Distributed Bitcoin Miner

University of Victoria
CSC462/562
Bing Gao
Jinghan Jiang
Rui Liu
Gloria Wang
Joshua Park

Introduction **4**
1.1. Background 4
1.1.1. Blockchain 4
1.1.2. Bitcoin mining 4
1.1.3. The hash algorithm 5
1.1.4. The digital signature scheme 5
1.2. High-level of the network 6
1.2.1 The bitcoin network 6
1.2.2 The mining process 6
1.3. Application scenarios 7

Literature Review **9**
2.1. LR by Gloria 9
2.2. LR by Rui Liu 10
2.3. LR by Jinghan 12

System Design **13**
3.1. Components of system 13
3.2. High level diagram 14

Implementation and Development **15**
4.1. List of technologies 15
4.2. Implementation of Live Sequence Protocol (LSP) 15
4.2.1. Basic Concepts and Operations 15
4.2.2. Detailed Implementation 17
4.2.3. LSP Evaluation 21
4.3. Implementation of bitcoin mining system 22
4.3.1. Client 22
4.3.2. Server 23
4.3.3. Miner 27
4.3.4. Message 28
4.3.5. Hash 29

Preliminary Evaluation **29**

Evaluation of Tradeoffs **30**
6.1. LSP vs. TCP network comparison 30
6.2. Evaluation of Distributed system 32
6.3. Failure Handling 32
6.3.1. Logic of failure handling 32
6.3.2. Tests of failure handling 35

Conclusion **35**

Project Repository **36**

Demo video **36**

References **36**

Appendix **38**

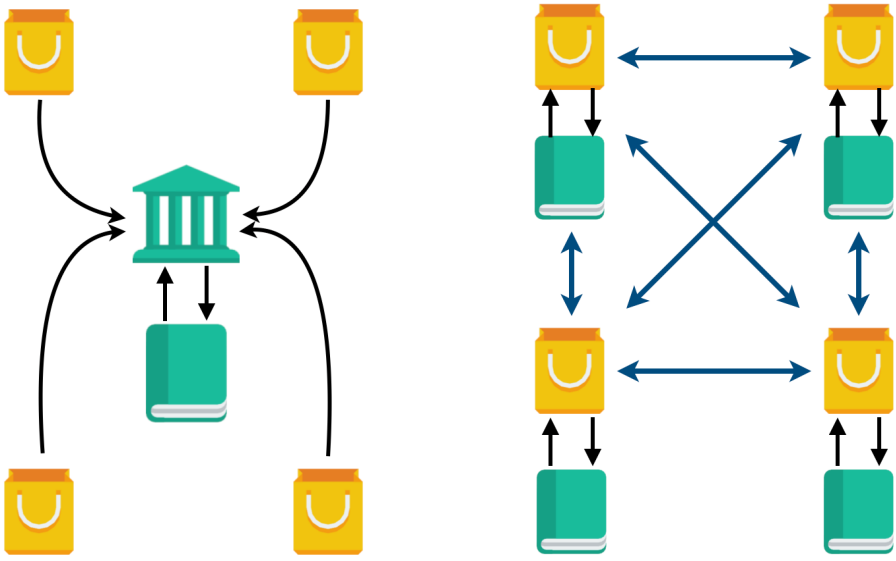
1.Introduction

1.1. Background

The following three sections provide a basic introduction about Blockchain, Bitcoin mining and the hashing algorithm.

1.1.1. Blockchain

Blockchain currently receives more and more attention from many aspects, but what is blockchain? In a simple way to explain it, is a series of immutable record of data appended with timestamps [1]. More precisely, a blockchain contains a growing block of records and bounded using cryptographic principles. Inside of each block, it includes a hash value of the previous block's timestamp and transaction data. With the technology of blockchain, it allows distributed systems to hold immutable data in a secure and encrypted way. All data in blockchain can be managed by any computers in the world instead of any third party or centralized currencies [1]. In this way, the information in it can be shared and seen by everyone. Apart from that, it reduces the cost during transaction since a blockchain carries no transaction fee [2].



(a) traditional currencies (b) blockchain

Figure 1.1 Comparing Traditional Currencies with The Blockchain

1.1.2. Bitcoin mining

Bitcoin is one the most popular example of blockchain usage. If we imagine Bitcoin as gold, then there should be someone called miner to find them. Same as gold, Bitcoin is a limited resource. At the first 4 years, miners can be rewarded by

newly-created Bitcoins and transaction fees for every 10 minutes which is 50 Bitcoin [3]. Then, this number will decrease to 25 for the next four years and so on. By doing a simple calculation it is not hard to know that there are

$$50 \times 6 \times 24 \times 365 \times 4 \times \left(1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \cdots\right)$$

which is 21 million Bitcoin by year 2040 [3]. Once all of the new coins have been mined, Bitcoin mining stops immediately. After that, the currency Bitcoin will be used for transaction. Therefore, as time goes on, Bitcoin will become more and more valuable.

1.1.3. The hash algorithm

Hashing is the process of turning any data into a cryptographic output through some mathematical algorithm. SHA-256 algorithm is one of the strongest hash functions and commonly used in Blockchain [4]. It generates a unique and fixed size 256-bit hash. Hashing is a one-way algorithm, so it is impossible to generate from hash value back to the original data. Different input will generate different results. Even a little bit changes will change everything [5]. For example Table 1.1:

Data	SHA-256 value
Hello	185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
hello	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Table 1.1 By SHA-256 calculator [4]

At the same time, the same input data will always produce the same result. In this way, SHA-256 has high security and privacy.

In blockchain, hash values are used to represent the current state. Every input contains every transaction that has happened on a blockchain. The first block of a blockchain called genesis block which contains validate transactions and timestamp [5]. It produces a unique hash on top of blockchain. This hash and all the new transactions that are being processed will be treated as an input to create a new hash value for the next block in the chain. Therefore, changing any record in previous blocks will change all the hash values. This is how blockchain protects itself from hacking.

1.1.4. The digital signature scheme

A digital signature scheme consists of the following three algorithms:

- **(sk, pk) := generateKeys(*keysize*)**The generateKeys method takes a key size and generates a key pair. The secret key *sk*is kept privately and used to

sign messages. *pk* is the public verification key that you give to everybody. Anyone with this key can verify your signature.

- **sig := sign(*sk*,*message*)**The sign method takes a message and a secret key, *sk*, as input and outputs a signature for *message* under *sk*
- **isValid := verify(*pk*,*message*,*sig*)**The verify method takes a message, a signature, and a public key as input. It returns a boolean value, *is Valid*,that will be ***true*** if *sig* is a valid signature for *message*under public key *pk*,and ***false*** otherwise.

1.2. High-level of the network

In this chapter, we describe the components of the bitcoin network, and the mining procedure in a high-level.

1.2.1 The bitcoin network

In the bitcoin network, there is a global log, named as blockchain. Every node, i.e., the users in the network, can get access to the public blockchain. A user can send or receive bitcoins to another user. All the transactions are recorded in the blockchain in units of blocks.

Besides, every node can be a bitcoin miner. Each miner that successfully solves a cryptopuzzle can generate a new transaction block and get a reward of Bitcoins. A valid block contains the solution of the cryptopuzzle, the hash of the previous block, the hash of the transactions in the current block, and a bitcoin address which is to be credited with a reward for solving the cryptopuzzle [6]. Such a process is called bitcoin mining. Any miner in the network may generate and add a valid block to the blockchain and publish it to all other miners. A miner can also gather a mass of nodes to get huge computational resources for solving the crytopuzzle.

1.2.2 The mining process

Bitcoin mining is a decentralized process. It can confirm transactions, create a new block and add it to the current blockchain. There are mainly four steps in the mining process.

A. Transaction Verification

Upon receiving a new bitcoin transaction, miners should validate it by verifying previous transactions to make sure the sender actually has enough money and to prevent the double-spending problem. Double spending means that a user illicitly spends the same money twice.

B. Transaction Bundling

The transaction should be bundled in a data block. A data block is composed of several components. The head of a block contains the version number of the software, the hash of the previous block, the root hash of the Merkle tree, the time in

seconds since 1970–01–01 T00: 00 UTC, the goal of the current difficulty and the nonce. The body of a block contains all transactions that are confirmed with the block.

C. Proof-of-work Problem Solving

Though any miner can bundle the transaction into a new block, only one block can be accepted by the whole bitcoin network. To be specific, a new valid block must contain a proof-of-work (PoW). The PoW requires miners to find a number called a nonce, such that when the block content is hashed along with the nonce, the result is numerically smaller than the network's difficulty target. [7]

$$H(nonce \parallel prev\_hash \parallel tx \parallel tx \parallel ... \parallel tx) < target$$

The nonce is easy to verify but extremely hard to choose. A miner may try a mass of different values. The block of the first miner who meeting the difficulty target, i.e., solve the proof-of-work problem, is considered valid.

D. Blockchain Appending and Propagating

The successful miner first finding the new valid block can append the block to the blockchain. In the meanwhile, the miner gets transaction fees and newly created bitcoins as rewards. The new block is quickly published to all nodes without requiring central oversight. A blockchain fork is happens when a blockchain diverges into two potential paths forward. The longest blockchain in the network is considered valid, as shown in figure 1.2.

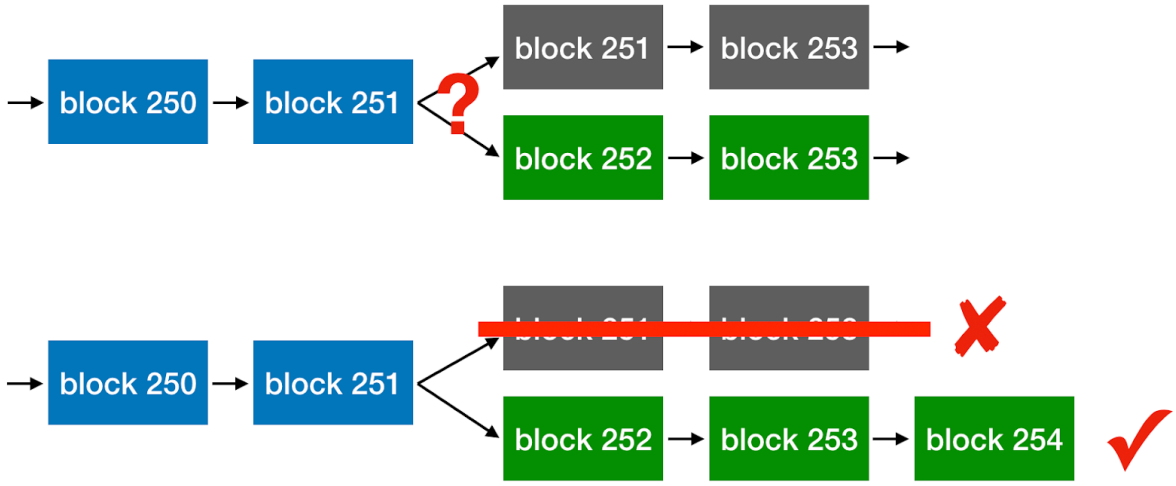


Figure 1.2 Blockchain Fork

1.3. Application scenarios

Our protocol implementation guarantees the distributed operations in the following scenarios can be proceed smoothly.

Scenario 1

1. Bitcoin can be seen as an investment product nowadays. Now Tom wants to have some bitcoins as his assets. He knows the most effective way to get

bitcoins is to earn it by “mining”. So Tom sends a “join” application to the server to let it know that he is ready to accept the mining tasks.

2. The server always breaks a task into more manageable-sized jobs and then assign them to the available miners.
3. Now Tom starts mining as one of the available miners, if he could calculate the correct answer of a task (i.e., The mine is dug up.by him), he will get a certain amount of bitcoins in return.
4. After a fixed period, the server will broadcast who firstly work out the required answers (i.e., get the mine) and the new transaction records will be appended to the original blockchain.
5. After receiving the broadcast from server, Tom will immediately stop the computational task in hand and accept a new mining task to get the chance of earning bitcoins.

Scenario 2

1. Mia is a long-term user in the Bitcoin market. One day she makes several new transactions with others in the market. As a result, her transactions need to be recorded in time, so a new request will be sent to the server.
2. The server will collect all the final results from various miners and find the correct one from them. Finally, it will send the final result to Mia to let her know the transaction is recorded successfully.
3. After receiving the response from the server, Mia should print a standard output to confirm the answer has been found. If she loses connection after her request is sent, Mia also needs to print a message to inform this thing.

- Challenges:
    - ***Identity authentication***, which means to confirm the transactions involve user A are actually conducted by A.
    - ***Double payment problem.*** For example, user A only has 10 bitcoins, so when he tries to transfer 10 bitcoins to both user B and C at the same time, only one of the two transactions can be made successfully.
    - ***History rewriting.*** We should ensure that a transaction which has occurred and connected to the blockchain cannot be deleted or modified by anyone.
- Solutions：
    - Asymmetric encryption
    - Traceability on the block chain
    - The “longest chain” principle

2. Literature Review

2.1. LR by Gloria

Christian [8], In this paper, they propose a new Bitcoin Blockchain system which enables strong consistency. It presents that the traditional Bitcoin Blockchain system is eventual consistency. In order to realize real-time exchange, they need to abandon the weak consistency and use strong consistency. They build a strong consistency system called PeerCensus which acts like a certification authority. It manages peer identities in a peer-to-peer network. They analyze the system by calculating the optimizations and prove the theorems. Apart from that, they set up protocol and evaluate it layer by layer. The result shows that PeerCensus is in a secure state with high probability. The only problem is that this paper does not give any examples in the real world. We can make an experiment based on this system to prove their ideas in the future.

Arthur [9], In this research, it points out that recent studies found the performance of PoW based blockchains cannot be enhanced without impacting its security. In this way, they provide a quantitative framework to analyse the security and performance of the Blockchain. It introduces the concept PoW in Blockchains and build security model based on Markov Decision Processes for double-spending and selfish mining. Their work provides a way to compare the security and performance with different parameters. The test results are given by changing the parameters of the system. However, it is hard to capture the best case since there are too many parameters need to be considered. In the future, we can allow merchants to take the security provision when accepting transactions for double-spending and help miners against selfish mining.

Hiroki [10], This is a very interesting paper, it proposes how to use Blockchain technology for recording contracts and sent between people. Instead of one-way transaction in traditional Blockchain, it changes to bi-directional. They made a protocol where a transaction is used as evidence of contractor consent. In order to protect the privacy of the contract, the protocol involves key pairs for both e-signature and contract data. However, this method only based on theorem without an actual cryptocurrency. In a real example, there is a possibility that devices or network break down, then the content of contracts may be hacked or unsent. In the future, we can implement a test for this method.

Raja [11], In this paper, it studies how to use Bitcoin addresses in securing the sender and recipient address for transactions on Bitcoin. They use python programming language to make an application which includes interfaces and transactions. The Bitcoin addresses are created by using cryptography type SHA-256 and Base58Check. By encoding of Base58 which has a built-in error code,

the system will calculate the checksum and compare it with the one in the code. The only problem with this method is it does not check the running time for hash.
Christian [12], This paper describes a consensus protocol of a permissioned blockchain. Compared to the traditional permissionless blockchain everyone can use and share the state, permissioned blockchain can control who can issue transactions. They evaluate the system by comparing the resilience and trustworthiness of different protocols. There are many mathematical analyses within evaluations part. The only limit is all protocols presented in this paper assume independence among the failures, selfish behavior and subversion of nodes. In this way, we could extend this protocol to more complex fault assumption in the future.

2.2. LR by Rui Liu

The literature review here is based on the five questions required. There is also another paper-style literature review, which Yvonne asked me to include it in the project. I put it at the end of the report as an attachment. Thanks.

A. Majority is not enough: Bitcoin mining is vulnerable [13]
1. In most research [14,15], it is commonly accepted that the Bitcoin network is sound and secure when a majority of the participants are honest.
2. Eyal et.al. want to show that the Bitcoin mining protocol is not incentive-compatible even with majority nodes following the protocol.
3. To prove it, they proposed a new attack with which the Bitcoin system ceases to be a decentralized currency.
4. In this research, the authors used a bitcoin protocol simulator to simulate 1000 miners mining at identical rates and some of them run the selfish-mine algorithm. They modified some parameters and observed that the revenue of each pool member increases with pool size for pools larger than the threshold.
5. It leads to future work on adjusting the Bitcoin network to adopt an automated machine that can thwart selfish miners.
B. Bitcoin: A peer-to-peer electronic cash system [16]
1. In this paper, the author proposed a solution to the double-spending problem for online payments.
2. In this work, an ongoing chain of hash-based proof-of-work is proposed. The longest chain proofs that it came from the largest pool of CPU power and is considered to be valid so that the double-spending problem is solved.
3. Some researches [17] used digital signature to ensure the security of online payment, but the idea in this paper is pretty innovative. The proof-of-work system used in this work is similar to Adam Back's Hashcash [18].
4. The author implements it with C language and analyzed the probability of an attacker generating an alternate chain faster than the honest chain. The results show that it is secure enough.

5. However, the huge computational resources is a main problem of the design, which should be considered in future work.

C. Difficulty control for blockchain-based consensus systems [19]

1. To avoid finding unlimited bitcoins, the network adjusts the proof-of-work difficulty dynamically to regulate the mining frequency. To update the difficulty while ensuring stable average block times over a long time, Kraft proposed a new method.
2. The author modified the computing formulations to control the difficulty and proposed a rule that each miner should provide a specific value, which is related to the hash rate, to the newest mined block. The new updating protocol can improve the accuracy of the resulting rate for exponentially increasing hash rate.
3. There are much related work [20,21] on bitcoin mining but focus on the double-spending transaction problem. This paper is the first one focuses on the difficulty updating.
4. With the new updating approach, a simulation of the mining process was done. In the simulation, how the average block time behaves is compared with the original updating approach in different scenarios.
5. In the future, to model simulation with random fluctuations in the hash rate instead of a deterministic way, can be considered.

D. Data Transmission Scheme Considering Node Failure for Blockchain [22]

1. Communication efficiency is pretty important in the blockchain network. Thus, node failures should be considered carefully.
2. In this research, the authors use a response threshold level to detect node failures and construct a communication tree to organize nodes. The proposed scheme shorten the validation time of transactions and improve the resistance to node failures.
3. In related work [23,24], to improve the communication efficiency, tree topology stricture is often used without considering the node failures. Paper [25] propose a node failure recovery approach based on a localized hybrid timer.
4. Some simulation experiments are done with different percentage of failure nodes. The concurrent communication time and average end-to-end delay are collected and compared.
5. This work implies a future work to find an appropriate approach to replace the failed node and to implement it in the real world.

E. The blockchain-based digital content distribution system [26]

1. The main purpose of this paper is to design a distributed digital content delivery system and to defend the pirate attacking.
2. The ideal of the digital content distribution system was released in 1983 [27]. The core idea in this research is that the usage model would be installed by way of compensation for the ownership model. The authors also proposed a system based on the metadata and ID control mechanism [28].

3. In this research, the authors design a public and private key operation system based on the blockchain technique. Each block includes the rights information. Different from the research mentioned above, this system can be operated by the rights holder themselves.
4. The author checked the system at the semi-closed forum with 100 people and collect comments from them. However, except for the analysis of comments, there is no other evaluation of the system.
5. The future work is to design the system more sophisticated as the copyright problem has been a hot topic for many years and every country has the copyright law.

2.3. LR by Jinghan

Bitcoin is a decentralized peer-to-peer digital currency, which is the most popular example that uses blockchain technology . At present, the researches on the implementation and operation of the Bitcoin market can be roughly summarized into two categories: how the block chain technology adapts to the features and needs of Bitcoin market; how to meet the different levels of consistency in the Bitcoin market.

In the first category, the authors of **[29]** try to make a comprehensive description on the blockchain technology in both technological and application perspectives. Before that, there are some researches focus on the using of blockchain in different application, but few of them summary the application situation of this technology in a thorough way. To achieve the goal, the authors do a number of related readings, and then summarize them in: architecture and key characteristics; the typical consensus algorithms; comparison of different protocols; the typical blockchain applications; challenges and future directions in blockchain. This paper does evaluation mainly by code analysis and performance comparison of different protocols. In the future, they put their eyes on the development of blockchain in five aspects: blockchain testing, stop the tendency to centralization, big data analytics, smart contract and artificial intelligence.

In addition, **[30]** shows us the process of block chain technology from being proposed to continuous development by time in the current financial as well as non-financial industry. The author introduces the process of adopting blockchain in Bitcoin market in detail by showing a number of flow-charts and then explain them. In the end, the authors suggest that the adoption situation should be continually observed in a decade or two because most of the start-ups will fail with few winners.

[31] recently presents a measurement study on decentralization metrics in operational systems which rely on the novel measurement techniques and the application of well-established internet measurement techniques. To do so, it relies on novel measurement techniques to obtain application layer information using the

Falcon Network, and the authors collect data by examining the application and recording its weekly distribution over a continuous period of time. By the comprehensive data collection, the authors are able to evaluate the bandwidth, latency and geographical distribution of different time periods by various comparison graphs. With the tremendous growth of Bitcoin market, the authors of this paper is interested to take another metric: the volatility of mining rewards into account in their further measurement.

Those mentioned works only discuss and evaluate this technology at the application level. In the second category, the researches focus on the more detailed level of protocol implementation in blockchain.

To achieve consensus and consistency in a distributed system, **[32]** adopts a collective signing method to scale protocols to large consensus groups and enable clients to verify operation commitments efficiently. For the reason that the "ByzCoin" model proposed by this paper is a novel Byzantine consensus protocol that uses collective signing to commit Bitcoin transactions.It also does a lot of introduction on Byzantine consensus protocol.In the evaluation part, this paper focuses on the consensus latency and transaction throughput for different parameter combinations (# miners, block size, etc). The results show that the ByzCoin model can increase Bitcoin market's core security guarantees and also enables high scalability and low transaction latency in it.And the paper also leaves a more thorough analysis of some irrational attacks under the ByzCoin model as their future work.

From a broader range, **[33]** provides an overview of consensus models as adopted by popular blockchain platforms and analyzes their merits and demerits. It also gives the conclusion that: when looking at Blockchain to solve a business problem, it is imperative to look at the type of applications the platform expects to cater to and the threats it expects to before determining the platform and the consensus model to use. For evaluating, it provides a comprehensive table to compare the popular blockchain consensus mechanisms in terms of transaction rate, cost, fault tolerance, and so on. This may mean that in the future, more strategy of adopting consistency models in combination can be introduced.

3.System Design

3.1. Components of system

This distributed system will consist of the following three components:

- Client: It sends a user-specified request to the server. After that, it receives and prints the result, and then exits.
- Miner: It accepts requests from the server continually, computes all hashes over a given range of nonces, and then tells the server the result.

- Server: It manages the entire Bitcoin cracking enterprise. The server includes any number of workers available and receives any number of requests from any number of clients. For each request from client, it splits request into smaller chunks of jobs and distributes them to its available miners. The server has to wait all responses from workers before sending the final result to the client.

Predefined message types in the Bitcoin distributed system:

Type	Fields	From–To	Use
Join Request Result	Data Lower Upper Hash Nonce	M–S C–S, S–M M–S, S–C	miner's join request send job request report job's final result

Table 3.1 Message Types (C as a client, M as a miner, S as the server)

3.2. High level diagram

Based on above, we can design our distributed bitcoin miner.

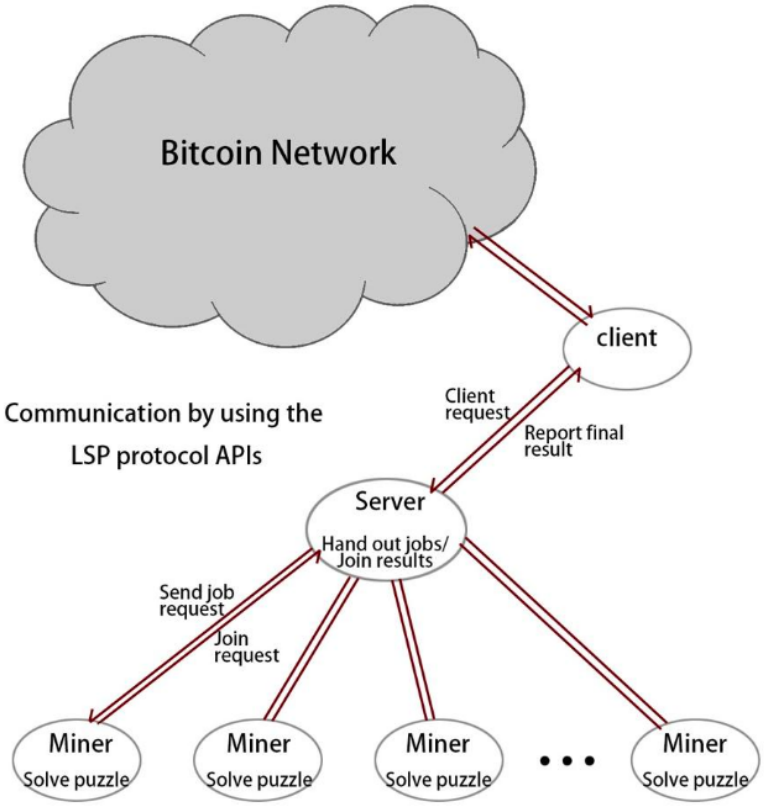


Figure 3.2 System Architecture Diagram

4.Implementation and Development

4.1. List of technologies

- **AFS cluster machines**: https://uit.stanford.edu/service/afs
  AFS (Andrew File System) is a distributed, networked file system that enables efficient file sharing between clients and servers.
- **Live Sequence Protocol** (LSP is based on UDP, we will implement this protocol by ourselves) a homegrown protocol for providing reliable communication with simple client and server APIs on top of the Internet UDP protocol.
- **SHA-256**
  SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash. Hash is a one way function – it cannot be decrypted back.

4.2. Implementation of Live Sequence Protocol (LSP)

In our system, we need to guarantee the smooth communication between different nodes (server, clients) in the low-level internet protocol, which means that the sent packets have the probability to be delayed, dropped, or duplicated. To achieve this goal, we implement the Live Sequence Protocol (LSP).

In LSP, communication between nodes consists of a sequence of discrete messages sent in each direction. In the following part of this section, we will firstly give some basic concepts and operation principles in the implemetantion of LSP. Then, we will introduce the detailed implementation from the perspective of client, server and the designed test respectively. At last, the evaluation of implementing LSP on the local machine will be shown. The result shows that adopting LSP in our bitcoin mining system can bring stable, reliable and efficient distributed communication that meet our expectations.

4.2.1. Basic Concepts and Operations

1. **Building connection**

A connection must be made between the sender and receiver before any data transfer. In our impemetation, each LSP connection is defined by five values:

- Message Type: An integer constant identifying one of three possible message types:
  - Connect: Sent by a client to establish a connection.
  - Data: Sent by either client or the server to transmit information.
  - Ack: Sent by either client or the server to acknowledge a connect.

- Connection ID: A positive, non-zero number that uniquely identifies the client-server
  connection.
- Sequence Number: A positive, non-zero number that is incremented with each data message sent, starting with the number 0.
- Payload Size: The number of bytes of the payload.
- Payload: A sequence of bytes, with a format determined by the application.

2. Sending packet with acknowledgment

Once a connection is established, data may be sent in both directions (i.e. from client to server or from server to client).

In LSP, the communication between client and server is asynchronously, so there is no need to guarantee that packets arrive in the same order as they are sent. However, messages must be acknowledged when received, and processing cannot occur out of order. For example, if the server has not yet received a message with sequence number 5, but received a message with sequence number 6, it must store message 6 until it receives message 5.

3. Epoch (Term)

According to the design discussed above, the LSP so far is not robust. Altough its sequence numbers makes it possible to detect when a message has been dropped or duplicated, if any connection request, data message, or acknowledgment gets dropped, the linkage in either one or both directions will stop working, with both sides waiting for messages from the other.

That is the reason we need to define a duration of epoch on both the clients and servers, which is denoted as δ . Our default value of δ is 2000 ms, although this can be varied.

4. Data size change

We also define a method to ensure the correct transmission by using the size of transmitted data. The designing of this function is inspired by the fact that if the size of the transferred data is not included, and some bytes are lost in transmission, the receiver would not be able to know the loss of data.

So in out LSP, if the size of the received data is shorter than the given size included in the message, the message should be rejected. On the contrary, if the size of the data is longer than the given size, it will be simply truncated to the correct length.

4.2.2. Detailed Implementation

Based on the discussion in Section 4.2.1, we define the client and server api like follows:

- **LSP Client**

Firstly, we define a function NewClient(`Client, error`) to set up and initiate the activities of a client. Like what is shown in the figure below, after a connection with the server is established successfully, this function will return. And it should return a "non-nil" error if a connection cannot be built successfully.

```
serverAddr, err := lspnet.ResolveUDPAddr("udp", hostport)
if err != nil {
        return nil, err
}
conn, err := lspnet.DialUDP("udp", nil, serverAddr)
if err != nil {
        return nil, err
}
c.conn = conn
//fmt.Println("Connects to server(addr is:", serverAddr)
if err := c.sendData(NewConnect()); err != nil {
        return nil, err
}
```

In the Client interface, applications simply read and write data messages to the network by calling the Read() and Write() methods respectively. The connection can be signaled for shutdown by calling function Close().

In our implementation, the Read() function blocks until data has been received from the server and is ready to be returned. It should return a "non-nil" error if either: 1). the connection has been, or 2). the connection has been lost due to an epoch timeout and no other messages are waiting to be returned by Read(). Just like what is said by the "if sentence" in the following figure:

```
func (c *client) Read() ([]byte, error) {
        if c.closed {
                return nil, errors.New("Connection has been closed")
        }
        if c.connLost {
                // Already timed out
                return nil, errors.New("Connection is lost due to timeout")
        }
        c.requestRead <- true
        msg := <-c.responseRead
        if msg != nil {
                return msg.Payload, nil
        } else {
                // Timed out during waiting for message
                return nil, errors.New("Connection is lost due to timeout")
        }
}
```

About the Write() function, it should not block. And it will return a “non-nil” error only if a connection to the server has been lost. And the Close() function should block until all pending messages to each client have been sent and acknowledged.

- **LSP Server**

The API for the server is similar to that for an LSP client, except for a few differences.
When an application calls the NewServer() to set up and initiate a LSP server, unlike NewClient() function, this function should not block. Instead, it should simply begin listening on the specified port and spawn one or more goroutines to handle things like accepting client connections, triggering epoch events at xed intervals, etc. If there is a problem setting up the server, the function should return a non-nil error. As follows:

```
hostport := lspnet.JoinHostPort("localhost", strconv.Itoa(port))
addr, err := lspnet.ResolveUDPAddr("udp", hostport)
if err != nil {
        return nil, err
}
conn, err := lspnet.ListenUDP("udp", addr)
if err != nil {
        return nil, err
}

s.conn = conn
s.epochTicker = time.NewTicker(time.Millisecond * time.Duration(params.EpochMillis))

go s.receiveData()
go s.handleEvents()
return &s, nil
```

The Read() and Write() functions of the Server are similar to those of the Client. However, it's worth to note that because the server can be connected to several LSP clients at once, the Write() and Close() methods should take a client's unique connection ID as an argument, indicating the connection that should be written to or that should be closed, which is like the following:

```
func (s *server) Write(connID int, payload []byte) error {
        s.requestWrite <- idPayloadBundle{connID, payload}
        if !<-s.responseWrite {
                return errors.New("Client " + strconv.Itoa(connID) + " has lost")
        }
        return nil
}

func (s *server) CloseConn(connID int) error {
        s.requestCloseConn <- connID
        if !<-s.responseCloseConn {
                return errors.New("Client" + strconv.Itoa(connID) + " does not exist")
        }
        return nil
}
```

To see the further working performance of our LSP in the bitcoin mining system, we also design different tests to see the details in the communication process.

- **Tests implementation**

We design our tests from five aspects to see the performance of our LSP implementation.

- The first test is a basic test, in which a server and possibly multiple clients and runs a simple echo. In this test, we test that all messages sent by one side must be received by the other. Messages read by the other are then immediately written back. We also test the LSP robustness by inserting some random delays between client/server reads or writes. The following methods are declared to achieve our goal:

```
func newTestSystem(t *testing.T, numClients int, params *Params) *testSystem {

// runServer sets up the server and reads/echos messages back to clients.
func (ts *testSystem) runServer() {

func (ts *testSystem) randSleep() {

func (ts *testSystem) runClient(clientID int, doneChan chan<- bool) {

func (ts *testSystem) runTest(timeout int) {
```

- The second test is designed for the sliding window. The sliding window represents an upper bound on the range of messages that can be sent without acknowledgment. TestWindow1-3 test the case that the sliding window has reached its maximum capacity. TestWindow4-6 test the case that messages are returned by Read() in the order they were sent. The important methods are declared as below:

```
func newWindowTestSystem(t *testing.T, mode windowTestMode, numClients, numMsgs int, params *Params) *windowTestSystem {

// Client sends a stream of messages to the server.
func (ts *windowTestSystem) streamToServer(connID int, cli Client, sendMsgs []string) {

// Server sends a stream of messages to a client.
func (ts *windowTestSystem) streamToClient(connID int, sendMsgs []string) {

// Client reads a stream of messages from the server.
func (ts *windowTestSystem) readFromServer(connID int, cli Client, totalMsgs int, checkpoints ...int) {

// Server reads a stream of messages from all currently connected clients.
func (ts *windowTestSystem) readFromAllClients(totalMsgs int, checkpoints ...int) {

func (ts *windowTestSystem) runMaxCapacityTest() {

func (ts *windowTestSystem) runScatteredMsgsTest() {
```

- The next step is to test the LSP server/client close. In other words, the methods in this test are designed to check that the client/server Close methods work correctly. For example, pending messages should be returned by Read() and be written and acknowledged by the other side before Close() returns. At the same time, these methods also test that a client is able to connect to a slow-starting server. The important methods are declared as following:

```
func newCloseTestSystem(t *testing.T, mode closeTestMode) *closeTestSystem {

// Create client. Have it generate messages and check that they are echoed.
func (ts *closeTestSystem) buildClient(clientID int) {

func (ts *closeTestSystem) buildServer() {

func (ts *closeTestSystem) runTest() {
```

- The fourth test is LSP advanced buffering/synchronization test. In this test, messages are "streamed" in large batches and the network is toggled on/off (i.e. drop percent is set to either 0% or 100%) throughout. The server/client Close methods are also tested in the presence of an unreliable network. We design the following important methods:

```
// Alternates between turning network writes on and off in a loop.
// Runs in a background goroutine and is started once at the very beginning of a test.
func (ts *syncTestSystem) runNetwork() {

// Create and runs the server. Runs in a background goroutine and is started
// once at the very beginning of a test.
func (ts *syncTestSystem) runServer() {

// Create and runs a client. Runs in a background goroutine. Called once per client at
// the very beginning of a test.
func (ts *syncTestSystem) runClient(clienti int) {

// Toggle the network and wait until sends back a notification that it is done.
// Called from the master's goroutine.
func (ts *syncTestSystem) toggleNetwork() {
```

- The last test focuses on the discussion in “4. Data size change” of Section 4.2.1. The important implementation methods are listed as follows:

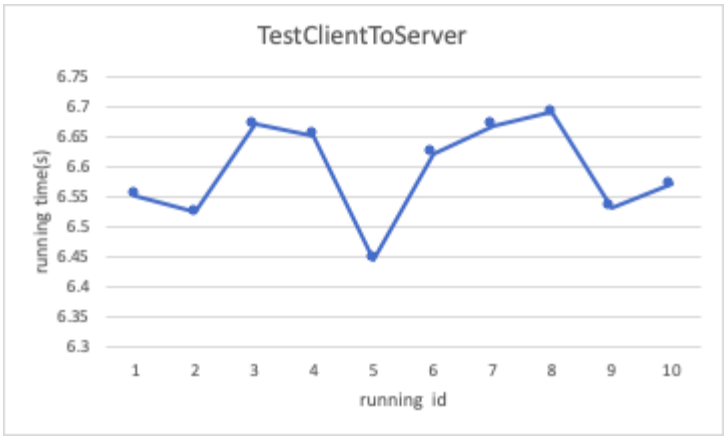
```
func (ts *testSystem) testServerWithVariableLengthMsg(timeout int) {

func (ts *testSystem) testClientWithVariableLengthMsg(timeout int) {

func TestVariableLengthMsgServer(t *testing.T) {
```

```
func TestVariableLengthMsgClient(t *testing.T) {
```

In the next section, we will display the experiment results based on the above discussed tests.

4.2.3. LSP Evaluation

The test results of LSP (running 10 times):



4.3. Implementation of bitcoin mining system

In this project, instead of realizing a real Bitcoin protocol, we decided to implement a mining infrastructure based on simplified variant. For example, if given a message M and an unsigned integer N, we need to find the unsigned integer n ($0 \leq n \leq N$) when concatenated with M can generate the smallest hash value. We refer each of these unsigned integers as a nonce in our paper.

In this Chapter, we will describe the coding and design logic of the bitcoin mining system with details. The contents can be divided into five parts: what a client should do, how a server works, how a miner works, what are the messages transmitted in the system, and how the hash value computed. Overall, the implementation logic is shown as follows.

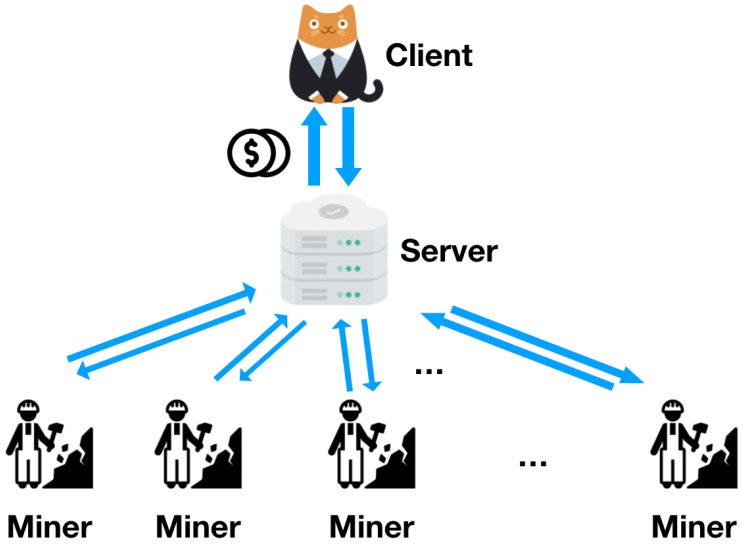


Figure 4.1 Bitcoin Mining System

4.3.1. Client

The first thing to do is to implement a new client using the function lsp.NewClient(hostport, params).

The logic of a client is simple. A client should send a mining request to a server. The request is initialized by:
bitcoin.NewRequest(message, 0, maxNonce)
where 0 is the lower bound of the testing value while maxNonce is the upper bound. Note that, the message and maxNonce are got from os.Args.
Before sending the request, we choose to use JSON to encode the message.
As we have implemented the LSP, we can directly use Write( ) function to send the request.

The client should always be here waiting for a response from the server with the Read( ) function to get the mining result.

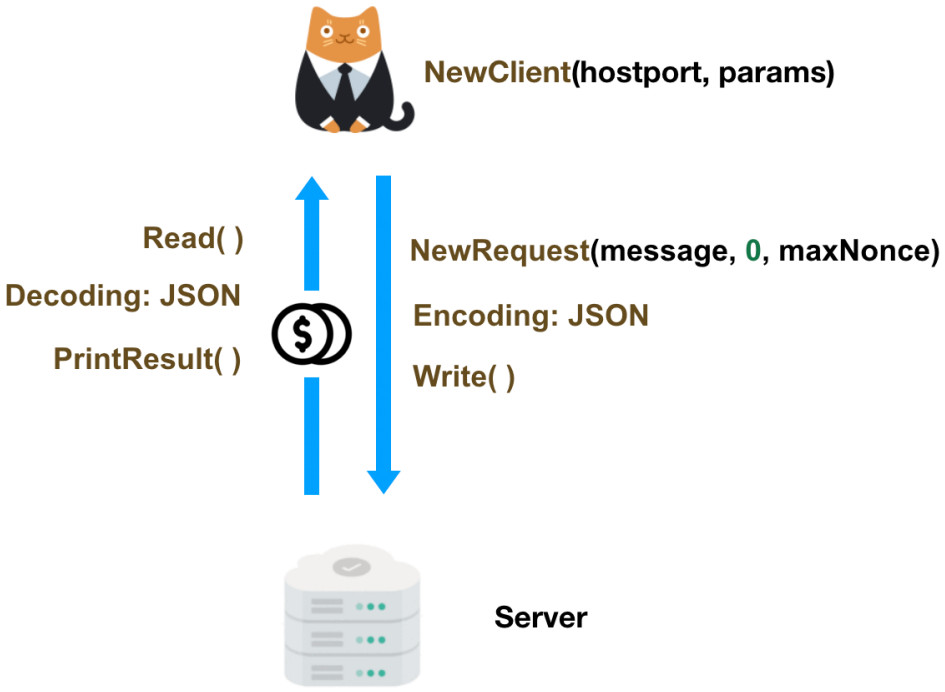


Figure 4.2 Implementation of Clients

4.3.2. Server

Before talking about the logic flow of the server, we first define some structs.

```
Job{
        portClient
        jobmessage
}
```

The Job struct is defined for jobs assigned to miners. portClient is the port of the client who sends the request. The data, upper and lower bounds are included in the jobmessage.

```
Miner{
        port
        currentJob
}
```

The currentJob in the Miner struct is the current job assigned to this miner.

```
Client{
        numMiner
        finishMiner
        minHash
        minNonce
}
```

In the client struct, we denote the number of miners the server assigns to the client as numMiner. To test the finish of the mining jobs, we define the finishMiner as the

number of miners who finish its job. minHash is the minimum hash value found while minNonce is the corresponding nonce of the minHash.

The main work of a server is to listen to clients, assign jobs to miners, handle results and return results from miners to clients.

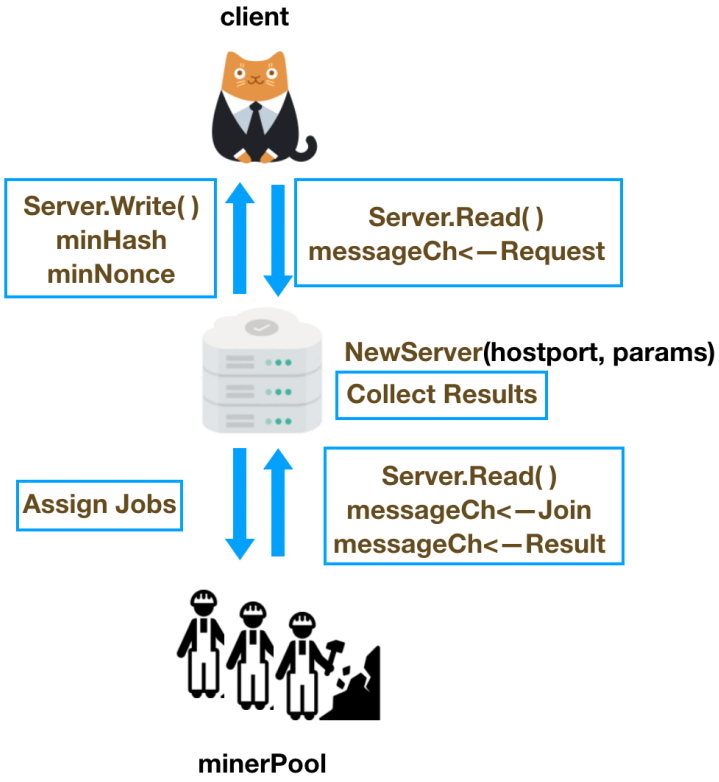


Figure 4.3 Implementation of Servers

We define the messageCh as a channel to receive messages and the minerCh to denote available miners. joblists is a list to record jobs. minerpool is a map of miners. clientpool is a map of clients.
First, a new server is initialized with lsp.NewServer(port, params). After that, the server is always listening to messages with server.Read( ), where we need a go function. The received message is put in the messageCh.
When there is a message in the messageCh, we should handle it.
When there is an available miner, we should assign jobs to it (if there is indeed some jobs.)
We use a *select-case* statement to handle these two cases.

```
for{
        select{
                case message <- messageCh :
                case avaMiner := <- minerCh :

        }
```

```
}
```

When there is a message in the messageCh, we should handle it based on different types.

a. Join message

If the server receives a Join message from a miner, it means that a miner is ready to work for the server. Thus, we should put the miner in the minerCh:

```
var newminer Miner
newminer.port= message.port
newminer.currentJob=nil
minerpool[newminer.port]=*newminer//add the new miner m to the minerpool list
minerCh := <- newminer
```

b. Request message

If the server receives a request message from a client, it should split the task from the client to several jobs. The number of jobs depends on how many miners the server want to assign to the client, i.e., num_miner.

To split the task to several jobs, we reset the upper and lower nonce of each job and add the job to job lists as follows:

```
for i:=0; i< num_miner; i++{
   newjob.portClient=message.port
   lowernum:=messagehandle.Lower+avg*i
   uppernum:=messagehandle.Lower+avg*i+avg-1
   if i==num_miner-1{
       uppernum=messagehandle.Upper
   }
   newjob.jobmessage=bitcoin.NewRequest(messagehandle.data, lowernum,
uppernum)
   //a new job of find hash from lower to upper
   jobList.PushBack(&newjob)
}
```

The client should also be added to the clientpool.

c. Result message

When receiving a result message, the server should put the miner back to the minerCh because it's available again.

The server should collect all the results from all the assigned miners. Each time the miner receives a result, it should update the minHash and minNonce.

```
if messagehandle.Hash < getclient.minHash{
    client.minHash = messagehandle.Hash
```

```
    client.minNonce = messagehandle.Nonce
}
```

When the server collects all the results, it's clear that the number of receiving results equals to the number of assigned miners.

getclient.numMiner==getclient.finishMiner

Then, the server can return the result to the client.

Note that the result:

bitcoin.message.NewResult(getclient.minHash, getclient.minNonce)

should be encoded with JSON.

The reply is via server.Write(job.portClient, result_message)

When there is an available miner, we first check if there is at least one job in the joblists. If the list is empty, right now there is no job can be assigned to the miner so that the miner is still available.

minerCh <- avaMiner

If there are jobs. We assign the job to the miner with

avaMiner.currentJob=job

and send the job message to the miner

server.Write(avaMiner.port, sendMessage)

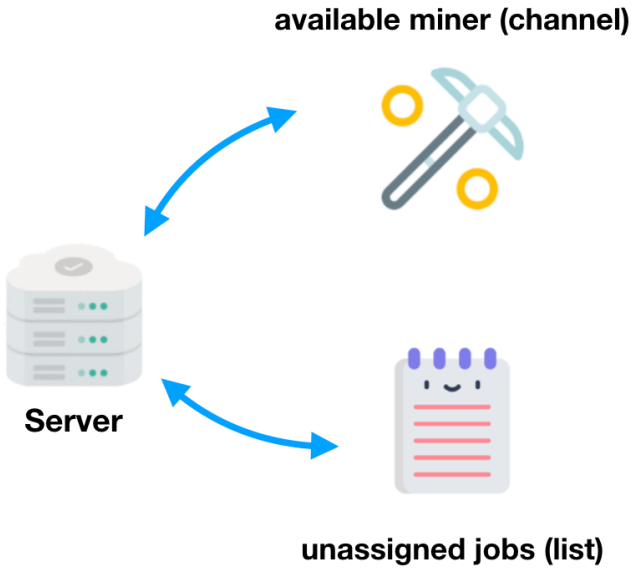


Figure 4.4 Logic of the Server, Jobs and Miners

4.3.3. Miner

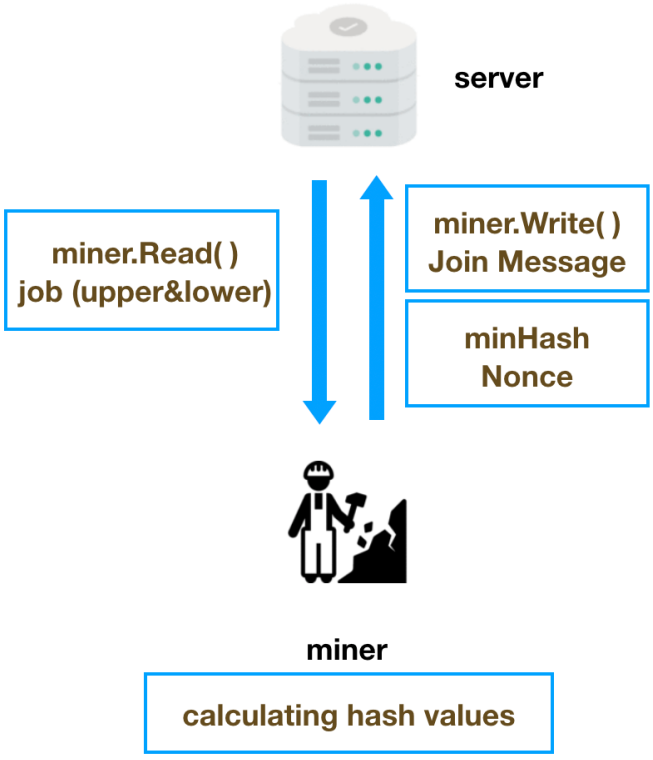


Figure 4.5 Implementation of Miners

The main work of a miner is to calculate the hash value of a range of given nonces. After establishing a new miner with lsp.NewClient(hostport, params), we should send a join message, bitcoin.NewJoin( ), to the server to let it know that there is a miner ready to work. The sending is via LSP miner.Write( ).

Then, the miner should always be listening to the server with miner.Read( ).
When there is a request message from the server, the miner get a range of nonces with the lower and upper. For each nonce, the miner calculates the corresponding hash value using bitcoin.Hash(message.Data, i).
Before returning the result to the server, the miner should figure out which hash is the smallest one it gets.

```
if hash<result.Hash{
    result.Hash=hash
    result.Nonce=i //only choose the smallest one
}
```

In the end, the smallest hash and its corresponding nonce are encoded with JSON and returned to the server via miner.Write()

4.3.4. Message

There are three kinds of messages in the mining system, Join, Request and Result. The struct of these three types of messages are defined in *message.go*.

```
type Message struct {
    Type        MsgType
    Data        string
    Lower, Upper uint64
    Hash, Nonce  uint64
}
```

We use the functions NewRequest, NewResult and NewJoin to initialize the messages. A Join message is sent from a miner to a server. A Request message is sent from a client to a server and from a server to a miner. A Result message is from a miner to a server and from a server to a client.

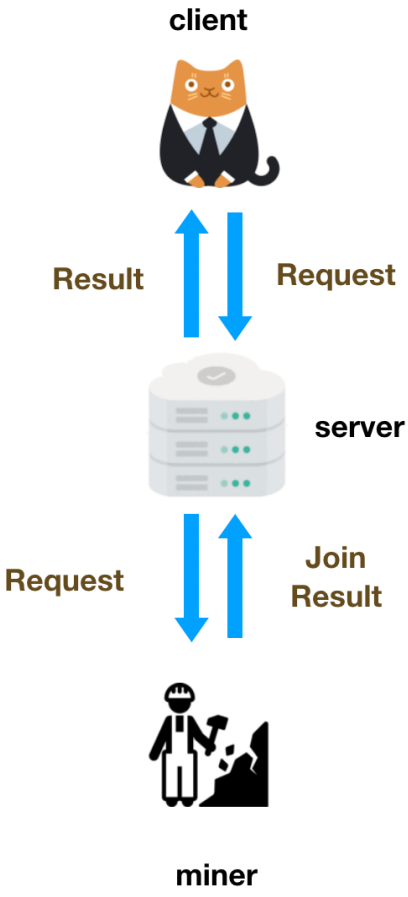


Figure 4.6 Messages in the Bitcoin Mining System

4.3.5. Hash

The hash of a given data and nonce is calculated with the package “crypto/sha256" as follows.

```
hasher := sha256.New()
hasher.Write([]byte(fmt.Sprintf("%s %d", msg, nonce)))
return binary.BigEndian.Uint64(hasher.Sum(nil))
```

5. Preliminary Evaluation

For single node system, we only need to evaluate the mining process of one miner. We have a target hash value with several leading 0s: 0x0000000000xxxxxxx. The number of prefix 0s is also called the mining difficulty.

And we need to combine prevHash and a nonce to compute the new hash. Only when the new hash is less than target, it stops and succeed. So we need to keep trying different nonce and compare the result with the target hash value.

To measure the complexity, we set a range for the nonce, like from 0 to 100,000. And see how long it takes to find a valid hash(less than target) or just reach the upper limit and stop. Also we can measure the relationship between the difficulty and the running time.

This is a high-level overview of our algorithm in single node implementation. Our aim is to analyze the run-time of this algorithm, given the desired number of leading 0’s from the hash. We confirm the hash by **Proof of Work**, which is the consensus algorithm in a Block-chain network.

Algorithm SINGLENODEMINER(N)
Input: A variable N; where N is the number of desired leading 0’s from the hash, known as “difficulty”
Output: Boolean value. **True** if hash could be verified and appended to BlockChain array. **False** else.

1. $BlockChain \leftarrow []Block$
2. $block \leftarrow createEmptyBlock()$
3. $block.hash \leftarrow computeHash(block)$
4. **if** $isBlockValid(block)$
   do $BlockChain.append(block)$
   return true
5. **return** false

Notice that each nonce value generates a *random* hash value. We cannot efficiently predict which nonce value will give how many leading zeros, but the occasional nonce will generate a hash that meets our condition. Nevertheless, we can verify that

more leading zeros results in exponentially larger number of nonces. To see this, let l_i be a hash with *i* leading zeros. Then, $l_i \subseteq l_j$ where $i < j$. It follows that higher number of "difficulty" leads to exponentially larger number of nonces and thus run-time.

For the distributed miner, we need to answer the following questions:
"Can we gather a group of workers as a single miner to get more computational resources?"
"How can we achieve the remote procedure call among workers?"
"What happens if a worker crashed or failed?"
"How can we assign PoW to workers and get the reply?"

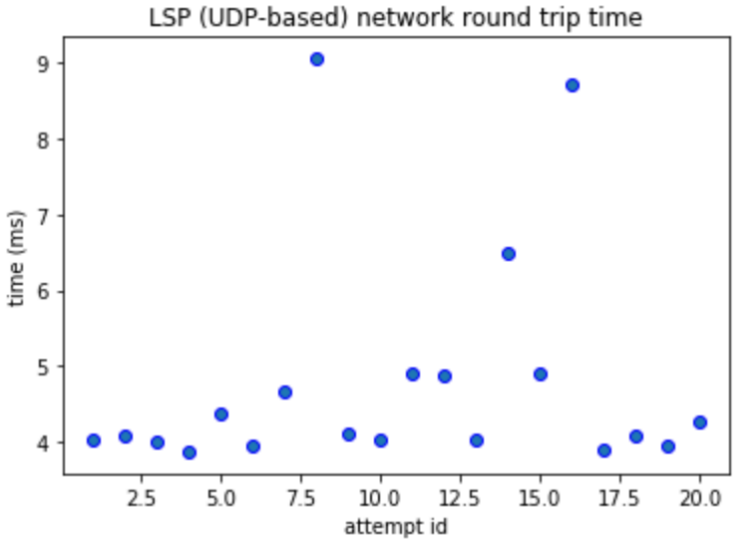
6. Evaluation of Tradeoffs

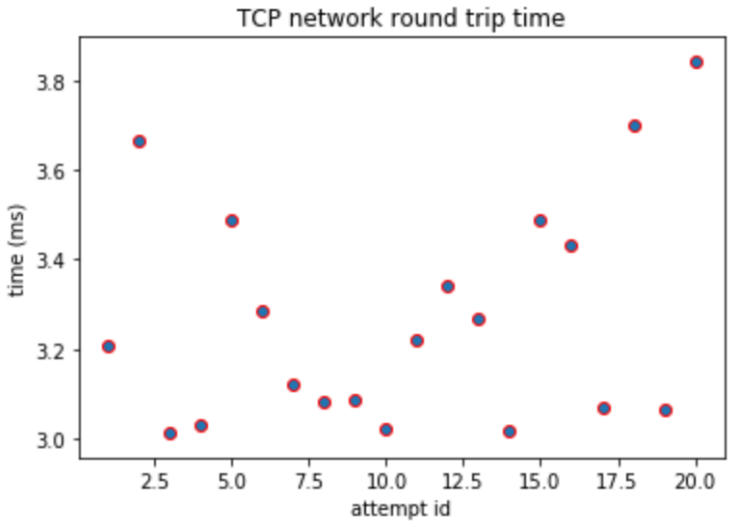
6.1. LSP vs. TCP network comparison

In our implementation, we implemented LSP ("live-sequence protocol"- a homegrown protocol on top of UDP to emulate some of the features of TCP). A natural question arose: what is the trade-off of implementing LSP as opposed to TCP? We know that TCP has overhead of setting up a connection, negotiating windows, and acknowledging data. LSP (much like TCP) sets up a client-server connection, negotiates window size, and acks data, but it's built on top of UDP, not TCP.

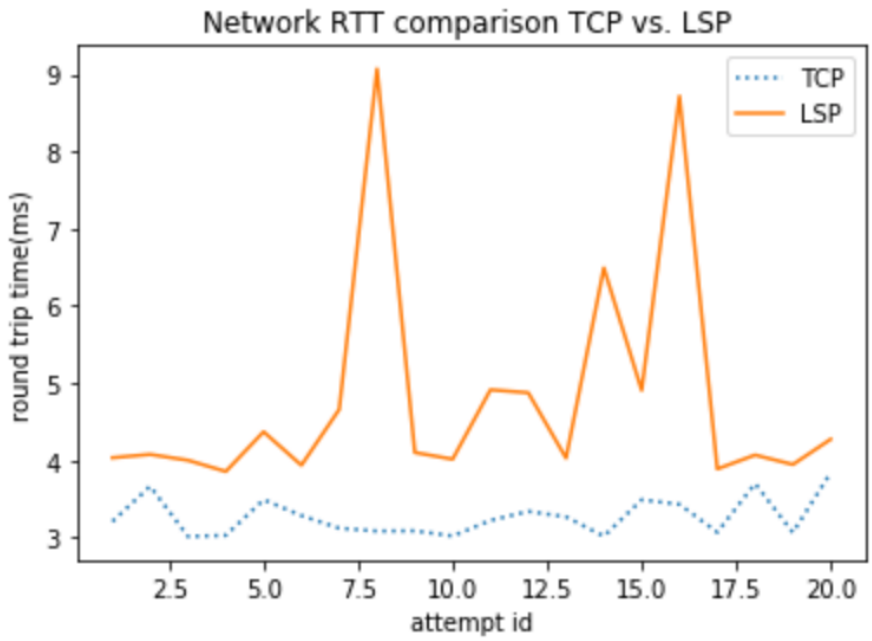
To calculate the round-trip time of both LSP and TCP, we also implemented a simple, concurrent TCP server and client. We wanted to gauge the RTT of each protocol and compare them side by side. By round-trip time, we imply: client → server → client. We obtained and recorded 20 round-trip-time attempts.

Using the data points we obtained, our plots are:





Note that LSP round-trip-times are clustered around 4 to 5 ms, with some heavy outliers (7ms+). On the other hand, TCP round-trip-times are relatively stable with most of the data points in the range of 3.4 ± 0.4 ms, and fewer outliers. TCP averaged at 3.27 ms, and LSP averaged at 4.81 ms. The plot of both protocols side-by-side is shown below:



From this plot, we learned that TCP on average has lower round-trip-time than LSP. Adding the TCP functionality to UDP slows performance of UDP (because we have to set up a connection and negotiate windows), and it shows in this evaluation.

For source code, ipynb, and csv files used for this plot and evaluation, refer to the second git repo link on ***8. Git repository***.

6.2. Evaluation of Distributed system

Refer to attached file: 462_562 Project Eval.pdf

6.3. Failure Handling

6.3.1. Logic of failure handling

There are three entities in the system, client, server and miner. In this section, we will describe how to handle failures in different entities. Some scenarios are given and analyzed. The basic logic and protocol are shown as follows.

A. Client

- If a server cannot send result messages to the client after three times tries, there is an error returned from server.Write() and a failure log is printed. Then the server can know that the client is crashed. This is based on the LSP we implemented.
  To serve the client when it re-joins with the same request, we can allow the server to maintain the result for less than 10 minutes as the time for

generating a new block in the blockchain is around 10 minutes. After the time section the server can remove the result permanently.

B. Server

- If a client cannot send messages to the server after three times tries, there is an error returned from client.Write() and a failure log is printed. Then the client can know that the server is crashed and the mining work can not start. This is based on the LSP we implemented.
- If a miner cannot send join messages or result messages to the server after three times tries, there is an error returned from miner.Write() and a failure log is printed. If we held a whole view of the system we can know that the server is crashed. This is based on the LSP we implemented.
- After a given time (threshold), if the client cannot receive the result from the server, it re-starts the work and re-sends the request message to the server. If the server is crashed, there will be no reply. The situation where some miners crashed rather than the server, is analyzed in the next subsection.

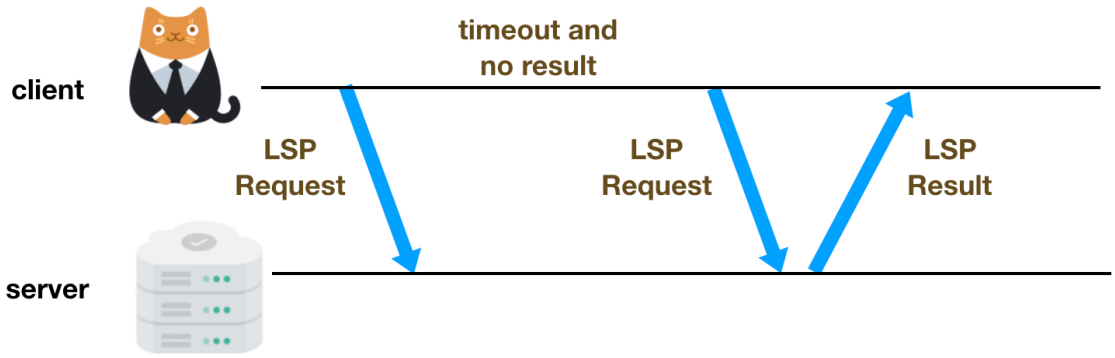


Figure 6.1 Scenario of Timeout Without Results

C. Miner

- If a server cannot send request messages to a miner, there is an error returned from server.Write() and a failure log is printed. Then the server knows that the miner is crashed and thus removes it from the miner list. The job is assigned to another available miner.
- After a given time (threshold), if the server cannot receive the result from some miner, it re-assign the job to the miner and wait for a threshold time again.
  If it fails again, the server knows that the miner is crashed. Now the server removes the miner from the miner list and re-assign the job to another available miner.
  In the meanwhile, to let the client know that the server is still working and waiting for the results from miners, the server should send some additional messages (a special kind of messages or heartbeats) to the client.

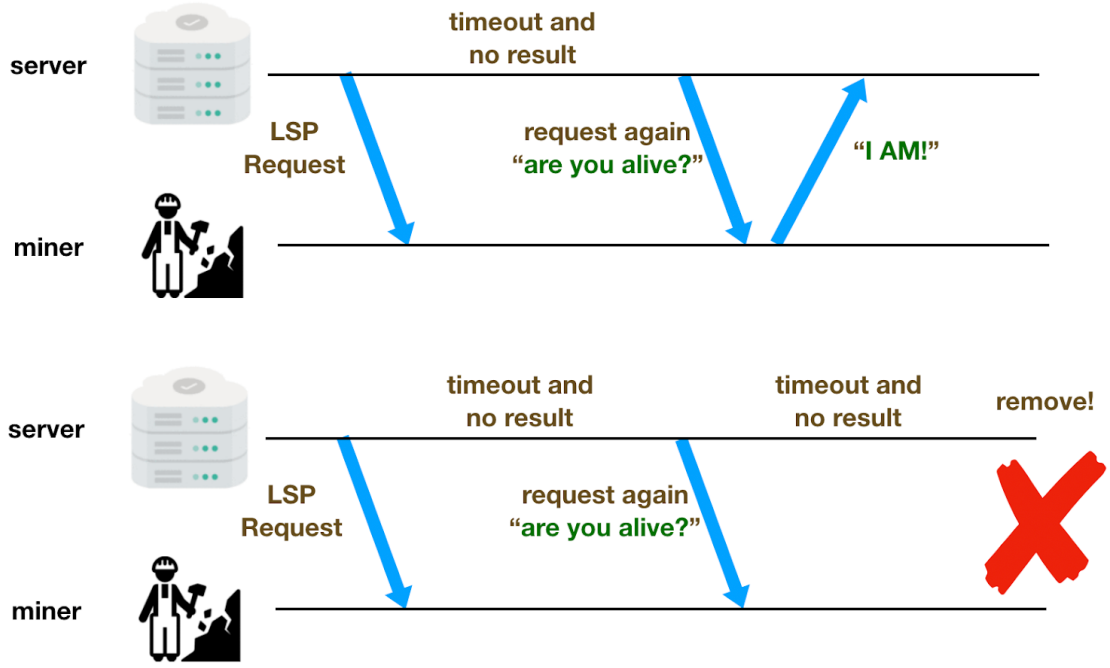


Figure 6.2 Scenario of Timeout Without Results

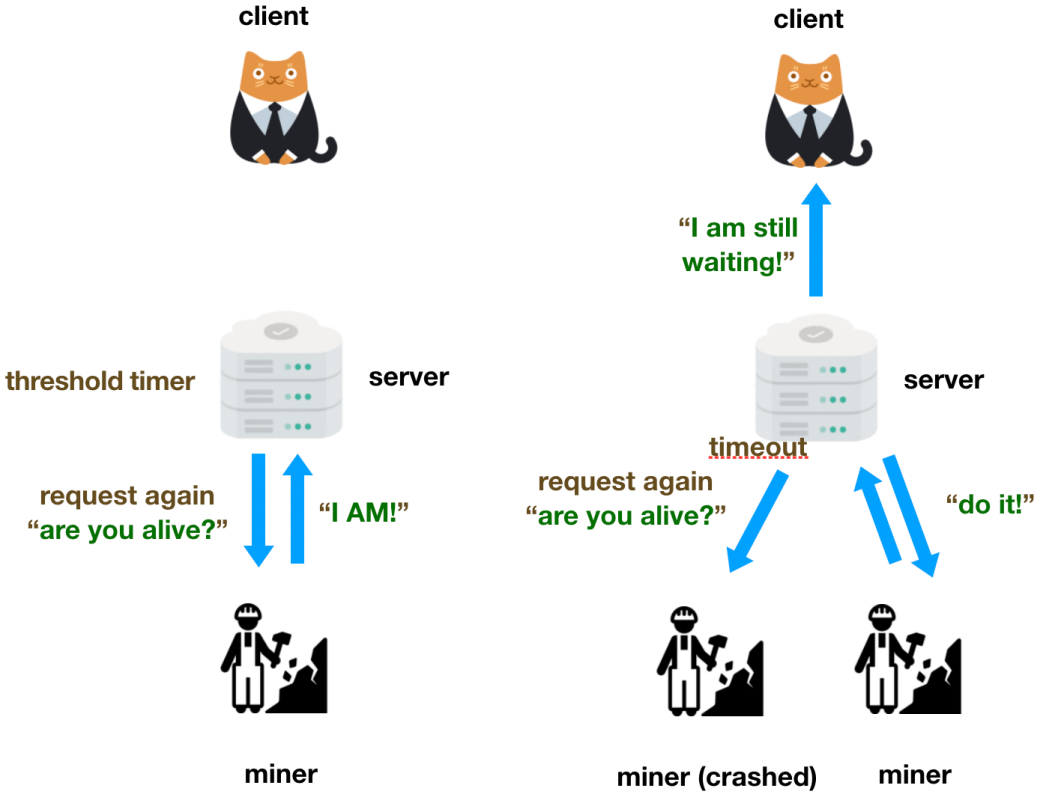


Figure 6.3 Scenario of Timeout Without Results

6.3.2. Tests of failure handling

We also test the failure handling mechanism in the system.

```
# Miner disconnected test
# Client Sends request to server: [Request 6073511289018497495 0 999999].
# Expecting result: [Result 51022575622139 3899]
# start server and 2 miners
$GOPATH/bin/server 6060 &
$GOPATH/bin/miner localhost:6060 &
$GOPATH/bin/miner localhost:6060 &
#client send job to server
$GOPATH/bin/client localhost:6060 6073511289018497495 999999 sleep 0.1
#kill all miners
ps -ef | grep $GOPATH/bin/miner | grep -v grep | awk '{print $2}' | xargs kill
#connect a new miner
sleep 0.2
$GOPATH/bin/miner localhost:6060
```

And the test run as expected, the server still print out the correct result, which is [Result 51022575622139 3899].

7. Conclusion

What we learned:

- How to manage packages and set up work-station for a large project in Golang
- Familiarity w/ documentation of Golang and their APIs
- Trade off between TCP and UDP. In this project, we implemented both TCP and LSP (which is protocol on top of UDP to emulate what is built into TCP - sliding window protocol, reliable bidirectional data transfer, client-server model, etc). Adding extra functionalities to LSP to emulate TCP slowed performance of data transfer.
- Build a concurrent server that handles multiple clients @ the same time
- Build a concurrent server that also keeps track of its miners, and farm out tasks to available miners and retrieve results
- Use networking protocols (TCP, UDP, package net) in Golang for server-client operations
- Learned to use Go as a powerful server side language that is easily scalable to larger systems with great tools to handle concurrency (thanks to goroutines and channels), and support object orientation (via. interfaces)

- Manage data flow in a complex project involving multiple miner, server, client components (i.e. *Data type Msg* was marshalled into bytes from client and sent over network to server, which server unmarshalled back into *Data type Msg* and took action according to the *MsgType*). Designing and building multiple functional components of a larger project
- Dealing with scalability. There are two scalability issues that our server had to deal with:
    - (case 1) more clients join server
    - (case 2) more miners join server

In (case 1), we agreed that if more clients join server, our (UDP) server distributes a jobQueue used to handle all requests from different clients.

In (case 2), we agreed that if more miners join server, our server keeps a list to track all available miners, once a miner is idle, the (UDP) server will distribute jobs from jobQueue to it.

Future steps & Afterword:

This was an interesting project. Going forward, our ideas for next steps would be:

- Developing a web server for the client
- Connecting client to the Bitcoin network to retrieve real data to hash
- Encoding and decoding for security. We would like to include encryption and decryption mechanisms to support strong authentication of packets.

8. Project Repository

(1) https://github.com/binggao92/Distributed-Bitcoin-Miner
(2) https://github.com/josh-j-park/462-562-CSC-Project-Eval (supplementary for 6.1 & 6.2)

9. Demo video

https://youtu.be/Q42ezDlB2fI

10. References

[1]. A. Rosic, "What is Blockchain Technology? A Step-by-Step Guide For Beginners", *Blockgeeks*, 2019. [Online]. Available: https://blockgeeks.com/guides/what-is-blockchain-technology/. [Accessed: 27- Jun- 2019].

[2]. M. Dalton, "Feeless Cryptocurrency: How Do Blockchains Achieve Free Transactions?", *Bitrates.com*, 2019. [Online]. Available: https://www.bitrates.com/news/p/feeless-cryptocurrency-

how-do-blockchains-achieve-free-transactions. [Accessed: 27- Jun- 2019].
[3]. "What is bitcoin? - CoinDesk", *CoinDesk*, 2019. [Online]. Available: https://www.coindesk.com/information/what-is-bitcoin. [Accessed: 29- Jun- 2019].
[4]. "SHA-256 hash calculator «", *Xorbin.com*, 2019. [Online]. Available: https://www.xorbin.com/tools/sha256-hash-calculator. [Accessed: 28- Jun- 2019].
[5]. E. Kotow, M. Thibodeau, S. Moser and S. Joudrey, "What is Blockchain Hashing? - HedgeTrade Blog", *HedgeTrade Blog*, 2019. [Online]. Available: https://hedgetrade.com/what-is-blockchain-hashing/. [Accessed: 28- Jun- 2019].
[6]. Eyal, Ittay, and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable." Communications of the ACM61.7 (2018): 95-102.
[7]. Antonopoulos, Andreas M. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.
[8]. C. Decker, J. Seidel and R. Wattenhofer, "Bitcoin meets strong consistency," in 2016, . DOI: 10.1145/2833312.2833321.
[9]. A. Gervais et al, "On the security and performance of proof of work blockchains," in 2016, . DOI: 10.1145/2976749.2978341.
[10]. H. Watanabe et al, "Blockchain contract: A complete consensus using blockchain," in 2015, . DOI: 10.1109/GCCE.2015.7398721.
[11]. R. S. A. Daulay, S. M. Nasution and M. W. Paryasto, "Realization and Addressing Analysis In Blockchain Bitcoin," IOP Conference Series: Materials Science and Engineering, vol. 260, pp. 12002, 2017.
[12]. C. Cachin and M. Vukolić, "Blockchain Consensus Protocols in the Wild," 2017.
[13]. Eyal, Ittay, and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable." Communications of the ACM61.7 (2018): 95-102.
[14]. Lee, T.B. Four reasons Bitcoin is worth studying. forbes.com/sites/timothylee/2013/04/07/four-reasons- bitcoin-is-worth-studying/2/ (2013).
[15]. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system (2008).
[16]. Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).
[17]. Boly, Jean-Paul, et al. "The ESPRIT project CAFE—High security digital payment systems." *European Symposium on Research in Computer Security*. Springer, Berlin, Heidelberg, 1994.
[18]. Back, Adam. "Hashcash-a denial of service counter-measure." (2002).
[19]. Kraft, Daniel. "Difficulty control for blockchain-based consensus systems." *Peer-to-Peer Networking and Applications* 9.2 (2016): 397-413.
[20]. Bahack, Lear. "Theoretical Bitcoin attacks with less than half of the computational power (draft)." *arXiv preprint arXiv:1312.7013* (2013).
[21]. Shomer, Assaf. "On the Phase Space of Block-Hiding Strategies." *IACR Cryptology ePrint Archive* 2014 (2014): 139.
[22]. Li, Jiao. "Data Transmission Scheme Considering Node Failure for Blockchain." *Wireless Personal Communications*103.1 (2018): 179-194.
[23]. Pai, V., Kumar, K., Tamilmani, K., Sambamurthy, V., & Mohr, A. E. (2005). Chainsaw: Eliminating trees from overlay multicast. In *International conference on peer-to-peer systems* (pp. 127–140).
[24]. Biskupski, B., Schiely, M., Felber, P., & Meier, R. (2008). Tree-based analysis of mesh overlays for peer-to-peer streaming. In *Ifip Wg 6.1 international conference on distributed applications and interoperable systems* (pp. 126–139).
[25]. Ranga, V., Dave, M., & Verma, A. K. (2014). A hybrid timer based single node failure recovery approach for WSANs. *Wireless Personal Communications, 77*(3), 2155–2182.
[26]. Kishigami, Junichi, et al. "The blockchain-based digital content distribution system." *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. IEEE, 2015.
[27]. Ryoichi Mori, about Software Service JECC Journal, no. 3, pp. 16-26, 1983.

[28]. Koichi Sakanoue, Junichi Kishigami et al., "New Sevices and Technologies Associated with Metadata", *NTT Technical Review*, vol. 1, no. 3, pp. 46-50, 2003.
[29]. Zibin Zheng, Shaoan Xie, Hong-Ning Dai, and Huaimin Wang. Blockchain challenges and opportunities: A survey. Work Pap.-2016, 2016.
[30]. Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. Applied Innovation, 2(6-10):71, 2016.
[31]. Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin G un Sirer. Decentralization in bitcoin and ethereum networks. arXiv preprint arXiv:1801.03998, 2018.
[32]. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Kho, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In 25th fUSENIXg Security Symposium (fUSENIXg Security 16), pages 279-296, 2016.
[33]. Arati Baliga. Understanding blockchain consensus models. In Persistent. 2017.

Appendix

(1)

	Project proposal	Background	High-level of the network	Application scenarios	Code implementation	System architecture diagram	Single node evaluation	Distributed evaluation	Formatting	Literature review	Video Demo	Conclusion	Presentation/ slides
Bing Gao													
Jinghan Jiang													
Rui Liu													
Gloria Wang													
Joshua Park													

(2)

The three literature reviews put above is based on the five questions required. Here is also another paper-style literature review, which Yvonne asked me to include it in the project too. Thanks. -- Rui Liu

Proof-of-work Mechanism in The Bitcoin Network

— An Innovative Solution
for Consensus in Distributed System

I. INTRODUCTION

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another [1]. Nodes in a distributed system work with each other in a coordinated fashion while achieving a common goal.
The bitcoin network [2] is a decentralized distributed system. It allows truly global transactions, i.e., there is no single authority in the bitcoin network, which is different from some traditional currencies. Because of the distributed feature of transaction tracking and validation, the bitcoin network needs to achieve a consensus of the transactions and account balances.
In this paper, we first introduce basic algorithms for achieving distributed consensus and analysis their limitations in section II. In section III, a new solution in bitcoin system, the Proof-of-work

Mechanism, is described. We give a brief analysis of the future work in section IV and describe our project challenges in section V.

II. DISTRIBUTED CONSENSUS

The consensus is a process of reaching an agreement among nodes on some output data. Plenty of consensus algorithm has been proposed, proven correct, and widely accepted in past years. In 1998, Lamport [3] published Paxos, which is the first practical and fault-tolerant consensus algorithm for the real world. In 2014, Ongaro and Ousterhout [4] published the Raft algorithm which uses a shared timeout for termination.

However, both Paxos and Raft are designed as crash fault-tolerant but not Byzantine fault-tolerant. In a Byzantine system, nodes may act arbitrarily and even coordinate. To solve such a problem, Castro and Barbara [5] proposed the Practical Byzantine Fault Tolerance (PBFT) algorithm working in asynchronous environments. PBFT has a much better performance comparing with previous Byzantine fault-tolerant algorithms but can only work with a limited number of participants. Thus, it's not practical for real-world use.

III. PROOF-OF-WORK

In the concept of consensus, bitcoin mining is a process to facilitate the achievement of consensus with a consensus mechanism named Proof-of-work (PoW). PoW relies on that only when enough computational resources have been used, a value can be accepted by the whole network. With the proof-of-work mechanism, the bitcoin network can satisfy Partition tolerance (P) and Availability (A) of the CAP theorem while achieving eventual consistency.

The PoW requires miners to find a number called a nonce, such that when the block content is hashed along with the nonce, the result is numerically smaller than the network's difficulty target [6]. The nonce is easy to verify but extremely hard to choose. A miner may try a mass of different values.

The block of the first miner who meeting the difficulty target, i.e., solve the proof-of-work problem, is considered valid. The miner publishes the block to the whole network, then all nodes reach to an agreement. The value of the difficulty target is adjusting so that every valid block is found in around 10 minutes. Figure 1 shows the average time to mine a block with PoW. Data are collected online [7].

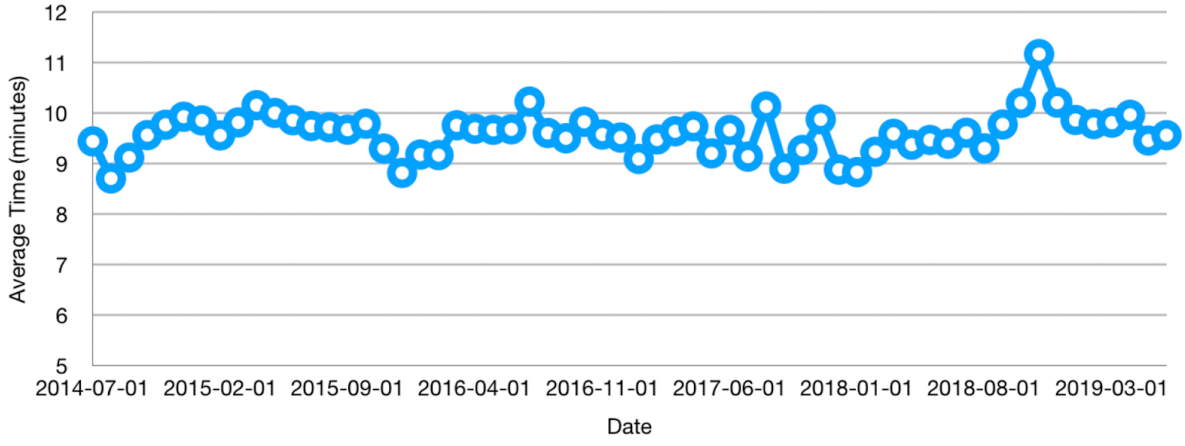


Figure 1. Average Time to Mine A Block

In the meanwhile, PoW solves the Majority decision problem, as only the longest blockchain is accepted and admitted by the whole bitcoin network. The security relies on the principle that no entity

should gather more than 50% of the processing power because such an entity can effectively control the system by sustaining the longest chain [8].

IV. FUTURE CHALLENGES

Though PoW can achieve the eventual consensus in bitcoin networks, it also implies some challenges. First of all, because of the high requirement of computational resources, energy consumption becomes a problem. The Bitcoin network uses a lot of power, which, in 2014, was roughly estimated to be in the ballpark of 0.1–10 GW [9]. Besides, blockchain requirements change rapidly, with high latency and low throughput of Bitcoin-like blockchain becoming a major challenge [10].

V. OUR PROJECT

In our project, our main goal is to achieve the distributed consensus in a simulative bitcoin network. Here are some questions we wanna think and try to solve:

- how to realize miner to solve the PoW problem?
- can we gather a group of workers as a single miner to get more computational resources? how to achieve the remote procedure call among workers?
- what if a worker crashed or failed?
- how to assign the PoW work to workers and get the reply?

The evaluation is based on the following questions:

- what is the average time to find a solution in PoW with the different number of workers for a miner?
- what is the average time to find a solution in PoW with the different number of worker failures?
- what is the average time to find a solution in PoW with different length of the blockchain?
- what is the average time to find a solution in PoW with different difficulty target?

REFERENCE

[1] Tanenbaum, Andrew S., and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[2] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).

[3] Lamport, Leslie. "The part-time parliament." *ACM Transactions on Computer Systems (TOCS)* 16.2 (1998): 133-169.

[4] Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 2014.

[5] Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." *OSDI*. Vol. 99. 1999.

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

[7] online data resources: data.bitcoinity.org

[8] Gervais, Arthur, et al. "On the security and performance of proof of work blockchains." *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016
[9] O'Dwyer, Karl J., and David Malone. "Bitcoin mining and its energy footprint." (2014): 280-285.
[10] Clark, J. B. A. M. J., A. N. J. A. K. Edward, and W. Felten. "Research perspectives and challenges for bitcoin and cryptocurrencies." *url: https://eprint. iacr. org/2015/261. pdf* (2015).