handle too many transactions in a short period. To address the scalability problem with Bitcoin, a series of alternative schemes have been proposed. Lightning Network (LN) is a layer-2 protocol that enables fast Bitcoin transaction processing. However, it only works well for peer-to-peer micropayments, namely, small-amount payments between individual parties. In this thesis, we propose an LN-based framework for both peer-to-peer and customer-to-business payments, the Business-oriented Layer-2 Network (BLN). With BLN, Bitcoin transaction processing scales much better. Our experimental results indicate that BLN outperforms LN in terms of transaction failure rate, transaction fee, and processing time.

Acknowledgements

Thank you, Dr. Qiang Ye, for your time and your patient guidance.

Chapter 1

Introduction

The purpose of this thesis is to propose a new trading model based on the existing structure of Bitcoin and the Lightning Network, and to apply the revised function of the Redis cluster to improve transaction efficiency in this trading model. In this chapter, we will provide a general description of some Bitcoin and Redis issues, and then we will begin by presenting the motivation and outline of this thesis.

To avoid confusion, Table 1.1 has been provided here, which contains important abbreviations that will be frequently used throughout the thesis.

Table 1.1: Table of Abbreviations

Full Name	**Abbreviations**
Lightning Network	LN
Business-Oriented Layer-2 Network	BLN
Transaction Processing Center	TPC
Hash Time Lock Contract	HTLC

1.1 Scalability of Bitcoin

The concept of Bitcoin [2] was first proposed by Satoshi Nakamoto on November 1, 2008, and was officially launched on January 3, 2009. Unlike most currencies, Bitcoin is not issued by a specific monetary institution but is produced through a large number of calculations based on a specific algorithm. The Bitcoin economy uses a distributed database composed of many nodes in the entire P2P network to confirm and record all transactions, and uses cryptographic design to ensure the security of every link in the currency circulation. The decentralized nature of P2P [3], along with the algorithm itself, ensures that the value of the currency cannot be artificially manipulated by churning out Bitcoins. The cryptographic-based design allows Bitcoins to be transferred or paid only by the real owner, which ensures the

anonymity of money ownership and circulation transactions. Meanwhile, the total number of Bitcoins is limited. The currency system was capped at 10.5 million for four years, after which the total number will be permanently capped at 21 million.

The reason why the blockchain based on the Bitcoin network has poor scalability [4] is that the block size is limited to 1 MB, and the block generation speed is limited to 10 minutes per block. It cannot support transactions in real life where thousands of transactions will be created within a second. Therefore, the idea of the Lightning Network [5] has appeared. Traditionally, once a transaction is created, it will be broadcast to everyone. The miners in the Bitcoin network will collect this transaction in their transaction pool [6], and then they will pack many transactions into one block. If the block records on the blockchain, then all the transactions in that block are finalized. In order to show the advantage of the Lightning Network, we will take Alice buying coffee from Bob as an example. Specifically, Alice will buy a cup of coffee during her working days. So from Monday to Friday, there will be five different transactions in total, and they will all be broadcast to the network. But the thing is, one cup of coffee is not worth too much, and most people will not care about it. And they become five different transactions. If they are all recorded in one block, they not only cost too much transaction fees but also waste too much space on the block.

Therefore, the core idea of the Lightning Network is that we don't need to let everyone know the details of all transactions. Instead, we make a deal privately. In Alice and Bob's example, they can merge five transactions into one. That means both of them achieve an agreement that Alice buys five cups of coffee from Bob. The time when Alice buys the coffee does not matter. Alice could buy all of the coffee at the same time or buy the coffee at different times. For others, they don't care about when Alice buys the coffee. The only thing that matters is that Alice buys the coffee from Bob. In a nutshell, the five transactions of Alice buying coffee from Monday to Friday merge into one transaction that Alice buys five cups of coffee from Bob, and we don't know and we don't care about when Alice buys the coffee. This method reduces the waste of resources on the block in the Bitcoin network and also saves the transaction fee for Alice.

1.2 Revised Redis for Fast Processing

Before discussing how to revise the Redis cluster, let's briefly describe the basic functions of Redis. Redis [7] is a server-based database structured with key-value pairs, supporting various data structures such as strings, lists, hashes, sets, and ordered sets. To ensure efficient reading, Redis stores data objects in memory and periodically writes updated data to disk files.

In the case of applications, Redis is often used as a data cache because of its high efficiency in reading and writing since data is accessed and operated on in memory. Some data that needs to be frequently accessed and will not change in a short time can be put into Redis for operation. If we want to improve the speed of user requests and reduce the load on the website, as well as reduce the number of database reads and writes, we can put this data into the cache. Another commonly used function is real-time computing. If you need the ability to change and display data in real-time, you can put the relevant data in Redis for manipulation, which greatly improves efficiency.

The important structure of Redis can be divided into a master node and a slave node. The master node stores all master data and performs all read and write operations for the client. To prevent the master node from being out of service due to an unexpected failure, the slave node stores a replica of all master data. When the master node fails unexpectedly, the slave node can be used as an alternative to help process the client request.

By default, the master node is responsible for all read and write operations from the client and the slave node only works as a backup, but we can let the slave node undertake part of the read or write operation by changing the configuration. However, data synchronization between the master node and slave node becomes a challenge. If the master node and slave node change the same data at the same time, data on the master node and slave node may be inconsistent. So, most read and write operations happen only on the master node. Even so, this can lead to another problem: under high pressure conditions, the master node has to undertake too much pressure, resulting in a decline in service efficiency. Now, Redis cluster has been created, which

means multiple Redis singletons cooperate with each other. Doing so takes some of the pressure off the master node and improves scalability. But there is still no solution if the high-pressure request is concentrated on a Redis singleton.

As our proposed trading model requires faster processing efficiency, we have revised some of the configurations in the Redis cluster. In a later chapter, we will explain in more detail how to optimize the Redis cluster for faster processing.

1.3 Thesis Motivation and Contribution

The purpose of this thesis is to propose a new trading model based on the existing structure of Bitcoin and the Lightning Network. We want to solve the problem that the Lightning Network does not support large transactions and the inequality between the two parties in transactions. Additionally, we also aim to extend the transaction scenario to online shopping, rather than just in-store purchases.

We propose a structure called the Transaction Processing Center (TPC) to act as a broker and supervisor for all roles involved in the purchase scenario. This structure ensures the proper rights of each role and imposes reasonable penalties when a role makes a mistake. Additionally, the TPC is user-friendly, helps to support large transactions, and saves transaction fees for users.

To improve the TPC's reliability, maintain stability and correctness under high pressure of mass transaction requests, we also propose revising the function of the Redis cluster. This involves storing master data evenly across all nodes, rather than only in master nodes and replicas in slave nodes. This reduces stress on one node in the case of a large number of requests.

1.4 Thesis Outline

The rest of the thesis is organized in the following manner. In the next chapter, we will introduce the basic functions of Bitcoin, the Lightning Network, and Redis. After analyzing the disadvantages of these functions, we will propose a new solution called the Business-Oriented Layer-2 Network (BLN). After explaining how BLN works, a series of experiments will be conducted to compare the performance of the Lightning

Network and BLN in terms of transaction failure rate and transaction fee. We will explain in detail why they behave differently.

Chapter 2

Related Work

In previous chapters, we provided an overview of Bitcoin, Lightning Network, and Redis. In this chapter, we will delve into these topics in more detail to provide a better understanding. We will also provide practical examples to illustrate their characteristics.

2.1 Bitcoin

2.1.1 Bitcoin Transaction

The components of a transaction are described in Table 2.1. To better understand, Alice's transaction of buying coffee at Bob's cafe is shown in Fig. 2.1 below. The most important part of a transaction is its output. When a new transaction is created, its output is called an unspent transaction output (UTXO) [8] [9], which means that no one has spent the money yet. Take a simple example: Alice creates a transaction to buy a coffee from Bob which costs her 1 bitcoin. When the transaction is recorded on the blockchain, there will be one unspent transaction output worth one bitcoin for Bob. Then Bob wants to spend his bitcoin, so he cites the previously mentioned Alice transaction output as an input for his new transaction. At the same time, because Bob cites Alice's transaction output, that transaction output is no longer "unspent". Therefore, we can see that all transactions on the blockchain are linked together by inputs and outputs so we can easily track whole transactions along the blockchain. If no one cites one transaction's output, that means the transaction is complete, and the owner can spend it to create another transaction.

Meanwhile, the output and input of a bitcoin transaction are not fixed. For example, if Bob receives many bitcoins from many people, he can merge all unspent transaction output in one transaction input like Fig. 2.2, which is named aggregating transaction.

Table 2.1: The Structure of a Transaction [1]

Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1-9 bytes(VarInt)	Input Counter	How many inputs are included
Variable	Inputs	One or more transaction inputs
1-9 bytes(VarInt)	Output Counter	How many outputs are included
Variable	Outputs	One or more transaction outputs
4 bytes	Locktime	A Unix timestamp or block number

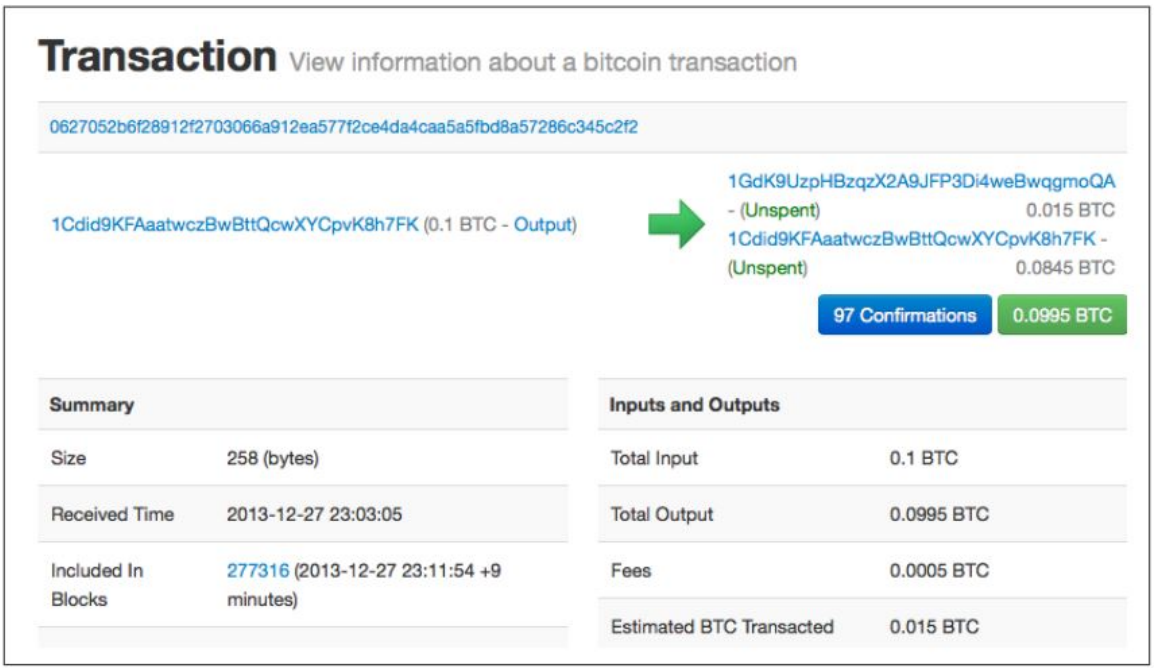


Figure 2.1: A Transaction Involving Alice and Bob's Cafe [1]

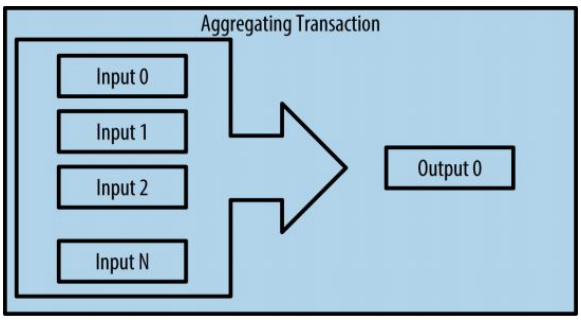
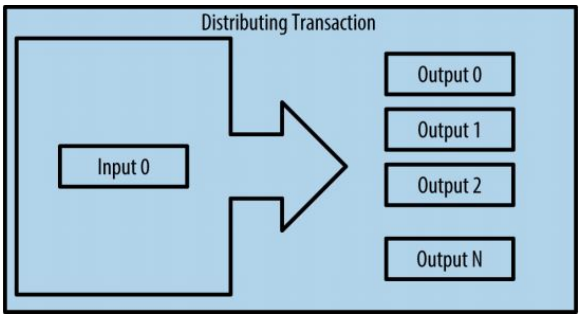


Figure 2.2: Aggregating Transaction [1]

Alternatively, Bob can distribute the output to a different bitcoin address, as shown in Fig. 2.3. This type of transaction is called a distributing transaction, which can be used to pay the transaction fee to the miner in the blockchain network while buying something from a third party.



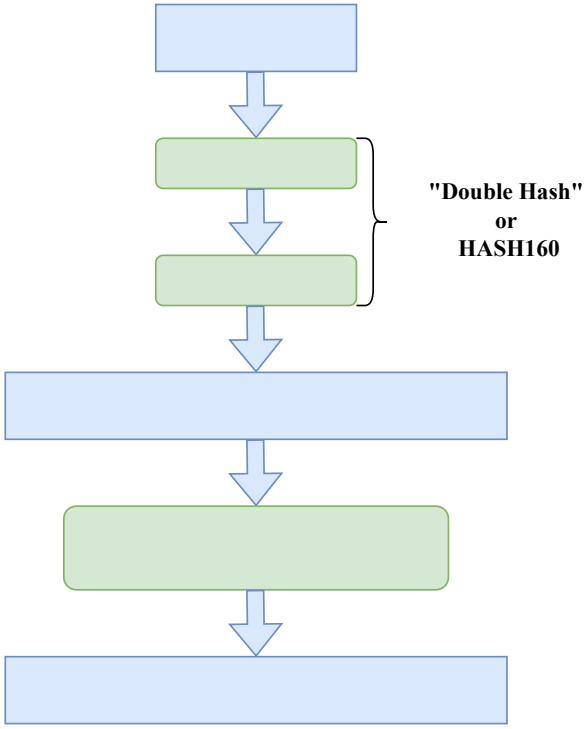


Figure 2.4: Conversion of a Public Key into a Bitcoin Address

Bitcoin address [10] [11] [12] is another important core element of the Bitcoin network. Basically, all transactions depend on a Bitcoin address, which also serves as a personal wallet. Generally, if someone wants to create a wallet, they must create their own private key using the underlying system's random number generator, which generates a 256-bit binary number [13] [14]. The private key will be written as a 64-bit hexadecimal number. Of course, the person cannot let anyone know what the private key is, otherwise the bitcoins in the wallet will be stolen. Then, the person must generate the public key by calculating elliptic curve cryptography, which results in a point (x,y) on an elliptic curve. The benefit of this is that the direction is irreversible. The public key can only be derived from the private key, while the private key cannot be derived from the public key. Finally, a Bitcoin address is generated by the base58Check Encode [15] result of the public key hash value. The specific process of [illegible] is shown

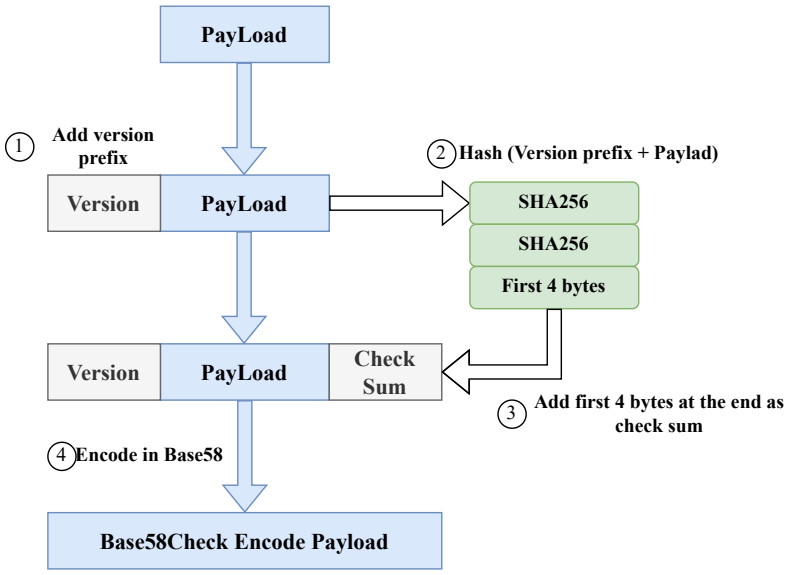


Figure 2.5: Base58CheckEncode Payload

2.1.3 Script in Bitcoin Transaction

To better understand transactions in the bitcoin network, it's important to discuss the essence of their inputs and outputs. Both the input and output are scripts [16],

with the input being the unlock script and the output being the lock script. To illustrate this concept, let's consider the scenario of Alice buying coffee from Bob. When Alice creates her transaction, the transaction output is the lock script, which means that the money is "locked" by Alice and only the one who can "unlock" it has the right to spend the money.

Before delving deeper into this topic, let's first discuss the two main payment methods in the bitcoin network. The first method is Pay to Public Key Hash (P2PKH) [17] [18], which means that we pay the money to a specific bitcoin address. The second method is Pay to Script Hash (P2SH) [19] [20], which includes an additional component called the redeem script, allowing for the payment to be redeemed. We will explore this concept further later on, but for now, let's return to our Alice and Bob example. Before Alice creates the transaction, Bob must provide his personal bitcoin address, which we will refer to as his "wallet" for clarity in the following discussion. Of course, Alice cannot use the money inside because only Bob has access to the private key. Alice then uses Bob's public key address to generate the lock script, which contains a series of operations. In our example, the lock script would look like this:

OP-DUP OP-HASH160 [Bob Public Key Hash] OP-EQUAL OP-CHECKSIG

Meanwhile, when Bob wants to spend the money from that transaction output, in other words, the lock script, Bob should also provide his own unlock script, which will look like:

[Bob Signature] [Bob Public Key]

As soon as Bob spends the money from Alice's transaction, the two scripts will merge together and the operation order is:

[Bob Signature] [Bob Public Key] OP-DUP OP-HASH160 [Bob Public Key Hash] OP-EQUAL OP-CHECKSIG

The s

through

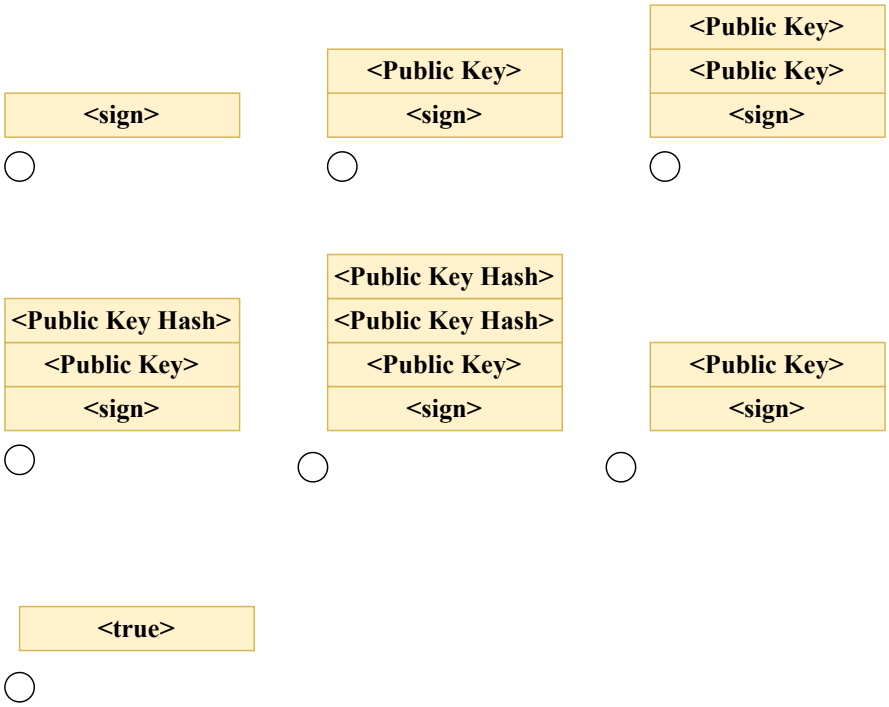


Figure 2.6: Operation Stack

In step 1 and step 2, the script pushes Bob's signature and Bob's Public Key into the stack. Bob's unlock script ends here.

In step 3, OP-DUP duplicates the content at the top of the stack, which is Bob's Public Key.

In step 4, OP-HASH160 calculates the hash result of the top of the stack, and we get Bob's Public Key Hash at the top, which is also Bob's bitcoin address.

In step 5, the script pushes the Bob Public Key Hash that Alice received from Bob.

In step 6, OP-EQUAL checks if the top two contents in the stack are the same. If everything is correct, they must be the same, and the stack will pop both out.

In the final step, the system will check if the digital signature belongs to Bob. Bob's digital signature is generated by his private key through encryption. A public key is required for signature verification. Therefore, Bob cannot let anyone know his

private key because someone can create a fake unlock script and steal the money from Alice's transaction, which is recorded on the blockchain and visible to everyone. If Bob's digital signature is valid, the stack will return "true," and the transaction is valid, allowing Bob to spend the money. This is how a person in the bitcoin network can make a payment to another person.

2.2 Lightning Network

2.2.1 Multi-signature Address

Multi-signature scripts [22] [23] [24] impose a condition where a lock script contains N public keys and at least M of those keys must provide signatures to release the encumbrance. In simple terms, N people share the same wallet, and the wallet requires the agreement of at least M people to open and use the money inside. The lock script of the M-of-N multi-signature address will take the following form:

M [public key 1] [public key 2] ...[public key N] N OP-CHECKMULTISIG

A lock script for a 2-of-4 multi-signature address would look like this:

2 [public key 1] [public key 2] ...[public key 4] 4 OP-CHECKMULTISIG

That means there are four public keys in total and the unlock script is required to provide at least two signatures. So the unlock script will be like this:

OP-0 [signature 1] [signature 2]

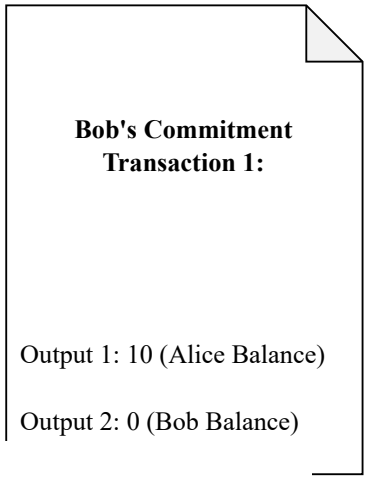
OP-0 means to push an empty array onto the stack. Normally, a multi-signature address is used for multi-party cooperation. In the case of the Lightning Network, a 2-of-2 multi-signature address is used as a wallet shared between two people. The only way to open the wallet is if both parties come to an agreement.

2.2.2 Funding Transaction and Commitment Transaction

In this section, we will continue using the example of Alice buying coffee from Bob to explain how the Lightning Network works. If they decide to use the Lightning Network, they will both use their own public keys to set up a multi-signature address. In order to make the transaction work, they have to put money inside, which is called the funding transaction [25] [26]. Let's say Alice wants to put 10 bitcoins inside the wallet. She will create a funding transaction and broadcast it. Once the transaction is recorded on the blockchain, Alice will have her own 10 bitcoins in this wallet. It should be noted that although Alice and Bob share the wallet, it does not mean they share all the money inside. We can consider there is a balance in this wallet. Because Alice funded 10 bitcoins in the wallet, the current balance between Alice and Bob is 10 to 0.

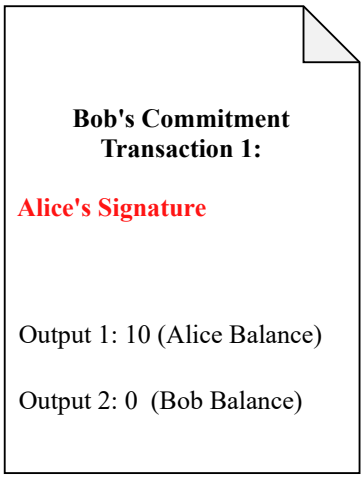
When Alice wants to buy coffee from Bob, the balance is changed through a commitment transaction [25] [26]. The commitment transaction changes the record of the balance between Alice and Bob. However, the commitment transaction must contain the signature from both sides, otherwise, it is not valid. After the funding is done, Alice and Bob create their own commitment transaction, like Fig. 2.7, which has two outputs for Alice and Bob separately.

In Fig. 2.7, both commitment transactions have no signatures, so they are both invalid. Therefore, Alice and Bob switch their commitment transactions and check if the balance is good for them. If it is, they will sign their opponent's commitment transaction like in Fig. 2.8. After both receive their opponent's signature, they can sign their own commitment transactions. Then both commitment transactions will have two signatures, making them valid.



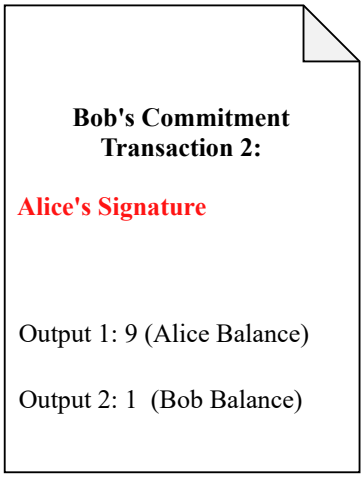
action 1

ignature



(b) Bob Commitment Transaction 1 with Alice Signature

Figure 2.8: Alice and Bob Commitment Transaction 1 with Opponent's Signature



(b) Bob Commitment Transaction 2 with New Balance and Signature

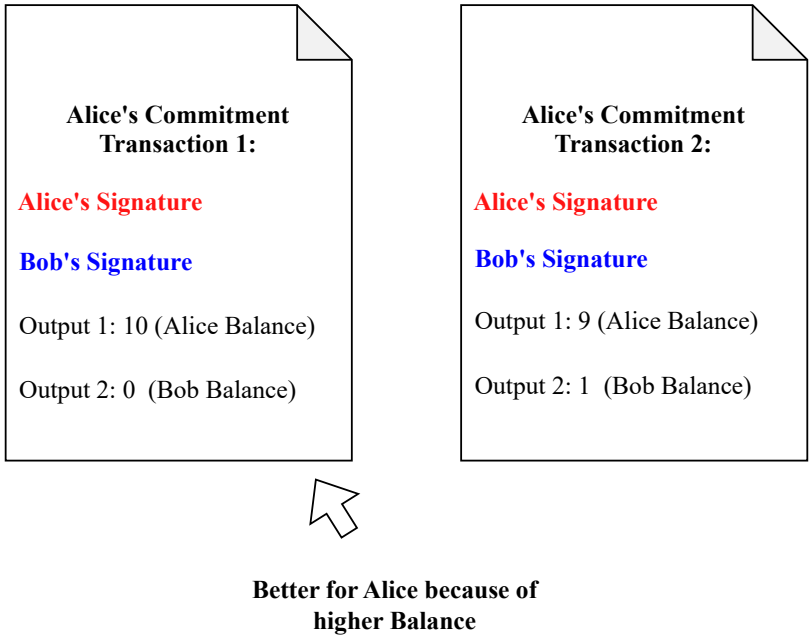


Figure 2.10: Alice's Valid Commitment Transactions

Next, Alice wants to buy a cup of coffee that costs 1 bitcoin. So the balance between Alice and Bob must be changed. In this case, Alice and Bob create a new commitment transaction on their own, and the new balance becomes 9 to 1. Then, they repeat the previous step to exchange the signature in Fig. 2.9.

However, as previously mentioned, a commitment transaction becomes valid as soon as it has two signatures. So for Alice, her commitment transaction 1 is still valid, and the original balance is 10 to 0 in Fig. 2.10, which is good for her. If she does not broadcast the latest commitment transaction, she can steal the money from Bob.

So we need punishments and restrictions that force both sides to only broadcast the latest commitment transaction. The Lightning Network adds a new structure called a Revocable Sequence Maturity Contract (RSMC) [27] [28]. In simple terms, this structure creates two paths for the output of commitment transactions. If one side broadcasts their commitment transaction, the other side can get their funds immediately according to the record in the commitment transaction. And the one who broadcasts the commitment transaction must wait until the confirmation is done. In the Lightning Network script, it requires waiting for 1,000 confirmations in total. Confirmation means how many blocks after the record exists on the blockchain. For example, if the commitment transaction that has been broadcast is recorded on Block 1, and the end of the blockchain is Block 100, we say that the commitment transaction has 99 confirmations because there are already 99 blocks after Block 1 which contains the commitment transaction. Meanwhile, there is a revocation key for each commitment transaction. We use Alice's commitment transaction 1 as an example, and some details of Alice commitment transaction with RSMC are shown in Fig. 2.11. When Alice and Bob switch their signature, they also switch the revocation key. If Alice really broadcasts the commitment transaction 1, output 2 will send funds to Bob immediately, while output 1 is pending confirmation. As we mentioned before, Alice already gave her commitment transaction 1 revocation key to Bob. There are only two cases:

Case 1: Confirmation is done, and Bob does nothing; output 1 sends the funds to Alice.

Case 2: Bob noticed that Alice did not broadcast the latest commitment transaction, so he broadcasts the latest version of the commitment transaction and cites Alice's commitment transaction 1 with the revocation key that he previously got from Alice. As a punishment, all funds in output 1 will be sent to Bob. Summing up with

output 2, all money in this wallet will belong to Bob.

In the Lightning Network sample script, it requires 1,000 confirmations for one commitment transaction. The generation speed of a block is 10 minutes. So it costs 10,000 minutes for the confirmation. The reason why it takes that long is that the confirmation time is just long enough for Bob to notice if Alice follows the rule. And, of course, if Bob does not notice before the confirmation is done, he will lose the money that should belong to him. The detailed structure is shown in Fig. 2.11.

Now let's summarize the trading sequence. Before Alice broadcasts her funding transaction, she needs to create her commitment transaction 1 and get Bob's signature. If Bob disappears after Alice broadcasts the funding transaction, her money will be locked in the multi-signature address because she cannot get Bob's signature. Then Alice wants to buy the coffee, so they both create a new commitment transaction 2 to update the balance. To sum up, the following process order is:

1.Alice signs Bob's commitment transaction 2.

2.Bob gives Alice his commitment transaction 1 revocation key.

3.Bob signs Alice's commitment transaction 2.

4.Alice gives Bob her commitment transaction 1 revocation key.

5.Bob gives Alice the coffee.

It should be noted that Alice should sign before Bob. If Bob signs first and gives his revocation key to Alice, it would be bad for him. Because if Alice disappears, Bob has already given out his revocation key for commitment transaction 1, and he does not have Alice's signature on his commitment transaction 2 yet. So he cannot broadcast either of the commitment transactions, and his money will be locked in this multi-signature address. Therefore, Alice must sign Bob's commitment transaction 2 first to ensure that Bob has a valid commitment transaction to broadcast. Additionally, Bob still needs to hold on to the coffee until Alice gives her commitment transaction 1 revocation key. Otherwise, Bob cannot punish Alice if she broadcasts the commitment transaction 1. After both sides switch commitment transaction 1 revocation key and signature for commitment transaction 2, Bob gives the coffee to Alice, and the trade is complete.

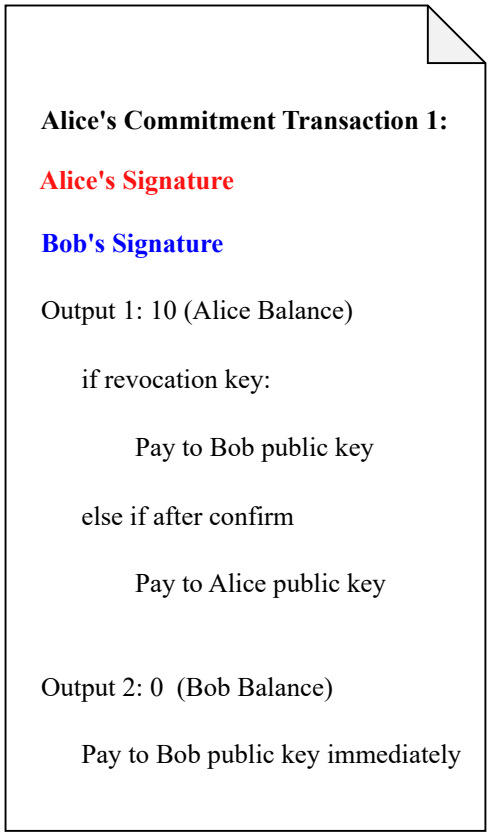


Figure 2.11: Alice Commitment Transaction 1 with RSMC

2.2.3 Hash Time Lock Contract

The Hashed Timelock Contract (HTLC) [29] [30] [31] is an important element in a Lightning Network transaction and consists of several components. The first element is the hash lock, which is a cryptographically scrambled version of a public key generated by the person who initiated the transaction. The associated private key is then used to unlock the original hash. The second important element of HTLC is the timelock, which causes the contract to expire if the private key is not provided by the time it expires.

In Lightning Networks, the hash time lock contract is mainly used for multi-node transactions. Generally, the private key R is generated by the transaction receiver, who provides the corresponding public key hash(R) to the sender. The transaction sender then uses the public key hash(R) to create a hash time lock contract. The

receiver can obtain the payment by showing the private key R to the contract, or the sender can be refunded if the time expires.

In the Alice and Bob scenario, Bob generates the private key R and gives the public key hash(R) to Alice. Alice creates a hash time lock contract to pay Bob. Bob can obtain the payment by revealing the private key to the contract or refunding the money to Alice if the time expires.

This is just a simple example. As mentioned before, the hash time lock contract is mainly used for multi-node transactions in Lightning Networks. We will discuss this in more detail in the next section.

The Hash Time Lock Contract is an application of "Pay to Script Hash" (P2SH). In P2SH transactions, a new redeem concept called the Redeem Script is used. When money is transferred to the address of the P2SH Script, the public key script is not filled with a list of public key addresses, but with the hash value of the Redeem Script, which makes the lock script very short. The unlock script is only long when P2SH transfers out. This avoids the problem of having a lengthy lock script in a multi-signature transaction, which could result in a dramatic increase in transaction fees.

Let's make a quick comparison between P2PKH and P2SH [32]:

We set up a 2-of-4 multi-signature address in P2PKH, so the lock and unlock script will be as follows:

Lock Script: 2 PubKey1 PubKey2 Pubkey3 Pubkey4 4 OP-CHECKMULTSIG

Unlock Script: Signature1 Signature2

In P2SH case:

Redeem Script: 2 PubKey1 PubKey2 Pubkey3 Pubkey4 4 OP-CHECKMULTSIG

Lock Script: OP-HASH160 [20-byte hash to Redeem Script] OP-EQUAL

Unlock Script: Signature1 Signature2 Redeem Script

We can notice that in the P2SH case, the lock script in P2PKH is changed into a redeem script, and the new lock script is the hash result of the redeem script.

To make it more visualized, we can look into the sample shown in Fig. 2.12. It is a P2PKH 2-of-5 multi-signature address-lock script. If we get rid of the abbreviations and replace them with the actual public keys, the lock script looks very long. So, in P2SH, the original lock script is changed to a redeem script and the redeem script is compressed into a 20-byte hash result. Transaction fees are high due to long scripts, so P2SH reduces the transaction fee for the transaction sender. However, the redeem script needs to be provided by the unlock script, so the transaction receiver will need to pay a higher transaction fee when they want to spend the money in a multi-signature address. In other words, when they provide the unlock script in the P2SH case.

```
2
04C16B8698A9ABF84250A7C3EA7EE-
DEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395D7EEC6984D83F1F50C900A24DD47F
569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3FB87A3495E7AF308EDF08DAC3C1
FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D0E34224858008E8B49047E63248
B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812F393DA7D4420D7E1A9162F0
279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255382ED7F78E946580657EE
6FDA162A187543A9D85BAAA93A4AB3A8F044DA-
DA618D087227440645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1EC-
ED3C68D446BCAB69AC0BA7DF50D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA7
74D207D137AAB59E0B000EB7ED238F4D800 5 OP_CHECKMULTISIG
```

Figure 2.12: Sample Multi-Signature Address Lock Script [1]

2.2.4 Transaction in Lightning Network

In the entire Lightning Network, the multi-signature address acts as the payment channel among users. That means if we want to pay someone, it is not necessary to create a new multi-signature address; the payment can be transferred across existing payment channels. Let's complicate the Alice and Bob scenario a little more. Alice and her friend Carol usually buy breakfast together every morning from Dave. Carol has a multi-signature address with both Bob and Dave, but Alice has a multi-signature address only with Bob. The Fig. 2.13 shows their status. And here's the interesting thing: Alice does not have to create a new multi-signature address with Dave to pay

for her breakfast. She can pay Dave through the payment channel between them. In other words, Alice pays Bob, Bob pays Carol, and Carol pays Dave.

So, here is the specific process of Alice pay to Dave in Fig. 2.13: When Dave knows Alice wants to pay him, he generates the private key R and public key hash(R). Then, he gives hash(R) to Alice. Alice uses hash(R) to create HTLC 1 to pay Bob. Bob uses hash(R) from Alice to create HTLC 2 to pay Carol. Carol uses the same hash(R) to create HTLC 3 to pay Dave. In this case, all multi-signature addresses work as the payment channel to Dave. All HTLC use the hash(R) from Dave. Once Dave notices the HTLC3 from Carol, he will reveal the private key R to HTLC3 to get funds. Then, Carol will know the private key R as well because Dave reveals it in HTLC3. So Carol can get funds from HTLC2 by using the same private key R. By the same theory, Bob can get funds from HTLC1. Judging from the results, the general balance of Bob and Carol is not changing. Alice pays for her breakfast, and Dave receives payment. And Alice saves the cost of creating any new multi-signature addresses for Dave.

It is worth noting that even if Dave does not reveal the private key R, Alice, Bob, and Carol will get the refund from their HTLC after the time expires. Also, if Carol does not reveal the private key R to Bob, which she gets from Dave, there is no benefit to her because her money has already been sent to Dave. The only way to get her money back is to reveal the private key R to Bob on HTLC 2. The same theory applies to Bob and Alice.

This is how the Lightning Network makes transactions through every user. One person does not have to create a multi-signature address with everyone in the network but is linked through the payment channel made by many multi-signature addresses.

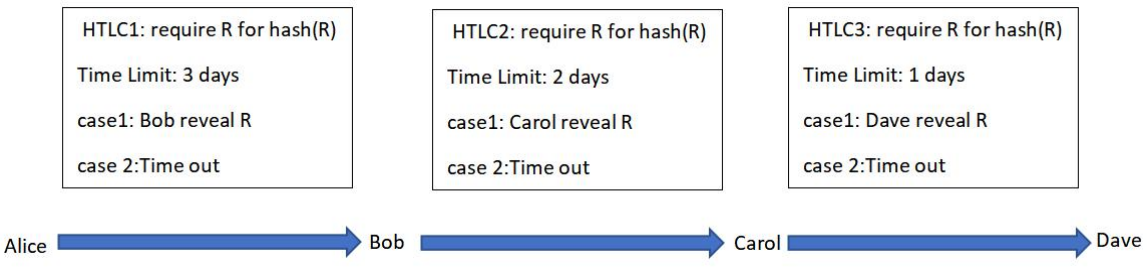


Figure 2.13: Transaction Path from Alice to Dave

2.3 Redis

2.3.1 Redis Cluster

Before discussing the architecture of the Redis cluster, let's briefly review the architecture of a single Redis instance [33] [34] [35].

In order to achieve read/write splitting, servers in Redis are classified as masters and slaves. Write operations from the client are directed to the master server. The master server then synchronizes the data to the slave server, which contains a copy of the data from the master. Read operations from the client are directed to the slave server. In this case, write and read operations happen on different servers. However, in some scenarios, a single-instance storage Redis cache will encounter several problems.

Although Redis single-instance read and write splitting can balance the load of read operations, all write operations still fall on the master node. The scenario of massive data and high concurrency puts great pressure on the master node.

In the case of massive data storage, a single Redis server will not be able to contain all the data. Too much data means high persistence costs. In severe cases, the client request may block the server, resulting in a decline in service requests and reduced service stability.

Therefore, Redis version 3.0 introduced the cluster mode, which realizes the distributed storage of data by splitting the data and storing different data on different master nodes, thus solving the storage problem of massive data.

The Redis cluster adopts the idea of decentralization, and there is no central node. From the client's perspective, the entire cluster can be viewed as a whole and can be connected to any node for operation, just like operating a single Redis instance. When the key of the client's operation is not allocated to a node, Redis returns a redirect instruction, pointing the client to the correct node.

As shown in Fig. 2.14, the Redis cluster can be viewed as a combination of multiple master-slave architectures, each of which can be viewed as a node (where only the master node has the capability to handle requests).

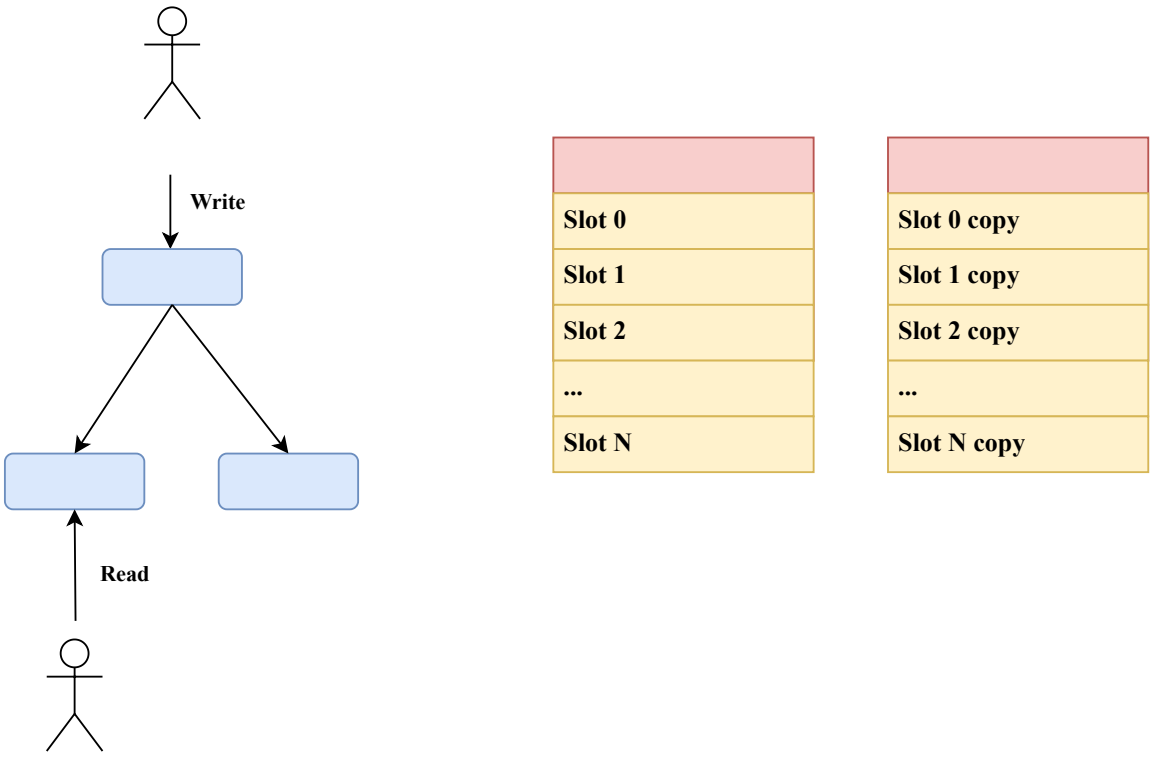


Figure 2.14: Redis Single Cluster Structure

2.3.2 Hash Slot

As mentioned earlier, Redis cluster solves the problem of storing massive data on a single node by utilizing distributed storage. When considering distributed storage, it is essential to determine how to split the data among different Redis servers. Common partitioning algorithms include hash algorithm and consistent hash algorithm.

In a normal hash algorithm [36], the key is computed by hash(key) mod N, where N is the total number of nodes. For example, if there are three nodes and six clients, the key could be the client's ID, and the hash algorithm would be hash(id) mod 3. The advantage of this approach is its simplicity. However, if server 3 goes down, only two machines will be available in the cluster, resulting in the hash function becoming hash(id) mod 2. This leads to a problem where all the data needs to be recalculated to find new storage servers, requiring a lot of data migration each time a server goes down or a new server is added. This can cause the system to become unreliable and

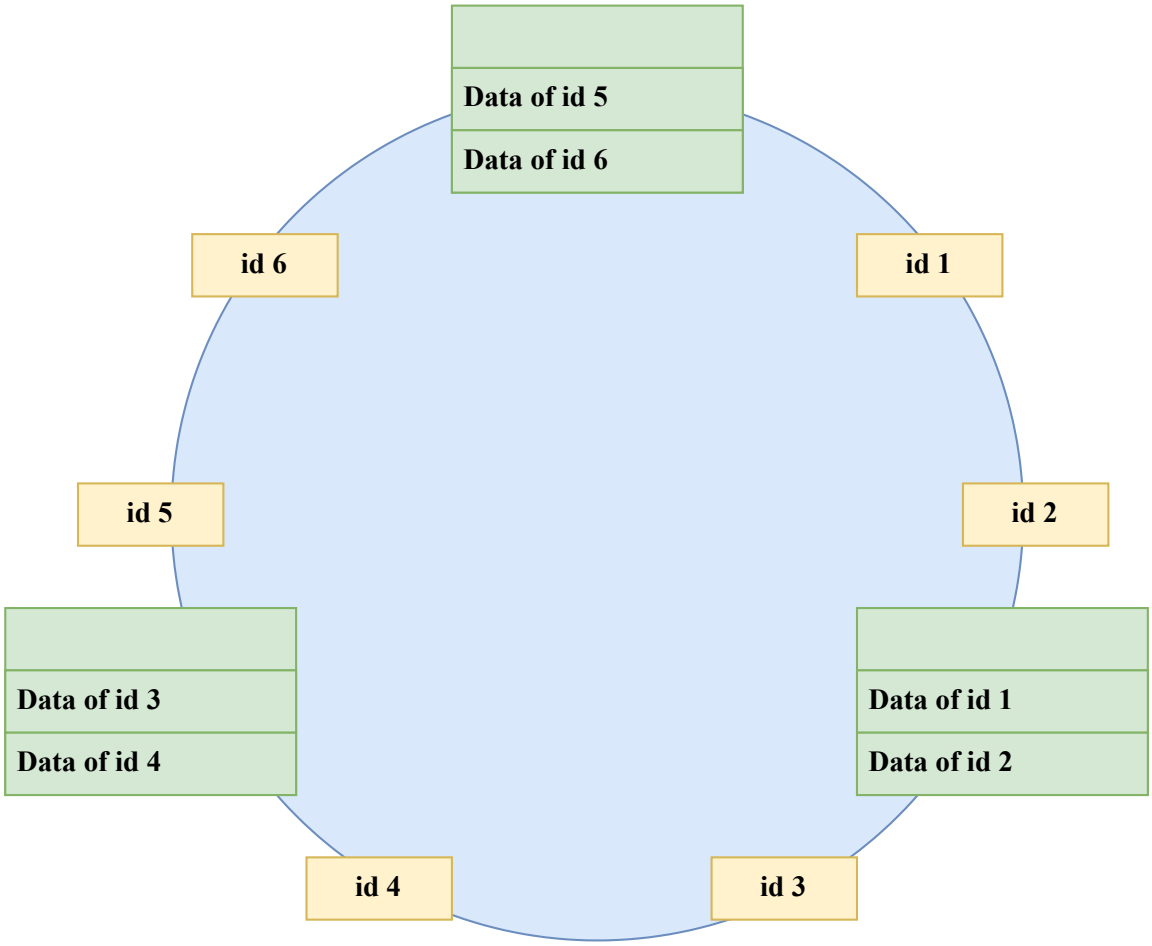


Figure 2.15: A Simplified Sample Hash Ring

In the consistency hash algorithm [37], both the server and data are mapped to an end-to-end hash ring [38] through the hash function. The storage node mapping can be based on node IP addresses. After the data is mapped to the hash ring like Fig. 2.15, the storage server is searched in a clockwise direction, i.e., the first storage server is found in a clockwise direction starting from the location of data mapping on the ring. So it's stored on this server. In the figure below, client 0 and client 1 will be mapped to server 1 because it is the first storage server they found in the clockwise direction. When server 1 goes down, client 0 and client 1 will be mapped to server 2.

Compared with the normal hash algorithm, the consistency hash algorithm reduces data migration when the storage server goes down, but cannot solve the problem

of data loading imbalance.

Therefore, Redis is using a hash slot algorithm [39] to shard the data. A hash slot is essentially an array space. To solve the problem of even distribution, Redis added another layer between data and nodes. This layer is called a hash slot, which is used to manage the relationship between data and servers. Now, it is equivalent to placing slots on servers and placing data in slots.

There are 16384 hash slots in the Redis cluster (the range of slots is 0-16383, hash slots). Different hash slots are distributed on different Redis nodes for management, i.e., each Redis node is only responsible for a part of the hash slots. During data operation, the cluster will use the CRC16 algorithm [40] to calculate the key and modulo 16384 (slot = CRC16(key) mod 16383). The result is the slot in which the key-value is placed. Through this value, the Redis node corresponding to the corresponding slot can be found. And then, we can access directly to the corresponding node.

The advantage of using hash slots is that nodes can be easily added or removed without making the cluster unavailable. When you need to add nodes, you simply move some of the hash slots from other nodes to the new node. When a node needs to be removed, it simply moves the hash slot from the removed node to another node.

2.3.3 Advantages of Redis

Memcached [41] is a simpler system that is optimized for storing and retrieving small pieces of data, such as cache keys and values. It has a very fast, single-threaded architecture that can quickly handle high volumes of simple read and write operations. Memcached also has a distributed architecture that allows it to scale out across multiple servers, which can further increase its performance in certain scenarios.

If your application requires simple, high-speed caching of small pieces of data, Memcached may be a better choice. But in our case, BLN need a more full-featured data store that can handle a wider range of use cases and support persistence, so Redis may be a better choice.

KeyDB [42] is another relatively new database that was designed to improve Redis's performance in multi-threaded scenarios. KeyDB's multi-threaded architecture allows it to handle more concurrent requests than Redis, making it a better choice for

high-concurrency workloads. KeyDB also includes features such as Adaptive Memory Management that can improve memory usage efficiency.

In summary, if your workload requires high concurrency and you need better performance in multi-threaded scenarios, KeyDB might be the better choice. If you value a mature ecosystem, a wide range of modules, and efficient performance in low-concurrency scenarios, Redis might be the better choice.

Chapter 3

BLN: A Fast Layer-2 Framework

By describing the problems identified in the previous chapters, we can gain an understanding of the issues present in Lightning Network and Redis. In this chapter, we will introduce the functions and structures of BLN, and explain how it addresses these problems.

3.1 Problem with Lightning Network

The main components of the Lightning Network have been introduced, so here's a quick summary:

1.Users set up a payment channel by creating a 2-of-2 multi-signature address.

2.Users put money into the multi-signature address by broadcasting a funding transaction.

3.Users broadcast a commitment transaction to update the balance of the multi-signature address.

4.A commitment transaction is valid only if both parties sign it.

5.Only the latest commitment transaction can be broadcast; otherwise, the party who broadcast it will be punished, and their money will be sent to the other party.

6.The purpose of the hash time lock contract is to allow for global transactions across multiple nodes via hashes.

The idea is to reduce the number of unnecessary broadcasts by making an agreement between both parties to the transaction and merging small transactions. More importantly, it reduces the number of transaction broadcasts, which improves the scalability of the Bitcoin blockchain. In the meantime, a small transfer, such as buying a cup of coffee, might cost less than the transaction fee in the original Bitcoin network. In the Lightning Network, the transaction fee for such small transactions has been greatly reduced for the user. However, the design also has disadvantages.

First of all, the Lightning Network only supports small transactions, such as buying a coffee, because paying through multi-node is like moving pieces on an abacus. If Alice wants to pay Carol through Bob, in the Lightning Network case, she needs to move one piece to Bob, and Bob needs to move one piece to Carol. Even though Bob does not lose money in general, two multi-signature addresses are still independent. Like in Fig. 3.1, after Bob moves one piece to Carol, he has no pieces left to move to Carol. So, in fact, Bob's payment ability to Carol is affected by Alice's transaction. He can no longer pay Carol if he wants to buy something from her. Moreover, if Alice wants to move three pieces to Carol, Bob cannot help this time because he only

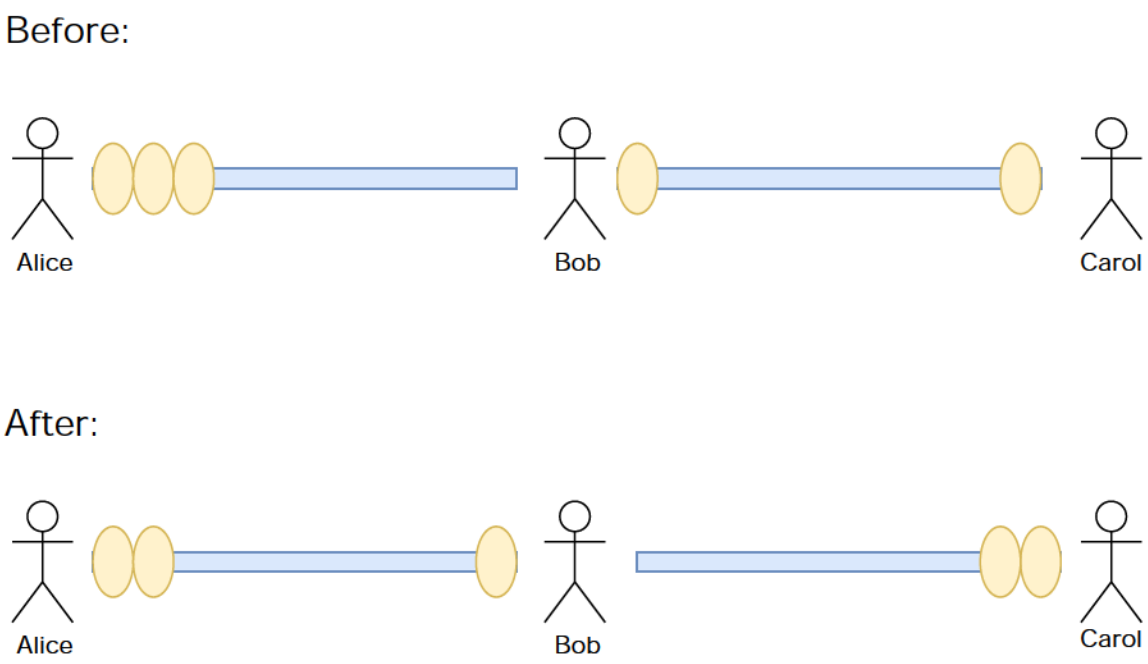


Figure 3.1: Moving One Piece from Alice to Carol

Next, in Fig. 2.13, we can see that the time limit for each HTLC along the transaction path from Alice to Dave is different. That is because we need to provide an adequate response time. If the time lock length is too short, many HTLCs might expire before they get a response. Additionally, taking into account that the other

person might not respond until the last minute. For example, HTLC3 gives Dave 1 day to respond and Dave reveals the private key at the end of the day, Carol might not be able to respond to HTLC2 in time if the time lock in HTLC2 is also 1 day. Therefore, when the path from Alice to Dave gets longer, the time lock is also longer and has more routing fees. However, we do not want to buy coffee and have it take 3 days for a response. This is another problem.

Last but not least, the decision on whether a transaction is canceled lies not with the sender, Alice, but with the receiver, Dave, because Dave provides the private key and is the only one who knows it. In the situation discussed earlier where Alice buys breakfast face-to-face from Dave, if Dave receives funds from HTLC3 and he does not give breakfast to Alice, they could argue directly. However, in real life, online shopping is as frequent as shopping in stores. If the situation discussed above is the case of online shopping, there is no guarantee that Dave will actually deliver something to Alice. So not only is it unfair to Alice, but her rights to receive merchandise after she pays cannot be guaranteed based on the current rules.

3.2 Overview of BLN

In Section 3.1, we discussed the existing problem with the current structure, using Alice buying a coffee from Bob as an example to illustrate how the Lightning Network facilitates transactions. But according to our observation, in real-life scenarios, transactions are more complex than a simple coffee purchase. The transaction between Alice and Bob in the example could be completed directly as it was a face-to-face transaction where one person pays and the other delivers. However, in online shopping, which constitutes a significant portion of real-life transactions and often involves larger transaction amounts, the Lightning Network's performance in terms of safety and reliability is poor as we described at the end of Section 3.1.

As mentioned in Section 2.2.5, the ability of intermediate nodes involved in a transaction to pay is affected by the length of the transaction path. The more intermediate nodes involved, the higher the transaction fees charged to the buyer. In the current Lightning Network, the transaction fee for intermediate nodes depends

on themselves, making it a free market where fees may fluctuate.

In terms of safety, if Alice and Bob are trading online, the transaction's safety is unfair to Alice because Bob decides the private key R, and she only knows the public key hash(R). When Alice sets up her HTLC for Bob using the public key, she cannot get her money back until the HTLC expires. In this case, Bob can directly obtain the money in HTLC because he is the only one who decides the private key R. Bob can then refuse to deliver the merchandise, claiming that he already delivered it, and there is no way to determine who is telling the truth. Bob faces no direct loss, whether he lies or not, but Alice risks paying the money without receiving the merchandise she deserves. This phenomenon occurs because the payment and delivery times are staggered. In contrast, face-to-face transactions like buying coffee rarely involve paying without receiving the merchandise. If such a situation occurs, it results in a real-life argument. However, in an online transaction, there is a possibility of lying as no one knows what is really happening except for those involved. Therefore, if we want to extend shopping to online shopping, a new business model is necessary that involves an impartial third party to ensure both parties' rights. In the previous example, a reliable third party must be involved in the transaction between Alice and Bob to ensure that Bob receives Alice's money transfer and that Alice receives Bob's merchandise.

Therefore, we propose the Business-Oriented Layer-2 Network (BLN) as a solution to these problems. BLN combines the Lightning Network with the business model adopted by many online retailing platforms, such as Amazon. BLN includes a Transaction Processing Center (TPC) that serves as a supervisor and broker for every transaction. We can set up multiple BLN platforms in different regions, with each platform having its own TPC. In addition to the buyer (Alice) and the seller (Bob), we introduce two additional roles: delivery and storage, to account for all shopping scenarios. The TPC acts as a transaction broker for all roles and must monitor each role to prevent mistakes. If any mistakes occur, they will be punished appropriately. At the same time, the TPC must ensure everyone's legitimate rights: the buyer must receive the merchandise they paid for, and other roles should receive payment from the buyer. Naturally, the TPC must handle a large volume of transaction requests

since it is the only intermediary. We chose to use Redis cluster, a key-value database, as the TPC's structure because it can efficiently process frequent read and write operations. We also revised some configurations to improve its performance.

3.3 Architecture of TPC

This third-party structure must not only ensure the due rights of both sides of the transaction but also solve the existing problems of the lightning network. An

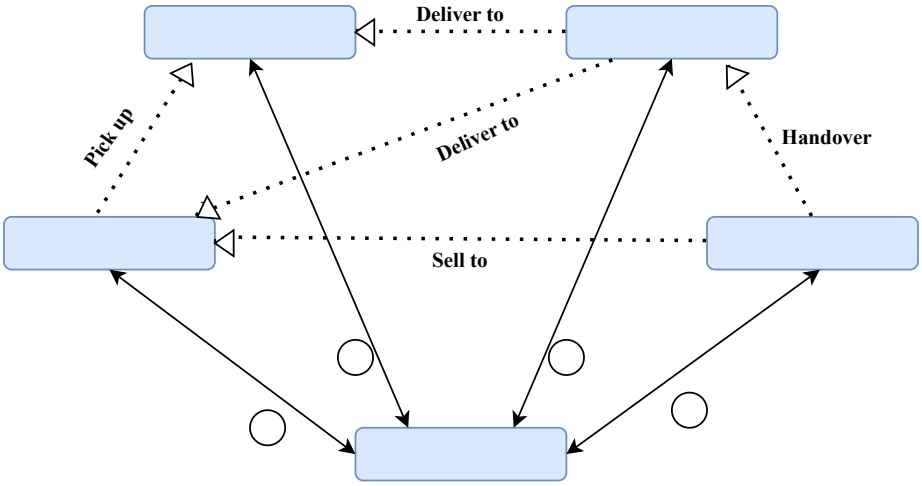


Figure 3.2: Business Model Based on BLN

Most modern shopping can be summed up into two types. The first is where the buyer goes directly to the store, pays the money, and then receives the merchandise directly. The second is online shopping. The buyer pays online, and the merchandise is delivered directly to the buyer's home by a deliveryman. Alternatively, the deliveryman will deliver the merchandise to the storage, where the buyer presents a voucher and picks up the merchandise. Therefore, the members who might be involved in a transaction include the buyer, seller, delivery, and storage. If we add the third-party TPC, there will be five members involved in a transaction. Based on the

transaction mode of the lightning network, we have built a business model shown in Fig. 3.2.

To be noticed, the TPC and four other members each have a separate payment channel. So, the primary role of the TPC is as an intermediary agent. And here, we assume that TPC is absolutely impartial and unbiased. Although we follow the Lightning Network's way of trading via HTLC here, in this model, the private key of HTLC is determined by the transaction sender rather than the receiver in the Lightning Network. The reason the Lightning Network decides on the private key by the receiver is to guarantee that the transfer will eventually be received by the receiver. If not, the receiver will not reveal the private key, and all HTLCs along the path will expire eventually. The difference in our business model is that the private key is decided by the buyer to make sure the buyer will get the merchandise in the end, or he or she will not reveal the private key. And TPC needs to make sure the seller gets the payment after the buyer receives the merchandise.

We will explain how this business model works through the online shopping process of a buyer. And the merchandise will be passed through the deliveryman and storage. At first, the buyer selects what he wants to buy online and initiates the first HTLC1 in channel 1 using the public key hash(R). The HTLC1 contains the cost of paying for the merchandise, the cost of delivery, and the cost of storage, and the buyer shall not reveal the private key R to anyone until he or she receives his or her merchandise. Then, the TPC will initiate another HTLC2 using the same public key hash(R) and send it to the seller side through channel 2. The HTLC2 contains only the payment for merchandise. But the seller cannot get the payment because the seller does not know the private key R yet.

Next, the seller notices HTLC2, and he or she needs to give the merchandise to the deliveryman. Then both the seller and deliveryman need to inform TPC that they have already completed the handover of merchandise. In this case, TPC initiates HTLC3 in channel 3 as the delivery fee to the deliveryman. Of course, the deliveryman will not receive the delivery fee at this time. The same procedure occurs when the deliveryman and the storage complete the handover of merchandise, TPC initiates HTLC4 in channel 4 as a storage fee.

At this point, all that remains is for the buyer to collect the merchandise at the storage. When the buyer reaches the storage, he or she needs to reveal the private key R to the storage to prove that he or she is the owner of the merchandise. When the storage confirms the private key R by using the key to get the storage fee from HTLC4, the storage can give the merchandise to the buyer. Because the storage reveals the private key in HTLC4, TPC will know the private key as well and it also knows that the buyer got the merchandise already. Then TPC sends the same private key to the seller and deliveryman in another way. Again, all four HTLCs use the same public key hash (R) so the seller and deliveryman can get their payment in HTLC2 and HTLC3. The TPC can get the payment from the buyer in HTLC1. At this point, all steps of the transaction are complete. The TPC, storage, deliveryman, and seller get the payment from the buyer. The buyer gets the merchandise.

Compared to the Lightning Network, this new business model mainly solves the problem of staggered time between the buyer's receipt and the buyer's delivery in the online shopping scenario. By switching the determining party of the private key, the seller cannot receive any payment unless he or she deliver the merchandise. Similarly, the buyer must reveal the private key to receive the merchandise. Now, everyone's rights are guaranteed.

Another important point to note is that, in this business model, the buyer must have a face-to-face transaction with a party other than TPC. Specifically, if the buyer goes to a physical store to buy something, they must deal with the seller face to face. If it is an online shopping scenario, then the buyer must face the deliveryman or storage personnel to handover the merchandise. In this case, neither party can benefit by lying.

Last but not least, this business model covers all of the purchase scenarios mentioned earlier. Even if the buyer purchases directly from the physical store, the above business model only needs to exclude storage and delivery. The transaction only needs to be completed between the buyer, seller, and TPC.

3.4 Expired Transaction and Penalty

Although in normal transactions, the rights and interests of each party are equally protected, it is inevitable that a transaction cannot be completed due to human error. So when a transaction fails, the TPC needs to be able to determine which party is to blame. When the TPC identifies who is responsible, it needs to punish the responsible party so as to give no one an incentive to make mistakes. Here, we mainly consider which party has made a mistake: the buyer, seller, deliveryman, or storage. Because HTLC has a time lock, the TPC can determine which party is responsible for an expired HTLC in the channel. To prevent these four parties from making mistakes, each member must keep another resident HTLC as a margin with the TPC. All four used the old lightning network method to set up HTLC. In other words, the private key R is decided by the TPC. If the TPC can determine which party is responsible in the event of an expired transaction, it can collect the money in the HTLC as a penalty by revealing private key R.

So here is how the TPC ascribes blame: In the example discussed in Section 3.3, each HTLC is set up in chronological order. So now we are going to figure out who was responsible in chronological order.

First, let's assume that an error occurred with the seller. The possible error is that the seller accepts the order from the buyer but does not deliver the merchandise on time, which results in the expiration of HTLC1 and HTLC2 in channel1 and channel2. At this time, the HTLC3 corresponding to the deliveryman or the HTLC4 corresponding to storage has not been set yet. Therefore, the TPC can be sure that the mistake was made by the seller and will charge the seller's margin as punishment.

Next, we assume that the seller delivered the merchandise on time and delivered them to the deliveryman. When both of them inform the TPC, then HTLC3 will be set. If the deliveryman did not deliver on time, or delivery was lost in the middle of the transit, the buyer does not reveal the private key R because he or she has not received the merchandise. So HTLC1, HTLC2, and HTLC3 will expire. Since the merchandise has not arrived at the storage yet, HTLC4 has not been set. The TPC can confirm that there was a mistake in the previous step. The TPC can then determine whether

it is the seller's responsibility or the deliveryman's by the time HTLC3 is set up. Suppose that the seller handed over the deliveryman the merchandise more or less at the last moment before the time expires. Then the TPC can conclude that the seller bears the primary responsibility. Assuming that the seller delivered the merchandise to the deliveryman early, then HTLC3 will be set up early as well, and the TPC can determine that the deliveryman is delayed in the delivery process. So the deliveryman takes primary responsibility.

When the transaction has been carried out until the merchandise is delivered to storage, there are multiple cases where HTLC4 can expire. The first one is if the buyer does not pick up the merchandise from the store in time. In this scenario, not only should the TPC deduct the buyer's margin, but the buyer must also set up a new transfer of HTLC1 and pay an additional storage fee. Then, TPC can set up new HTLC2, HTLC3, and HTLC4. The second one is if the merchandise is damaged or missing from the storage. This will also cause the HTLC4 to expire, and the buyer cannot pick up the merchandise. So TPC will deduct the storage's margin. Another possibility is that the previous steps took too long, resulting in insufficient time for the buyer to pick up the merchandise. In this case, TPC can examine the setting times of HTLC2, HTLC3, and HTLC4 to determine who has primary responsibility. The excess deducted after deducting the margin can be refunded to the penalized party through another HTLC. Since there are many possible reasons for the HTLC4 to expire, the TPC needs to gather more accurate information before determining responsibility. In this case, TPC can set up a customer service channel to collect user feedback to understand the specific cause of the error.

3.5 Transaction Processing and Storage

From the previous description, you can see that the TPC can be considered as the central point for all transactions. It is also a node in the lightning network. However, if the TPC is just a single node, it will not be able to withstand the pressure when transactions occur frequently. When the number of users is large, the TPC needs to set up a payment channel with each user. This means that the number of commitment

transactions that need to be saved is high, and how to store, manage, and modify so much data is another problem. Therefore, we must learn from the function of the Redis cluster and make improvements based on this.

In terms of Redis, by default, reads and writes in the Redis cluster are executed on the master, and it does not support reads and writes of the slave node, which is different from the read and write splitting of Redis master-slave replication. The core idea of Redis cluster is mainly to use slaves to backup data and switch over the master when it fails to achieve high reliability. In other words, reliability comes at the cc

Si each other. abases. User RC16 calcul ase it shoul

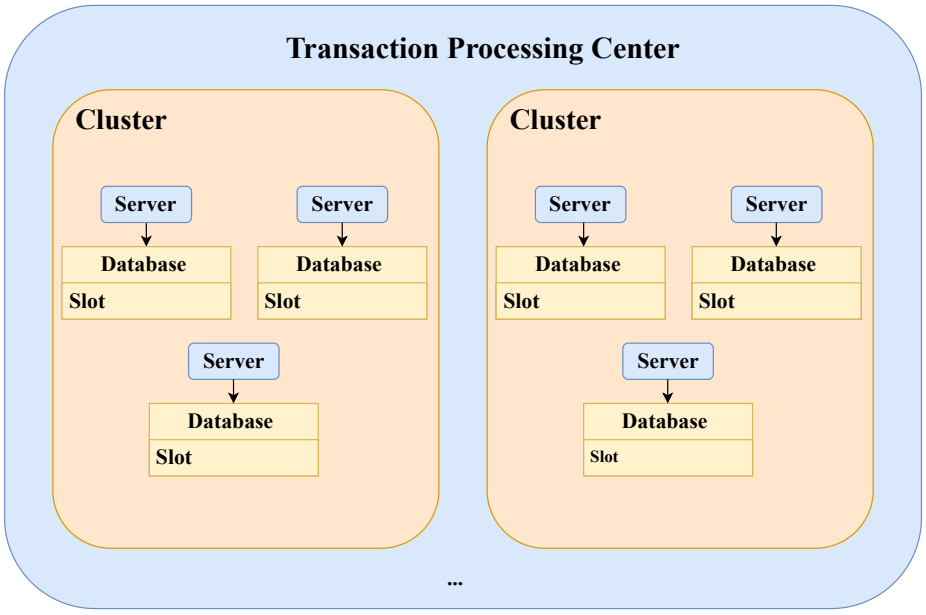


Figure 3.3: Simple Model inside TPC

In the transaction scenario, suppose a buyer has a transfer to a seller. Assuming server 1 is responsible for editing buyer commitment transactions and server 2 is responsible for the seller, the buyer sends a transaction request to one of the TPC

servers. The server receiving the request (it can be any server in TPC) determines which slot contains the buyer or seller commitment transactions. Based on our initial assumption, the server receiving the request knows that server 1 is responsible for the buyer, and server 2 is responsible for the seller. So the server forwards the request to server 1 and server 2 using the same communication method as Redis. According to our business model mentioned in Section 3.3, when server 1 and server 2 receive and confirm the information, the server that receives the request notifies the buyer to initiate HTLC1 to server 1, and server 2 initiates HTLC2 to the seller. If the transaction is completed without error, server 1 and server 2 will draft a new commitment transaction for the buyer and seller.

In terms of data storage, we do not want TPC to have master and slave nodes as in the Redis cluster. The reason is that Redis reads and writes on the master node by default, and the slave contains a copy of master data. Although in Redis, when the master fails, the slave takes over the position of the master to ensure the stability of the system, this stability is based on a large number of redundant copies, while in normal circumstances, the slave resources are idle.

Furthermore, although the Redis cluster allows read and write operations to be performed on the replica by configuration [43], we set the replica only to store copies but not to support read and write operations. During the writes operation, the following could happen: when the client writes data to the master data, the server replies "Done" to the client and then synchronizes data to the replica. In other words, the server does not reply "Done" to the client after all replicas are synchronized because that would be inefficient, especially in the exchange of money. Therefore, if a client wants to read or write data to a replica before server synchronization, replica data may be inconsistent with that of the master data. To avoid this situation for Bitcoin-based applications, it is better that the replica is not configured for read and write. The remaining problem is that both the read and write operations will be concentrated on the master, resulting in too much stress on one server.

So, our design is to average the slots for which each server is responsible. In other words, each server is responsible for a database with some slots as the master and some slots as the replica. Fig. 3.5 shows a simple example:

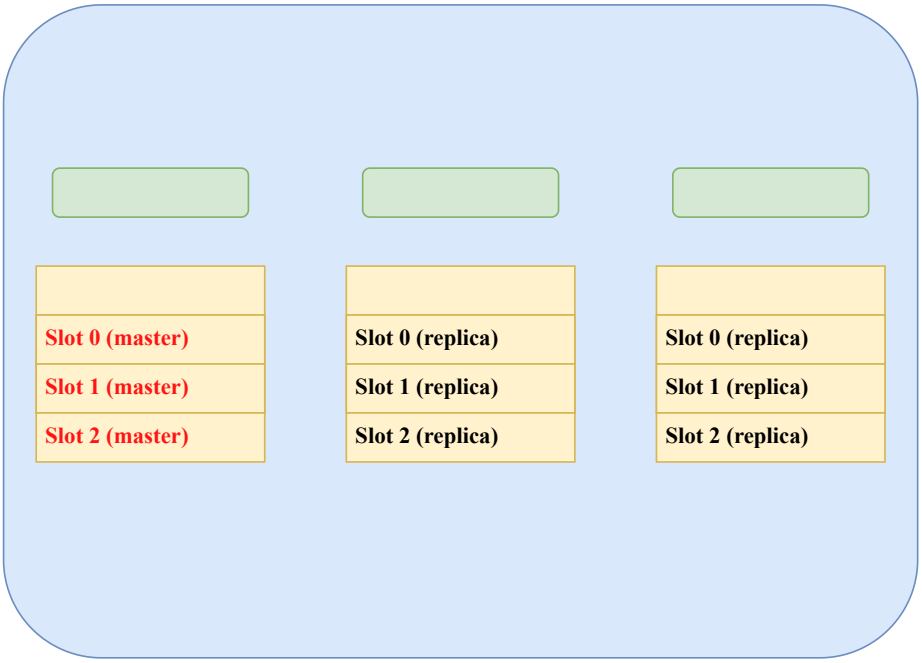


Figure 3.4: Data Storage in Original Redis Cluster

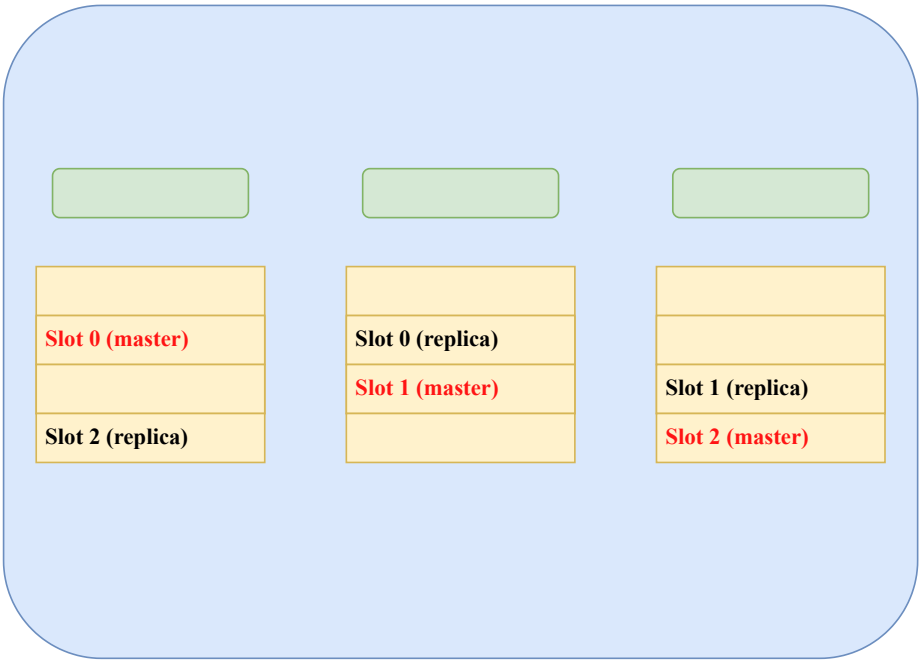


Figure 3.5: Data Storage in Revised Redis Cluster

The difference between the original Redis cluster and the revised Redis cluster is

described in Fig. 3.4 and Fig. 3.5. In the original Redis cluster, the master node contains all the master data, so when write operations are frequent, the master node will experience heavy pressure. The slave nodes also store all slot replicas. To solve these problems, TPC clusters have structures like the one shown in Fig. 3.5. Let's assume that this cluster is responsible for slots 0 through 3. Each server is responsible for editing the corresponding database. Compared with Redis, the master data is evenly distributed among different databases. Since writes only occur on the master data, the write operation pressure is equally divided. The replicas are also equally distributed to different databases in a similar way, and as you can see from this simple figure, some storage space is saved.

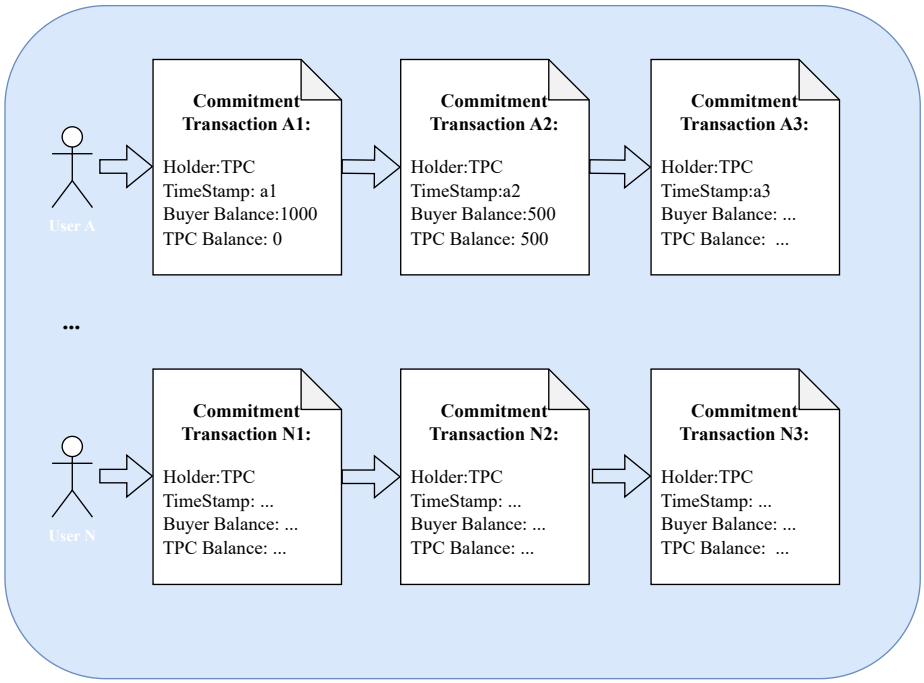


Figure 3.6: Data Storage in Slot

Suppose server 1 crashes because of some error, and server 2 has a copy of the master data that server 1 is responsible for. Server 2 can temporarily take over for server 1. Although there is only one replica for each slot, we can fill Database 1 with another slot 1 replica, Database 2 with another slot 2 replica, and Database 3 with another slot 0 replica. Each slot will have two replicas if needed. The example shown

in Fig. 3.5 is just a simple illustration of how the distribution works. The real data distribution can be controlled artificially by humans according to the capacity of the server. The content stored inside each slot is the commitment transaction between each user and the TPC. More details are shown in Fig. 3.6.

Chapter 4

Performance Analysis

In this chapter, we will simulate the behavior of Lightning Network nodes and compare their performance with BLN when they receive a large number of transaction requests simultaneously. The criteria for comparison will include transaction processing time, transaction fee, and the number of transaction failures. Our goal is to minimize the number of transaction failures and processing times.

4.1 Performance-Related Factors

4.1.1 Capacity of Intermediate Node

In the Lightning Network, for each transaction to be completed, there must exist a path consisting of some payment channel, and each intermediate node on the path should be able to cover the transaction amount. Since the capacity of each intermediate node varies, the amount should not be too large. Otherwise, a node in the path may not be able to cover the transaction. Thus, the Lightning Network can only support small transactions. Transaction failure occurs when there is no path in the current Lightning Network that can cover the transaction amount.

4.1.2 Transaction Fee for Intermediate Node

Meanwhile, the transaction initiator needs to pay a transaction fee to the intermediary node for its assistance. As the entire lightning network operates in a free market, each intermediary node can freely determine the amount of transaction fee it charges. An intermediary node can decrease its transaction fee to enhance its competitiveness and attract more users to choose it as their intermediary node. If the intermediary node is a super node, meaning it establishes payment channels with multiple nodes, more transactions will flow through that node. In such cases, the intermediary nodes can increase their transaction fees to generate more income. To

summarize, the transaction fee for a transaction can vary from high to low due to the influence of each intermediary node.

4.1.3 Number of Buyer and Seller

The impact of transaction failure and transaction fee mentioned in the previous section both comes from the intermediary node of the transaction. However, the sender and recipient of a transaction also affect the number of transaction failures. In this thesis, we will refer to them as the buyer and seller, respectively, to distinguish between them. The buyer is a user who usually transfers money out in the payment channel but rarely receives money, while the seller usually receives transfers most of the time but rarely transfers out. For example, in the scenario where Alice is buying coffee at Bob's cafe, Alice is acting as the buyer, and Bob is acting as the seller. Alice rarely receives a transfer from Bob, and Bob rarely initiates a transfer to Alice. In other words, under the influence of the buyer and seller, most transactions are initiated by the buyer and received by the seller, so the flow of transactions is almost one-way.

However, there is a significant issue: if Bob has only one payment channel with Alice, the path established by other buyers who want to buy coffee at Bob's cafe with the lightning network must flow through Alice. As a result, Alice has to buy coffee for others as well as for herself. Although Alice receives transfers from other buyers in other payment channels, each payment channel is independent. Other buyers' transfers to Alice do not go directly into the payment channel between Alice and Bob. In other words, in the payment channel between Alice and Bob, Alice's balance will only get smaller and smaller. When Alice's balance returns to zero, Alice can no longer act as an intermediary node for others to buy coffee. Meanwhile, Bob only established the payment channel with Alice, so other people who wanted to buy coffee could not transfer money to Bob. Therefore, Alice's balance in the payment channel will directly affect the number of transaction failures. When Alice's balance returns to zero, all other transactions will fail.

4.1.4 Topology

In the later experiments, we generated a connected graph to simulate the lightning network. To ensure a fair comparison, we started by forming a balanced graph, where each node had almost the same number of neighbors. However, in the real lightning network, there exists a super node, which has many neighbors. This is because when a new user joins the lightning network, they are more likely to establish a payment channel with a super node. The reason is that linking with a super node can greatly reduce the number of intermediary nodes in a transaction, resulting in a decrease in the transaction fee for each transaction. In contrast, if a user accesses the network from the edge of the graph, then each of their transactions will start at a corner of the graph. If the end of the transaction is at the opposite corner of the graph, the number of intermediary nodes in the transaction path will be high, leading to an increase in the transaction fee. Therefore, when generating the connected graph, we randomly selected several nodes as super nodes, increasing the probability of other nodes connecting with them.

4.2 Experiment Configuration

In the experiment, we used Java to set up a connected graph with a thousand user nodes to simulate the performance of the Lightning Network and BLN when a large number of transaction requests are encountered at the same time. To choose an optimal path, we used the DFS algorithm to find all the paths between the transaction sender and receiver. From the chosen paths, we selected the shortest one that could handle the transaction amount. If there was no path that satisfied the condition, we marked the transaction as failed. Finally, we measured the quality of the Lightning Network and TPC by counting the number of transaction failures, the total transaction fee, and the transaction processing time.

Before the transaction start up, each user funds N bitcoins for each payment channel (multi-signature address) they own. In other words, the initial balance between the two sides of each payment channel is N to N. Transaction failure will not occur in the Lightning Network and BLN if the amount of each transfer is too small, so we

set the range of transfer amount to 5 to 15 bitcoins, with an average of 10 bitcoins.

The intermediary node among the transaction path will charge a transaction fee for each transfer. We will use the sum of the transaction fees charged by the intermediary nodes for all transactions as the evaluation standard. Although the transaction fee charged by each intermediary node is inconsistent in reality, to prevent the final result from fluctuating, we set the transaction fee charged by each intermediary node at 0.1 bitcoin.

Finally, when the results are available, we will repeat the experiment many times and take the mean of the results of the repeated experiments to reduce bias. The bias can come from the shape of the connected graph and the size of the transaction amount.

Regarding transaction processing time, our main comparison is BLN with Redis Cluster vs. BLN with a single server. We will explain why we need Redis Cluster and what its advantages are. Table 4.1 summarizes the important fixed parameters in the experiment.

Table 4.1: Important Parameters

Parameter	**Description**
Node Num	The number of node in graph
Transaction Num	Numbers of transaction happened at the same time
Traffic Test Num	Number of repetitions of test
Transaction Fee Amount	Transaction fee amount for each intermediary node
Seller Num	Numbers of seller in graph
Buyer Num	Numbers of buyer in graph
Super Node Num	Numbers of super node in super node based graph
Max Neighbour	Max numbers of neighbour in balanced graph
Max Transaction Amount	Maximum amount of a transaction
Min Transaction Amount	Minimum amount of a transaction
Initial Balance in Channel	User initially fund in each payment channel

4.2.1 Graph Generation

Since the experimental results will be affected by the shape of the connected graph, we first created a balanced graph where each user node is limited to four payment

channels, meaning that each node cannot have more than four neighbors. For comparison, we also created some super node-based graphs, where a few nodes are randomly selected to be super nodes during the graph generation process. When a new node is added, we use random numbers to generate a value between 0 and 1. If the value is less than 0.5, the new node will connect to a random super node; otherwise, it is connected to any existing normal node.

To generate a connected graph, we use the traditional approach [44]: there are 1000 nodes in the experiment, and we use 500 of these nodes first. We set a probability, such as 30 percent, and each node has a 30 percent chance of being connected to any of the other 499 nodes. However, the graph generated in this way is likely not a connected graph, and there will be many isolated "islands" in the graph. Therefore, we select the island with the highest number of nodes and remove the edges of all other islands. Assuming that the largest island has N nodes, this leaves 1000-N nodes that have not yet been connected to the graph. Next, we randomly connect the remaining 1000-N nodes to any node on the island, resulting in a connected graph. It is worth noting that we only choose nodes among the largest island to be super nodes.

4.2.2 Choice of Buyer and Seller

When the transaction brings in buyer and seller factors, we set the number of sellers to be 1 percent of the total number of user nodes. There are 1000 nodes in the experiment. After graph generation, we will randomly select 10 nodes to be sellers. The rest of the nodes will be considered as buyer.

4.3 Experimental Results

4.3.1 Transaction Failure Rate Under Low Balance

For comparison, we change the initial balance to 50, 100, 150, and 200, which means the average transaction amount now costs about 20, 10, 6.6, and 5 percent of the initial balance.

From the result in Fig. 4.1, Fig. 4.2, Fig. 4.3, and Fig. 4.4, BLN's average transaction failure rate performance in the case of large transactions is far better than lightning network's. It doesn't matter whether the lightning network's connected graph is a balanced or super node-based graph. The reason is that each BLN user no longer acts as an intermediary node. Each person's transaction and payment channel are independent, meaning that one user's transaction does not affect another user's payment channel. The only intermediary node is the TPC. In Fig. 4.1, the transaction failure rate rises up to about 30 percent when its average cost is 20 percent (10 bitcoins) of the initial balance per transaction, but it performs much better if we increase the initial balance to 100, 150, and 200. In other words, if TPC has enough money in the payment channel, transaction failure rarely occurs.

Meanwhile, another reason why the lightning network performs worse than BLN is that if the payment path of a transaction in the lightning network is extremely long, there will be a large number of intermediary nodes involved. If one of the intermediary nodes does not have enough money in the payment channel, the entire payment path cannot be used. Therefore, in the case of large transactions, the probability that no path can meet the transaction amount increases. In other words, the transaction failure rate also increases. The transaction failure rate will also rise when the total number of transactions occurring at the same time increases. This is consistent with the fact that the lightning network does not support large transactions but favors small transactions.

We can observe that the transaction failure rate in a lightning network will slightly decrease when super nodes exist in Fig. 4.1 and Fig. 4.2. This is because some payment paths do not have to be as long, and the number of intermediary nodes involved in transactions decreases. The difference between the two super node-based graphs is that one graph has only 5 super nodes, while another graph has 10. According to our previous method of generating graphs, since the total number of nodes is fixed at 1000, the average number of normal nodes connected per super node in the graph with 5 super nodes is more than that in the graph with 10 super nodes. In other words, the graph with 5 super nodes is more concentrated. All the normal nodes are clustered around the 5 super nodes, which results the payment path becoming shorter

than when all nodes are clustered around the 10 super nodes.

Therefore, we can conclude that when super nodes exist and connect more normal nodes, the payment path will be shorter in general. As long as the payment path is shorter, fewer users will be affected by other transactions, and the transaction failure rate will go down. The presence of super nodes is beneficial to the transaction success rate. The TPC in BLN accomplishes this task by shortening the path of each transaction. Thus, even though there are a large number of transactions at the same time, the transaction failure rate for BLN is better than that for lightning network.

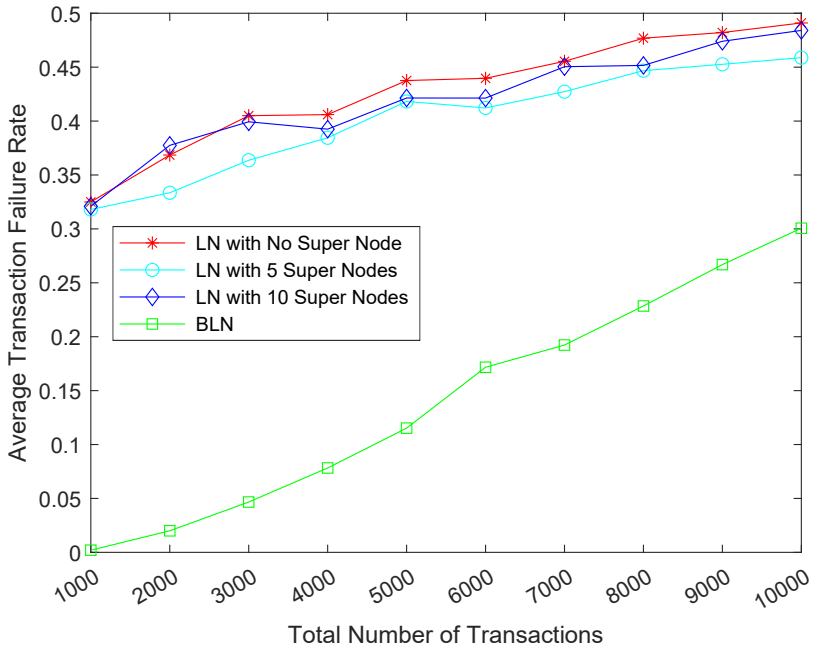


Figure 4.1: Transaction Failure Rate (Lightning Network vs. BLN, Initial Balance=50)

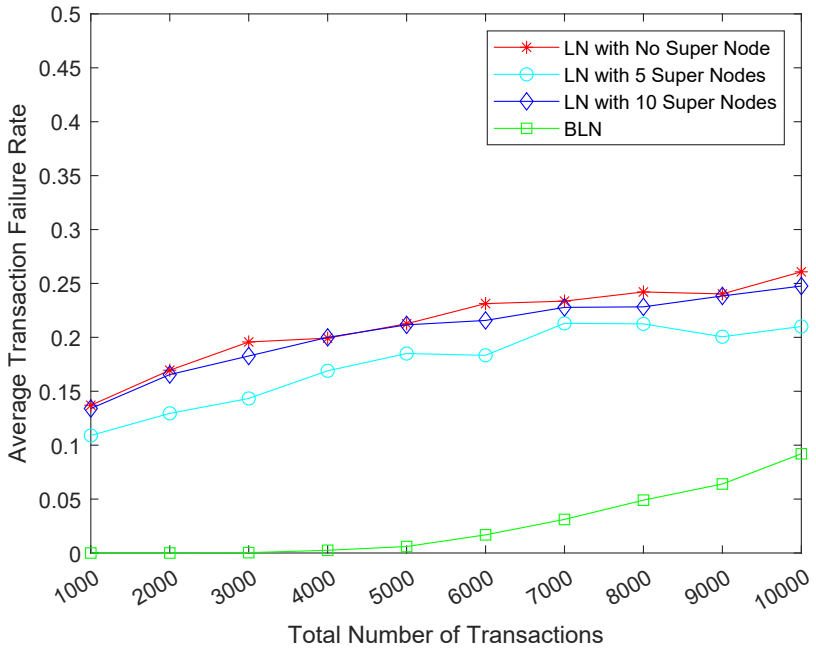


Figure 4.2: Transaction Failure Rate (Lightning Network vs. BLN, Initial Balance=100)

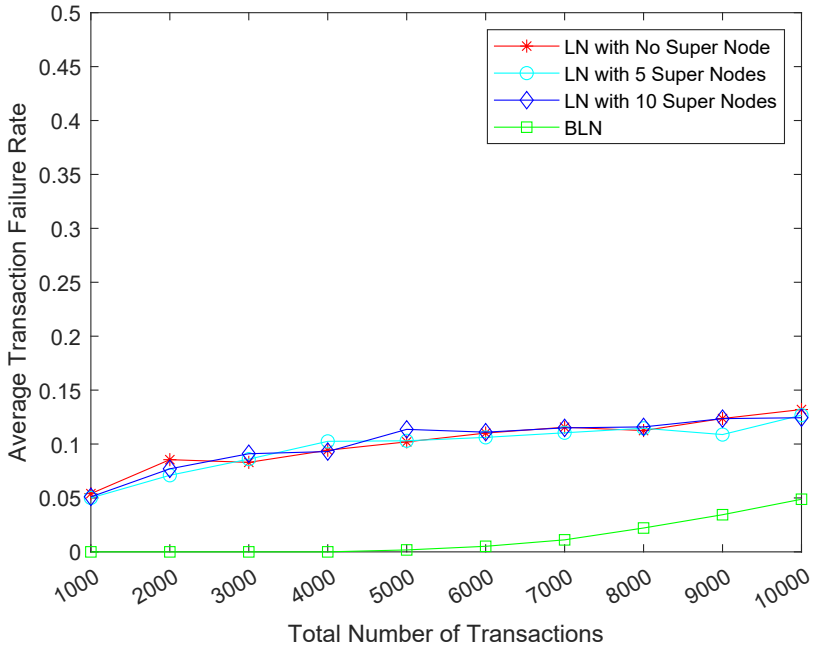


Figure 4.3: Transaction Failure Rate (Lightning Network vs. BLN, Initial Balance=150)

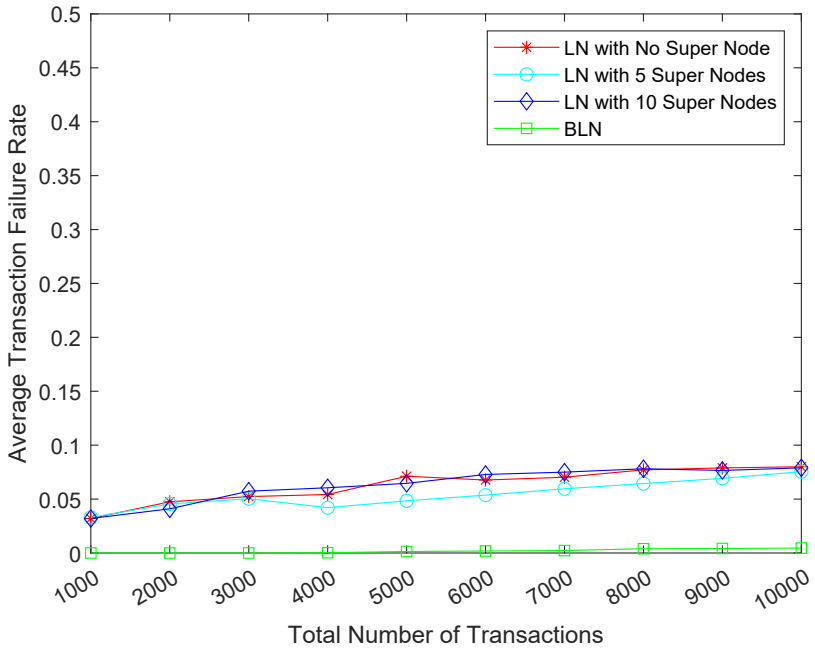


Figure 4.4: Transaction Failure Rate (Lightning Network vs. BLN, Initial Balance=200)

However, when we add the Buyer/Seller factor, that is, the receiver of the transaction is no longer random but biased, the transaction failure rate of the lightning network will increase significantly. At this point, whether a transaction can be completed is almost entirely determined by the node that establishes the payment channel directly with the seller. When all transactions flow one-way from buyer to seller, almost all transactions flow through the intermediary node that directly establishes the payment channel with the seller node. That is, if that intermediary node does not have enough balance to send money to the seller, no subsequent transaction can be made. Even if the intermediary node directly linked to the seller funds more money into the payment channel, it will not solve the problem. Because that intermediary node is essentially a user, he or she will not fund too much money in the payment channel with the seller in order to help someone else complete the transfer. Otherwise, he or she cannot complete his or her own transactions. This experiment also highlights a big problem of the lightning network: lightning network does not support large transactions. Only very small amounts of money can be transferred through the lightning network.

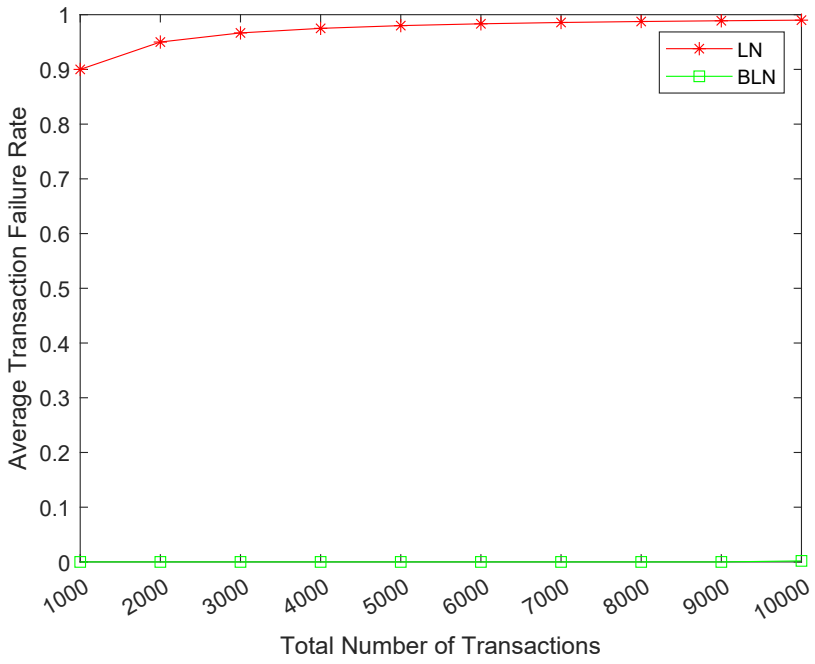


Figure 4.5: Transaction Failure Rate with the Existence of Buyer and Seller (Lightning Network vs. BLN)

The BLN solves this problem: The TPC itself is a completely user-serving broker. Instead of investing money in the payment channel with each user as in the previous experiment, TPC only needs to fund money in the payment channel with the seller and does not need to fund money in any buyer's payment channel. Because the money flows one way: from the buyer to the TPC, then from the TPC to the seller. In other words, the pressure to fund money to payment channels is completely shifted from the user to the third-party TPC. This improves the user experience and allows the TPC to handle large transactions as long as it has enough funds. We can see in Fig. 4.5 that BLN never experiences transaction failure.

The Table 4.2 summarizes the specific value of each parameter used in the transaction failure rate experiment.

Table 4.2: Parameters in the Transaction Failure Rate Experiment

Parameter	**value**
Node Num	1000
Traffic Test Num	100
Super Node Num	5,10
Transaction Fee Amount	0.1
Seller Num	10
Buyer Num	990
Max Neighbour	4
Max Transaction Amount	15
Min Transaction Amount	5
Initial Balance in Channel	50,100,150,200

Regarding additional information about this experiment, the transaction failure rate has an upper limit in each trial. As shown in the figures, when the total number of transactions increases, the transaction failure rate also increases. This is because the payment channel balance of some nodes has been depleted. Nevertheless, in subsequent experiments, we raised the total number of transactions to 100,000 or even 200,000, resulting in the transaction failure rate fluctuating around a fixed value. Since the experiments involve random transactions, the transaction initiator and receiver are chosen randomly. Consequently, a node's payment channel balance can be replenished in later transactions, leading to stabilization of the transaction failure rate around an upper limit.

4.3.2 Total Transaction Fee Under High Balance

Unlike previous transaction failure rate experiments, no transaction fee will be deducted for transfers that fail, and the final result will be affected by the transaction failure rate. This time, we assume that every transaction will be successful. We will do this by significantly increasing the initial balance of the users.

Similar to the previous results in Fig. 4.6, BLN also outperforms the lightning network in terms of the sum of transaction fees. Because the TPC is now the only intermediary node for all transactions, the payment path is greatly shortened. In addition, the transaction fee charged by each intermediary node is fixed at 0.1 in the

experiment. The payment path length of each transaction in the lightning network is greater than or equal to that of the TPC.

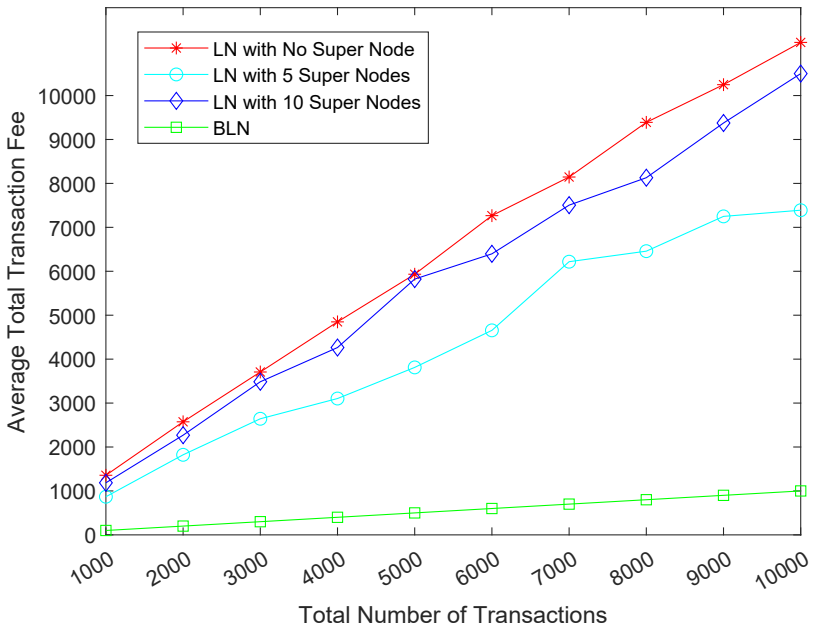


Figure 4.6: Total Transaction Fee (Lightning Network vs. BLN)

Table 4.3: Parameters in the Transaction Fee Experiment

Parameter	**value**
Node Num	1000
Traffic Test Num	100
Super Node Num	5,10
Transaction Fee Amount	0.1
Seller Num	10
Buyer Num	990
Max Neighbour	4
Max Transaction Amount	15
Min Transaction Amount	5
Initial Balance in Channel	1000000

The existence of a super node also affects the final result. Because super nodes shorten the payment path for a small proportion of transactions, the sum of transaction fees is slightly better than that of the lightning network with a balanced graph.

Table 4.3 summarizes the specific value of each parameter used in the transaction fee experiment.

4.3.3 Transaction Processing Time

In terms of transaction processing, when we consider the concepts of buyer and seller, TPC needs to process at least two payment channels for each transaction request. Specifically, it needs to update at least two commitment transactions - one between the buyer and TPC, and another between the seller and TPC. The TPC server has several steps that must be completed to update the commitment transactions:

1.Creating a new commitment transaction

2.Exchanging signatures with the counterparty

3.Voiding the old commitment transaction by exchanging revocation key with the counterparty

4.Recording the commitment transaction in storage

The processing time of these steps is mainly related to the Internet speed at the time, so to quantify the data, we default to assuming that it takes 20 time units to complete the four steps in the experiment.

Assuming that all transaction processing occurs simultaneously on one TPC server, the server needs to sort all transaction requests and process them in order. Therefore, the total processing time is linearly related to the number of transactions.

However, if we use the Redis cluster, each transaction processing will be distributed across different servers. In addition to the four required steps mentioned above, the server needs to determine which server is responsible for processing the request. When the TPC determines the server responsible for handling the buyer or seller, the corresponding server will update the commitment transaction. So we need to use a hash algorithm (CRC-16) to calculate the following additional steps:

1.Find the buyer's data slot by buyer ID (0.5 time units).

2.Find the server responsible for the buyer slot (0.5 time units).

3.Find the seller's data slot by seller ID (0.5 time units).

4.Find the server responsible for the seller slot (0.5 time units).

Because each Redis cluster server processes transaction requests simultaneously, when calculating the time required to process all transaction requests in the experiment, we will find the server that processes the most transaction requests (worst case) and use its processing time as the final measure.

Therefore, the formula for calculating the final processing time of TPC with 1-server is:

Time to update commitment transaction(20 time units) * the number of transaction * 2(buyer and seller commitment transactions)

The formula for calculating the final processing time of TPC with Redis cluster is:

The maximum number of update commitment transactions on server * time to update commitment transaction(20 time units) + find the server corresponding to each buyer and seller in each transfer(2 time units)

To be noted, in BLN with a Redis cluster, if the slot of the buyer and seller's data is updated by the same server, we will count that server as having processed two update commitment transactions when finding the maximum number of update commitment transactions on the server.

In Fig. 4.7, the advantage of BLN with Redis Cluster comes from having multiple servers able to process transaction requests together. Although there are additional steps to find out which server is responsible for updating commitment transactions, the total time is still much less than that of BLN with 1 server. Much of this gap comes from the time it takes to update commitment transactions. That said, the more time it takes to update a commitment transaction, the greater the gap between the two.

The difference between BLN with the original Redis cluster and BLN with the revised Redis cluster is that the revised Redis cluster distributes transaction requests to all 10 servers in one cluster, while the original Redis cluster focuses transaction requests on 2 master servers among 10 servers. That means the pressure on one server

in the revised Redis cluster is five times less than the pressure on the original Redis cluster.

Table 4.4 summarizes the specific value of each parameter used in the transaction processing time experiment.

Table 4.4: Parameters in the Transaction Processing Time Experiment

Parameter	**value**
Node Num	1000
Traffic Test Num	100
Cluster Num	10
Server Num Per Cluster	10
Master Server Per Cluster	2
Slave Server Per Cluster	8
Time Cost of Updating Commitment Transaction	20 (time units)
Time Cost of Finding Server	2 (time units)

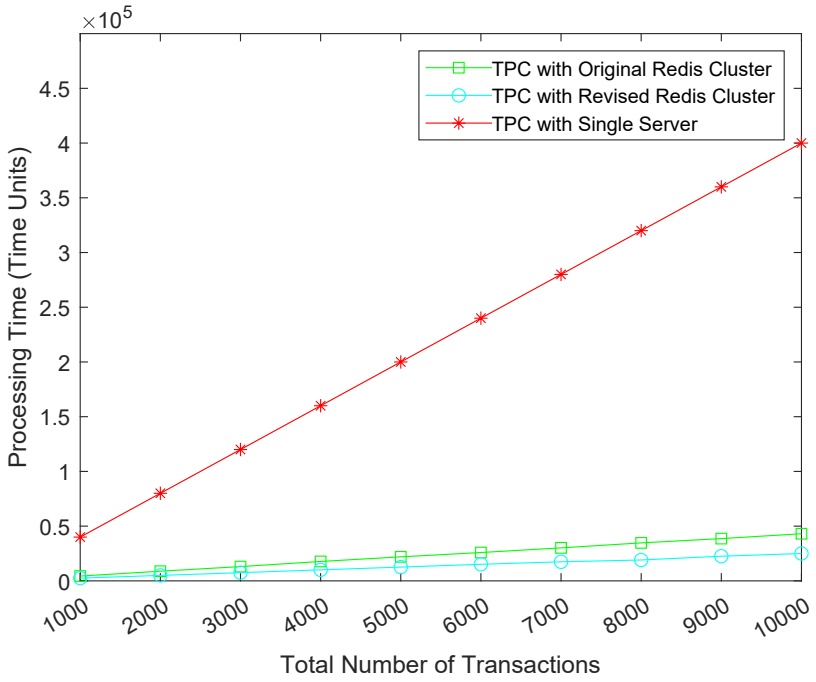


Figure 4.7: Transaction Processing Time (TPC with Redis Cluster vs. TPC with 1 Server)

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We propose the TPC business model to solve the following problems: Firstly, in the original Lightning Network, the sender and receiver of the transaction have different rights. Once a transaction is initiated, the sender can only get a refund until the HTLC expires. As a result, in an online shopping scenario, the receiver can collect the transferred money from the sender but falsely claim that the merchandise has been shipped. With no third-party oversight, no one can prove that the receiver has lied. In contrast to the HTLC used by the Lightning Network, the TPC business model determines the private key of the HTLC by the sender, guaranteeing the rights of the sender. TPC acts as a supervisor and also guarantees the rights of the recipient (can collect money after shipment). Moreover, the Lightning Network is only suitable for buying goods in physical stores, while the TPC business model extends the form of shopping to online shopping.

Additionally, when TPC has sufficient funds for payment channels, the entire business can support large transactions, which solves the problem that the Lightning Network can only support small payments. Furthermore, when TPC is the only intermediate node, all payment paths are shortened. In other words, the transaction fee per transfer is reduced for the user.

Roughly speaking, TPC resembles a centralized banking system, but there is one fundamental difference between them. In a centralized banking system, financial transactions are processed and recorded by a central authority, which maintains a ledger of all account balances and transactions, and clients do not have the ability to change the ledger of transactions. It is theoretically possible for a centralized banking system to maliciously change a user's account balance.

In TPC, both the client and TPC store a current ledger locally, which is the commitment transaction. The commitment transaction will not be valid if it does

not contain the signature from both sides. In other words, it makes no sense to fake the current balance because the counterparty will not sign the forged commitment transaction. The existence of RSMC in the commitment transaction ensures that malicious parties can be punished if they commit malicious acts, and all of their balance in the payment channel will be sent to the counterparty's personal bitcoin address. Both sides take equal responsibility, which is not the case in a centralized banking system.

Further more, to ensure the stability of TPC, we continue to use the Redis cluster function and make some improvements to ease the pressure of read and write operations on the master node, while also saving some storage space from redundant replicas.

Redis is selected to show that in-memory database can be used to speed up transaction processing at TPC. While other in-memory databases could also be used, Redis is the one chosen in our research.

5.2 Future Work

5.2.1 Trust Mechanism of TPC

In the previous discussion, we assumed that TPC was fully trusted. However, there is currently no mechanism for monitoring TPC. There are no penalties for TPC accidents or malicious mistakes. Although if TPC loses credibility, no one will use the business model again, the losses suffered by the users cannot be repaired. Therefore, it is necessary to establish a new trust mechanism for TPC. TPC can be constrained by a contract or other means so that it cannot make mistakes. If TPC makes a mistake, it should be subjected to reasonable punishment. We plan to refer to the proof of stake in Ethereum [45] [46]. That is, multiple committees will be set up to monitor the status of TPC and vote to report on the status of the data that the current committee members see [47] [48].

5.2.2 Financial Commitment of TPC

In the case of a large transaction, there is pressure on the funds from the user side all the way to the TPC. While it may be user-friendly, the TPC needs sufficient startup funds. However, the only source of income for TPC is the transaction fee charged as the intermediary node. As a result, TPC may increase transaction fees to boost revenue. Since TPC is the only intermediary node, it may raise transaction fees too high. Additionally, since the transaction method is HTLC like the traditional lightning network, a significant amount of TPC's funds are locked in HTLC until the transaction is completed, further increasing the financial pressure on TPC. This problem can be alleviated by having TPC increase the frequency of broadcasting commitment transactions.

5.2.3 Customer Dispute

When a transaction unexpectedly expires due to an unknown reason, TPC can only roughly determine who should bear the responsibility based on how far the transaction has progressed. However, the actual situation may be more complicated, and this approach may lead to an incorrect penalty. Therefore, when an error occurs, TPC needs to gather more precise information before issuing a penalty. This can be achieved by supporting real-time network communication with customers. Alternatively, TPC can set up a customer service channel to collect feedback from users.

5.2.4 Partially Decentralized BLN

If we want to add some decentralized features, one potential approach would be to adopt blockchain for supply chain management, allowing suppliers to input and track data related to the sourcing and delivery of merchandise. This could increase efficiency, reduce fraud, and improve accountability.

Another approach could be to adopt more collaborative decision-making processes, such as through the use of consensus-based decision-making models or employee-owned cooperatives. However, a fully decentralized BLN may not be feasible or desirable, as certain centralized functions may be necessary for the company's success

and growth.

Bitcoin is decentralized because it is a peer-to-peer network of nodes, which means that there is no central authority or server controlling the network. Transactions are verified and recorded by nodes in the network, called miners, who compete to add new blocks of transactions to the blockchain by solving complex mathematical puzzles. However, BLN is not decentralized because it requires a supervisor for all transactions. By sacrificing some decentralized characteristics, we adopted a supervision mechanism.

Bibliography

[1] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.", 2014.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[3] Karl Aberer and Manfred Hauswirth. An overview of peer-to-peer information systems. In *WDAS*, volume 14, pages 171–188, 2002.

[4] Nicola Dimitri. Transaction fees, block size limit, and auctions in bitcoin. *Ledger*, 4, 2019.

[5] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[6] Luqin Wang and Yong Liu. Exploring miner evolution in bitcoin network. In *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*, pages 290–302. Springer, 2015.

[7] Jeremy Nelson. *Mastering redis.* Packt Publishing Ltd, 2016.

[8] Xun Brian Wu and Weimin Sun. *Blockchain Quick Start Guide: A beginner's guide to developing enterprise-grade decentralized applications.* Packt Publishing Ltd, 2018.

[9] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers 16*, pages 399–414. Springer, 2012.

[10] Lynne Schleiffarth Burks, Andrew Elliot Cox, Kiran Lakkaraju, Mark James Boyd, and Ethan Chan. Bitcoin address classification. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2017.

[11] Yu-Jing Lin, Po-Wei Wu, Cheng-Han Hsu, I-Ping Tu, and Shih-wei Liao. An evaluation of bitcoin address classification based on transaction history summarization. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 302–310. IEEE, 2019.

[12] Kentaroh Toyoda, P Takis Mathiopoulos, and Tomoaki Ohtsuki. A novel methodology for hyip operators' bitcoin addresses identification. *IEEE Access*, 7:74835–74848, 2019.

[13] Massimiliano Sala, Domenica Sogiorno, and Daniele Taufer. A small subgroup attack on bitcoin address generation. *Mathematics*, 8(10):1645, 2020.

[14] Abba Garba, Zhi Guan, Anran Li, and Zhong Chen. Analysis of man-in-the-middle of attack on bitcoin address. In *ICETE (2)*, pages 554–561, 2018.

[15] Nirupama Devi Bhaskar and David LEE Kuo Chuen. Bitcoin mining technology. In *Handbook of digital currency*, pages 45–65. Elsevier, 2015.

[16] Harris Brakmić and Harris Brakmić. Bitcoin script. *Bitcoin and Lightning Network on Raspberry Pi: Running Nodes on Pi3, Pi4 and Pi Zero*, pages 201–224, 2019.

[17] Stefano Bistarelli, Ivan Mercanti, and Francesco Santini. An analysis of non-standard bitcoin transactions. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 93–96. IEEE, 2018.

[18] Maoning Wang, Meijiao Duan, and Jianming Zhu. The trojan message attack on the pay-to-public-key-hash protocol of bitcoin. In *Blockchain Technology and Application: Second CCF China Blockchain Conference, CBCC 2019, Chengdu, China, October 11–13, 2019, Revised Selected Papers 2*, pages 196–209. Springer, 2020.

[19] Joachim Zahnentferner. An abstract model of utxo-based cryptocurrencies with scripts. *Cryptology ePrint Archive*, 2018.

[20] Sachchidanand Singh and Nirmala Singh. Blockchain: Future of financial and cyber security. In *2016 2nd international conference on contemporary computing and informatics (IC3I)*, pages 463–467. IEEE, 2016.

[21] Craig S Wright. A proof of turing completeness in bitcoin script. In *Intelligent Systems and Applications: Proceedings of the 2019 Intelligent Systems Conference (IntelliSys) Volume 1*, pages 299–313. Springer, 2019.

[22] Zijian Bao, Wenbo Shi, Saru Kumari, Zhi-yin Kong, and Chien-Ming Chen. Lockmix: a secure and privacy-preserving mix service for bitcoin anonymity. *International Journal of Information Security*, 19:311–321, 2020.

[23] Rostislav Skudnov. Bitcoin clients. 2012.

[24] Jongbeen Han, Mansub Song, Hyeonsang Eom, and Yongseok Son. An efficient multi-signature wallet in blockchain using bloom filter. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 273–281, 2021.

[25] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. *Scalable o-chain instant payments*, 2015.

[26] Andreas M Antonopoulos, Olaoluwa Osuntokun, and René Pickhardt. *Mastering the Lightning Network.* " O'Reilly Media, Inc.", 2021.

[27] Shilan Yang, Huaimin Wang, Wei Li, Wei Liu, and Xiang Fu. Cvem: A cross-chain value exchange mechanism. In *Proceedings of the 2018 International Conference on Cloud Computing and Internet of Things*, pages 80–85, 2018.

[28] Liping Deng, Huan Chen, Jing Zeng, and Liang-Jie Zhang. Research on cross-chain technology based on sidechain and hash-locking. In *Edge Computing–EDGE 2018: Second International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings 2*, pages 144–151. Springer, 2018.

[29] Bin Yu, Shabnam Kasra Kermanshahi, Amin Sakzad, and Surya Nepal. Chameleon hash time-lock contract for privacy preserving payment channel networks. In *Provable Security: 13th International Conference, ProvSec 2019, Cairns, QLD, Australia, October 1–4, 2019, Proceedings 13*, pages 303–318. Springer, 2019.

[30] Colin Boyd, Kristian Gjøsteen, and Shuang Wu. A blockchain model in tamarin and formal analysis of hash time lock contract. In *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[31] Snoviya Dcunha, Srushti Patel, Shravani Sawant, Varsha Kulkarni, and Mahesh Shirole. Blockchain interoperability using hash time locks. In *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems: MCCS 2020*, pages 475–487. Springer, 2021.

[32] Dimaz Ankaa Wijaya. Extending asset management system functionality in bitcoin platform. In *2016 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, pages 97–101. IEEE, 2016.

[33] Maxwell Dayvson Da Silva and Hugo Lopes Tavares. *Redis Essentials*. Packt Publishing Ltd, 2015.

[34] Josiah Carlson. *Redis in action*. Simon and Schuster, 2013.

[35] Tiago Macedo and Fred Oliveira. *Redis cookbook: Practical techniques for fast data manipulation.* " O'Reilly Media, Inc.", 2011.

[36] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in context*, pages 1–18, 2008.

[37] Jian Yang and Hong Shen. Blockchain consensus algorithm design based on consistent hash algorithm. In *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 461–466. IEEE, 2019.

[38] Wang Zhong, Xiandong Zheng, Wenlong Feng, Mengxing Huang, and Siling Feng. Improve pbft based on hash ring. *Wireless Communications and Mobile Computing*, 2021:1–9, 2021.

[39] Wei Wang, Xiaojing Yao, and Jing Chen. A map tile data access model based on the jump consistent hash algorithm. *ISPRS International Journal of Geo-Information*, 11(12):608, 2022.

[40] Yan-Fei Li, Hong-Jun Wang, and Hua Li. A rfid algorithm based on cyclic redundancy check. In *2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication*, pages 278–281. IEEE, 2009.

[41] Damiano Carra and Pietro Michiardi. Memory partitioning in memcached: An experimental performance analysis. In *2014 IEEE International Conference on Communications (ICC)*, pages 1154–1159. IEEE, 2014.

[42] Daniel House, Heng Kuang, Kajaruban Surendran, and Paul Chen. Toward fast and reliable active-active geo-replication for a distributed data caching service in the mobile cloud. *Procedia Computer Science*, 191:119–126, 2021.

[43] Sharon Rithika. Redis data replication simplified 101. https://hevodata.com/learn/redis-data-replication/#1.5, May 2022.

[44] William L. Hamilton. *Traditional Graph Generation Approaches*, pages 107–111. Springer International Publishing, Cham, 2020.

[45] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.

[46] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*, pages 357–388. Springer, 2017.

[47] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[48] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv preprint arXiv:2003.03052*, 2020.