Blockchain Technology SoS

Pagoti Hemanth Naidu - 210050117
Mentor: Shikhar Agrawal

june 2023

Contents

1 Introduction

Bitcoin was introduced in 2009 by Satoshi Nakamoto in response to the 2008 global financial crisis and the need for a decentralized, peer-to-peer electronic cash system. Nakamoto's vision was to create a currency free from central control, censorship-resistant, and capable of secure, borderless transactions.

1.1 Demand & Supply of Bitcoin

Bitcoin's supply is regulated by Satoshi Nakamoto's algorithm, limiting the total number of bitcoins that can ever exist to 21 million. The demand for bitcoin is determined by market forces and influenced by factors such as investor sentiment and adoption. Balancing supply and demand is essential for maintaining stability in the Bitcoin market and mitigating the risk of inflation.

1.2 Double Spend Problem

Satoshi Nakamoto's design of Bitcoin ensures that money can only be spent once through the concept of ownership and peer-to-peer networks. With public keys, everyone can know how much each owner possesses while maintaining anonymity. Hash functions are used for encoding transactions and mining purposes, while proof of work enables Bitcoin to function as a cryptocurrency. The tamper-proof nature of blockchain ensures that once transactions are added, they cannot be modified, and each block in the blockchain contains the hash of the previous block. Smart contracts are self-executing contracts with the terms of the agreement directly written into code, automating and enforcing the agreed-upon conditions without the need for intermediaries. They enable secure and transparent transactions in various applications, from finance to supply chain management.

2 Motivation

2.1 Blockchains in High Level

- Blockchain utilizes a tamperproof data structure that ensures data integrity and security.
- Starting with a genesis block, each subsequent block in the chain holds information in the form of bitstrings or binary representations.
- Once data is added to a block and appended to the chain, it becomes immutable and cannot be removed or altered.
- This feature of blockchain, along with its decentralized and distributed nature enables various applications such as cryptocurrency, smart contracts and trust.

2.2 Applications

- **Governance**: Land records, health records, transportation data, virtual currency, electronic wills, passport, identification, etc.
- **Commercial**: supply chains, auctions, gaming, sale of music, financial services, smart grid etc.
- **Disruptive**: Cryptocurrencies, Initial Coin Offerings

3 cryptocurrency

Bitcoin has no central trudtrd authority, cryptography brings in trust and peer to peer networks provide decentralization. Bitcoin satisfies the characteristics of acceptability, portability, durability, divisibility, and fungibility.

- **Acceptability**: Bitcoin is widely accepted as a form of payment and store of value.
- **Portability**: Bitcoin can be easily transferred and accessed across geographical boundaries.
- **Durability**: Bitcoin's digital nature makes it resistant to physical damage or deterioration.
- **Divisibility**: Bitcoin is divisible into small units, allowing for precise transactions of any value.
- **Fungibility**: Bitcoin units are interchangeable, meaning that each unit holds the same value and can be exchanged without distinction.

Each transaction is brodcasted among all so that double spend is avoided.
Consensus protocols are mechanisms used in blockchain networks to achieve agreement among participants on the validity of transactions and the order in which they are added to the blockchain and this is also know as proof of work.

4 Peer to Peer

A permissionless, distributed system allows an arbitrary number of participants to join or leave at any time, enabling seamless peer-to-peer transactions where individuals have the freedom to pay anyone within the network, fostering inclusivity and decentralization.

4.1 Consensus Protocol

- To establish consensus in a protocol, cryptography is employed to secure transactions and validate the integrity of the data.

- While a peer-to-peer network enables direct communication and information sharing among participants.
- The proof-of-work mechanism adds a layer of computational effort to validate transactions and prevent malicious behavior, and Merkle trees provide an efficient and verifiable way to store and verify the integrity of large sets of data within the blockchain.
- Together, these components form the foundation for a robust and trustless consensus protocol.

4.2 Popular P2P versions

4.2.1 Napster

Napster was one of the earliest peer-to-peer networks that facilitated file sharing. It used a centralized server to maintain an index of available files, allowing users to search and download files directly from each other, enabling widespread file sharing but faced legal challenges due to copyright concerns.

4.2.2 Gnutella

Gnutella is a decentralized peer-to-peer network where participating nodes connect directly with each other. It operates on a query flooding mechanism, where search queries propagate across the network, and peers respond with matching files, ensuring a distributed and scalable file sharing system.

4.2.3 Distributed Hash Table

DHT is a decentralized peer-to-peer network architecture that provides efficient lookup and storage of key-value pairs. It operates by partitioning the hash space among participating nodes, enabling direct lookup and storage of data based on keys. DHTs provide fault tolerance, scalability, and efficient resource utilization in distributed systems and are commonly used in applications like BitTorrent and distributed storage systems.

5 Hash functions

Each block in the blockchain contains hash of the previous block, the hash is recognised as an unique ID.

5.1 Basic properties of Hash functions

- Input to the hash function can be of any length.
- Output of the hash function should be of fixed size.
- Hah should be efficiently computed.

5.2 Random Oracle

It is like a black box that takes input and generates a random output based on that input. It provides a consistent and unpredictable response to any query. Random oracles are often used to model idealized cryptographic functions and serve as a building block for designing secure protocols and systems.
However, in practice, real-world hash functions are used as a substitute for random oracles.

5.3 Properties of Cryptographic Hash functions

5.3.1 Collision Resistence

Collision resistance, in the context of cryptographic hash functions, means that it is computationally infeasible to find two different inputs that produce the same hash output. It ensures that a small change in the input will result in a significantly different hash value, making it difficult to forge or manipulate data without detection.
The birthday paradox states that in a group of relatively few people, the probability of two individuals sharing the same birthday is surprisingly high.

5.3.2 Hiding

suppose r takes values from r_1, r_2, r_3, ..., r_n with probability $p(r_1)$, $p(r_2)$, $p(r_3)$, ..., $p(r_n)$.
The minimum entropy is defined as follows

$$min - entropy = min_{i=1}^{n} - log(x_i)$$

where a secret value r is chosen from a probability distribution with high min-entropy and combined with another value x to compute the hash h(x||r), it is infeasible to find the original value of x given only the hash h(x||r).

5.3.3 Puzzle Friendliness

Hash function is puzzle friendly if for every N bit output Y if K is choosen from a probability distribution with high minimum entropy, then it is infeasible to find X such that

$$H(K||X) = Y$$

in time significantly less than O(2^N).

5.4 Proof of Work

- Miner selects pending transactions.
- Miner combines hash of previous block, nonce, and transactions.
- Nonce is of 16 bits, miner can include any number of transactions.

- Miner adjusts nonce until the resulting block hash meets the threshold for network consensus, creating the next block.
- Once a valid block is created it should be brodcasted to other miners for verification.

6 SHA256 and Merkle Trees

Here's a high-level overview of how SHA-256 works:

- **Padding**: The input message is padded with length of message to ensure it meets certain requirements for the SHA-256 algorithm. Padding includes adding bits to the message to make it a multiple of 512 bits.
- **Initialization**: An Initialization Vector (IV) is a predetermined set of initial values used to initialize the compression process in certain compression algorithms, ensuring consistent and reproducible compression results for the same input data.
- **Processing**: The padded message is divided into blocks of 512 bits, and the algorithm processes each block sequentially. The processing involves a series of logical and bitwise operations, including message expansion, data mixing, and bitwise operations such as AND, OR, and XOR.
- **Compression**: Each block is compressed using a combination of logical and bitwise operations, updating the state of the hash values.
- **Finalization**: Once all the blocks have been processed, the resulting hash value is obtained by concatenating the updated hash values.

Miners use Merkle trees (also known as Merkle hash trees) as an efficient and secure way to summarize a large set of transactions within a blockchain. These trees play a crucial role in the creation of the next block in the blockchain. Let's walk through the process:

- Miners gather valid transactions and sort them. They hash each transaction individually and combine the hashed values in pairs until a single root hash, known as the Merkle root, remains at the top of the tree.
- The Merkle root is included in the block header along with other block information. Miners attempt to find a nonce value that, when hashed with the block header, produces a hash that is less than the threshold.

6.1 Advantages of Merkle Trees

- Merkle trees enable efficient verification and tamper detection, making it difficult for a miner to modify a specific transaction without changing the Merkle root.
- If a miner want to add or Modify a transaction while solving the puzzle it can be done in O(log(M)) operations where M is no of transactions included.

- Light clients(New commer of Bitcoin) require fewer resources, enabling users with limited capabilities to participate in blockchain networks.

6.2 Prooving to Light Client

There is a situation that A is a light clinet and B need to proove that one of his transaction happend to C to A then B can provide A with a compact proof known as a Merkle proof.
This proof consists of the transaction, its corresponding Merkle path, and the Merkle root. A can then verify the validity of the transaction by using the Merkle proof to trace the transaction's inclusion in the Merkle tree and comparing the computed Merkle root with the one stored by A.

6.3 Full Client using Merkle Trees

If a full client want to discard any of the transaction he stored he can still proove that remaing transactions belonging to the same merkle tree which may not be happend if he stores in a naive way i.e concatenating all the transactions.

7 Bitcoin Transactions

7.1 Block Header contents

The block header in a blockchain typically contains the following elements:

- **Previous Block Hash**: Hash of the preceding block in the blockchain.
- **Merkle Root**: Hash of all the transactions included in the block.
- **Timestamp**: The time when the block was created, which is used after to find appropriate threshold.
- **Nonce**: A number used in the mining process to find a valid block hash.
- **threshold**: 4 Bytes are used to represnt threshold also known as bits, to save space instead of storing 256 bits.

$$bits = b_1 b_2 b_3 b_4$$

$$Threshold = b2_2 b_3 b_4 * 256^{b_1 - 3}$$

7.2 Digital Signatures

Digital Signatures are hard to forge, easy to create and easy to verify.
These signatures use the below three alogorithms to complete the need:

7.2.1 Generation of shared, public key pair

This uses GenerateKeys algorithm which takes key size as the parameter and generates a key pair in which shared key is kept secret and public key is known to everyone.

7.2.2 sign the message

This uses Sign algorithm which takes message and shared key as the algorithm and encypts the message, in this case message is typically a transaction.

7.2.3 verify the signature

This uses Verify algorithm which takes the message, signature and cipher text and verify the signature, this is done by everyone to verify transactions.

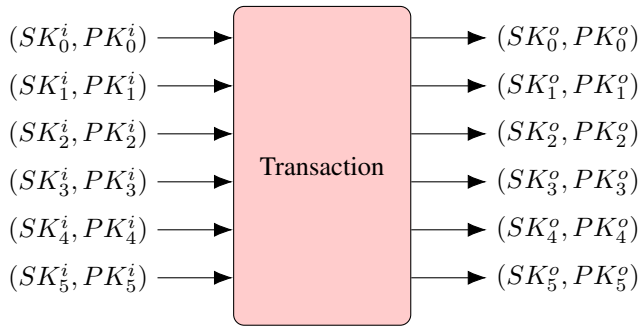
7.3 ECDSA

Elliptic Curve Digital Signature Algorithm is a cryptographic algorithm used for generating and verifying digital signatures, in which public key is 512 bits, shared key is 256 bits, signature is 512 bits and the message should be 256 bits.
Hash the message using SHA256 as many of the messages are not of 256 bits.

7.4 Transactions

Assume A(SK_A, PK_A) want to pay B(SK_B, PK_B), A has to make a transaction himself and sigh it with his shared key as discussed above.
Now the problem is how a miner verify this transaction is valid or not.

7.4.1 Structure of bitcoin transactions



Inputs are taken from earlier outputs of Bitcoin transactions, which consists hash of the transaction and index of the Block in which the transaction is present and some other feilds. Output pay Bitcoin to different public keys, each imput may correspond to multiple outputs. Reaming Bitcoin can be payed to himself by generating a new key pair, the transactions should be signed by shared keys of all inputs of the transaction.

7.4.2 Transaction fee

Transaction fees in Bitcoin are small amounts of Bitcoin paid by the sender of a transaction to incentivize miners to include and confirm the transaction on the network. The fee amount varies based on network congestion and transaction size.
If a sender wants to pay 1% of his output as Transaction fee he decreases the output by 1%.

7.4.3 Rules

- Output should not be negative in a transaction.
- sum of inputs must be greater than or equal to sum of outputs ina transaction.
- Whoever miner solves the puzzle can claim all the fee of the transcations he included and the mining fee.

7.4.4 Mining

Miner selects some of the pending transactions, compute merkle root and djusts nonce until the resulting block hash meets the threshold for net- work consensus, creating the next block.

7.4.5 Coinbase transaction

This the first transaction in which the miner claims the transaction fee and the mining reward, this transaction will not have any inputs it can create bitcoin i.e mining reward, which decreases exponentially so that no inflation occurs. Miner can claim his reward if he wins the game by creating next block.

7.4.6 Double spend prevention

There are set of rules which should be followed by miners so that double spend can be prevented these follows:

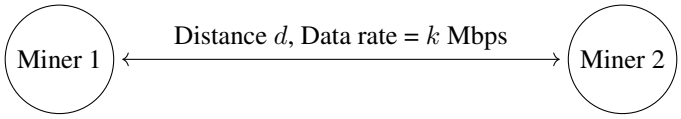
- Any output of a transaction should be only spend once, such outputs are calles UTXO(Unspend Transaction Output).
- If any miner includes invalid transaction and succeeded in creating new block every other miner should discard that block, if the block is in some chain the chain is also discarded.
- When a miner creates a block with all valid transactions every other miner should delete the transaction which are in the new block from their respective transaction pool.

From the above miner can also change some arbitrary feilds in the coinbase transaction to complete the proof of work.
The length of block header is 80 bytes, satoshi managed to keep it small.

8 Target Threshold

If the threshold is set too small it takes miners so much time to complete the proof of work as it takes so many trails, if it is set too large then forks may appear insted of chain we will have trees. Target threshold should not be to large or small.
Network delays while broadcasting block for verification should not be large because it may cause forks in the blockchain.

8.1 Network propagation



When miner 1 computes a block and want to transmit to miner 2 the delay is $\frac{d}{c} + \frac{b}{k}$, where b is the length of block size. Miner 2 verifies the block and broadcast it.
So the total time taken for end to end propagation of the block in peer to peer network is given below:

$$\sum_{i=1}^{n} \frac{d_i}{c} + \frac{b}{k_i} + p_i$$

where p_i is the time taken for verification of the block, and the links are selected by the spanning tree algorithm for avoiding multiple transfers.

8.2 How threshold is set

Thers should be no situation like some miner solves proof of work while other miner broadcasting his bolck, it may lead to forks and double spend. So threshold is choosen such that each block takes 10 minutes to create.
The threshold cannot be constant becuase as the users of bitcoin increase the hashing power increases, so satoshi decided to update threshold every 2 weeks based on hashing power. Users use **A6** hardware for solving proof of work. This 2 weeks is defined in terms of no of blocks, in 2 weeks roughly 2016 blocks were created.
So threshold is updated after every 2016 blocks as follows:

$$threshold = threshold * \frac{T}{1209600}$$

T is Time taken for creation of prev 2016 blocks in sec, T can be found using timestamp in block header as follows:

$$T = timestamp(B_{2016n-1}) - timestamp(B_{2016(n-1)})$$

8.3 Verification

Ideally miner should follow **NTP** for the timestamp while solving proof of work.

8.3.1 Rules for verification

Miner accepts the recived block if:

$$t_k > median(t_{k-1}, t_{k-2}, t_{k-3}.....t_{k-11})$$

$$t_k < 2hours + Netwokadjusttime$$

Nwtwok adjust time is defined as sum of miner local time according to NTP plus the median of the offsets with local times of his fellow peers.

8.4 Deal with forks

Satoshi's first rules were to slect longest chain if forks are present, if multiple longest chains were present should select the one with less timestamp.

8.4.1 Chain weight

If some attacker manages to create longest chain privately and broadcast afte he may kill the honest blocks to avoid this chain weight was used.
Chain weight uses the sum of expected work done for all the blocks in the chain, if it is built using fake timestamps he may get caught. expected work for creating i'th block is calculated as follows:

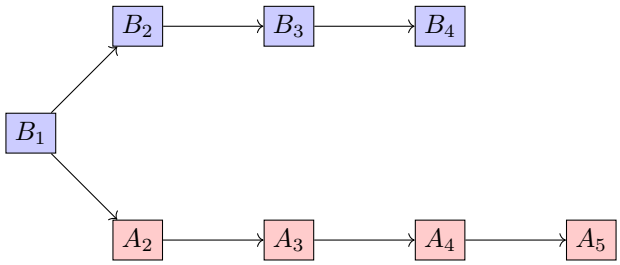
$$B_i = \lfloor \frac{2^{256}}{Threshold + 1} \rfloor$$

expected work for creating entire chain is calculated as follows:

$$\sum_{i=1}^{n} B_i$$

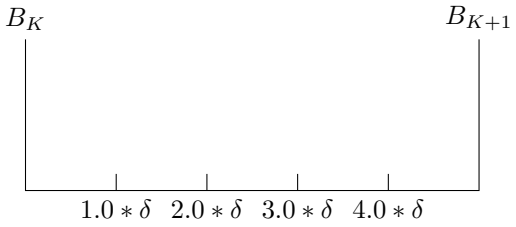
9 Confirmation Time

The confirmation time in blockchain refers to the amount of time it takes for a transaction to be validated and included in a block on the blockchain. The specific confirmation time can vary depending on the blockchain network and its consensus mechanism.
Zero Confirmation: A transaction that has been broadcasted but not yet added to a block or confirmed by the network.
First Confirmation: The first block in which a transaction is included, indicating its confirmation on the blockchain.
Second Confirmation: The second block in which a transaction is included, further validating its confirmation on the blockchain and increasing the level of confidence in its permanence.



9.1 Double spend attack

This attack was analysed by Satoshi himself, the above diagram shows the attack after B_1, so attacker creates another chain privately after B_1 to create double spend and kill the chain created by honest miners.
He creates until the other user confirms, i.e if the user waits till 3rd confirmation and attacker broadcast the transaction between B_1 and B_2, so the user checks if it is in B_4 are not, so attacker releases the chain after A_5 is created before b_5, otherwise he may not.



9.1.1 Inter arrival time of blocks

We want the distribution of seperation time between each block, let us say that we divided time into intervals of δ each and β is hashing power available.
The probabilty that a block is created in the time interval δ is $\beta\delta$, let us denote the inter arrival time as I so,

$$P[I = n\delta] = (1 - \beta\delta)^{n-1}\beta\delta$$

$$P[I > n\delta] = (1 - \beta\delta)^n$$

set $n\delta$ as x

$$P[I > x] = (1 - \frac{\beta * x}{n})^n$$

as n goes to ∞

$$P[I > x] = e^{-\beta * x}$$

9.1.2 Exponential distribution

The PDF is

$$P[I = x] = \beta * e^{-\beta * x}$$

The mean or expected value of the distribution is

$$E[x] = \frac{1}{\beta}$$

This distribution satisfies memoryless property

$$P(X > t + s|X > t) = P(X > s)$$

This property states that the probability of the random variable X exceeding a given threshold t+s, given that it has already exceeded t, is equal to the probability of X exceeding s.
From this we can conclude that when the transaction is broadcasted to the miners we should wait for $\frac{1}{\beta}$ which is roughly 10 minutes.

9.1.3 Poisson arrival process

The number of blocks created in an interval of T is suprisingly poisson distribution given as

$$P\left(A_T = n\right) = \frac{e^{-\beta * t}(\beta * t)^n}{n!}$$

9.2 51% Attack

let p be the fraction of hashing power with honest miners, $q = 1 - p$ be the hashing power with attackers, if $q > 0.5$ then attacker can always kill the chain of honest block.This attack is know as 51% attack.
The naive solution of this attack is to crete checkpoints is that slect a block which is deep enough in the chain must be present in the chain always, checkpoints are updated during a period of time.
It is very difficult to acheive more than 50% of hashing power. if attacker creates after time y and honest miner after x then probabilty of attacker winning the game is:

$$\begin{aligned} P[y < x] &= \int_0^\infty (1 - e^{-\beta * q * x})\beta * p * e^{-\beta * p * x}\, dx \\ &= \int_0^\infty \beta * p * e^{-\beta * p * x}\, dx - \int_0^\infty \beta * p * e^{-\beta * x}\, dx \\ &= 1 - p \\ &= q \end{aligned}$$

10 Satoshi's analysis

We want to know the number of confirmations user should wait such that his transaction will be in the blockchain forever.
When the transaction is about to enter the block chain attacker may be ahead or behind the honest miners.

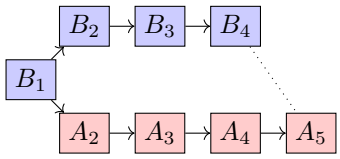


Figure 1: attacker is ahead of honest miners

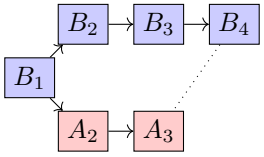
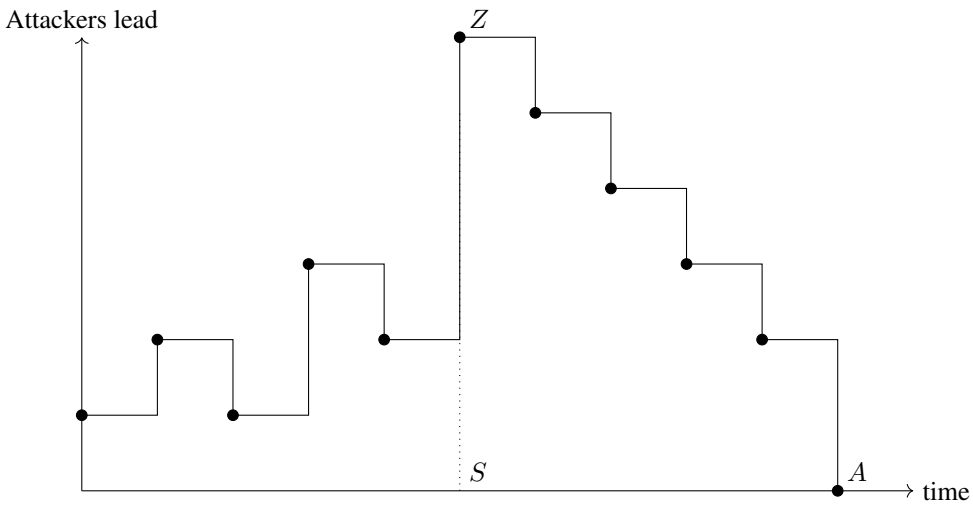


Figure 2: attacker is behind the honest miners

If attacker is behind the honest miners he should try create more blocks to kill the honest chain, if is ahead he will be wait until the transaction entered the block and releases the chain.

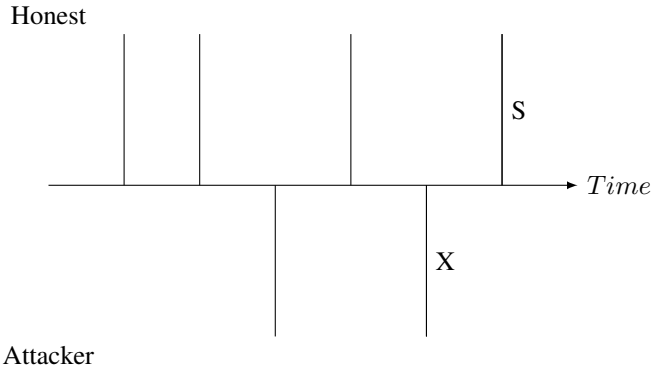
Satoshi assumed that network delay is zero and observed the attacker's lead with time.



From the above picture Z can be in the range $(-\infty, S)$, where S is indicating S'th confirmation, we also have to see after S if he can win at some point like A in above in future. S, A location is variable it can happen anywhere and it affects the value of Z.

10.1 Meni Rosenfeld discrete analysis

We will be marking points at which blocks were created and observe it when S number of blocks are created by honest miners.



If the honest miners are ahead of Z blocks at the S'th confirmation,

$$X = S - Z$$

we have to calculate the probabilty that honest miners have lead Z at S'th confirmation,

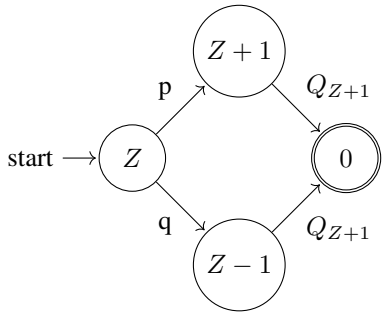
$$P_{Z,S} = Prob((S - X = Z)|(Honest = S))$$

Also the attacker catching up the lead Z as Q_Z.
Probability that attacker catches up after S honest blocks,

$$R_S = \sum_{-\infty}^{S} P_{Z,S} * Q_Z$$

$$P_{Z,S} = \binom{2 * S - Z - 1}{S - 1} * P^S * Q^{S-Z}$$

For finding Q_Z we should perform random walk as there are many possibilities to catch up the lead, this can analysed by an simple automeata as below,



So for $Z > 0$,

$$Q_Z = P * Q_{Z+1} + q * Q_{Z-1}$$

on solving,

$$Q_Z = \begin{cases} (\frac{q}{p})^Z & \text{if } Z \geq 0 \\ 1 & \text{if } Z < 0 \end{cases}$$

The above Q_Z is only true if $q < p$, else Q_Z is always 1 which leads to 51% attack.

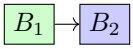


Figure 3: honest miners creates first

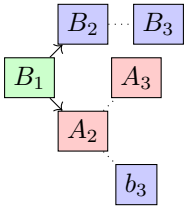


Figure 4: attacker and honest miners release simultaneously

11 Selfish mining

Selfish mining is a strategy where a miner intentionally withholds mined blocks to gain a competitive advantage and disrupt the fairness of the blockchain network. To perform selfish mining we should have an reasonable amount of hashing power.

11.1 Majority is not enough

This attack was analysed by Eyal and Siral around 2015.
If the block is created by honest miner before attacker creates a block, all the miners including attackers starts mining on this block as shown in figure 3.

If the attacker creates he waits and if the honest miner also creates some time after, if attacker has no lead he releases the block which leads all the attackers and some fraction of honest miners mining on his block, remaining fraction on honest block, as shown in figure 4. If B_3 is created first attacker chain is killed, similarly if b_3 or A_3 is created the honest miners chain is killed.

So if attacker had a lead 2 or more he waits until lead becomes 1 and releases the chain and kill the honest chain, each attacker block gets lead but not the honest blocks in this case attacker releases the chain as B_4a was created before A_6 as shown in figure 5.

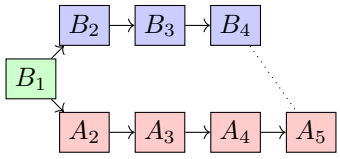
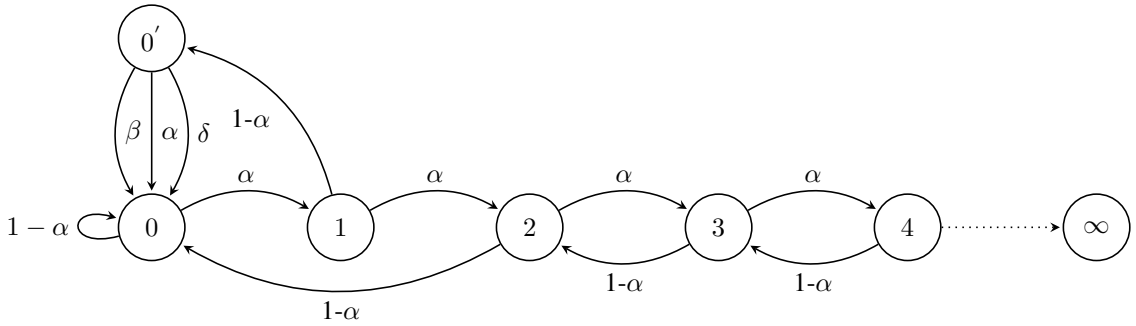


Figure 5: all the attacker blocks get into the chain

11.2 Markov chain

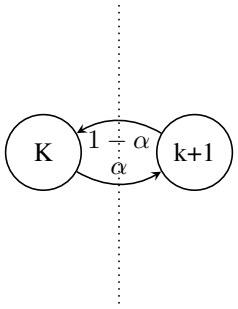


$$\beta = (1-\alpha)(1-\gamma)$$

$$\delta = (1-\alpha)\gamma$$

According to the paper attacker gets one block in each $(i, i+1)$, 2 blocks in (2,0) state change, honest miner may get 1,2 or 0 blocks in $(0^{'}, 0)$ state change depending on who creates the block.

11.3 Discrete time markove chain analysis



From the above figure if we make an vertical cut the flow should be equal from both sides in steady state so for $k \geq 2$,

$$P_k * \alpha = P_{k+1} * (1-\alpha)$$

make cut between 1 and $0^{'}$, 0 and 1,

$$P_{0'} = P_1 * (1-\alpha)$$

$$P_0 = (P_1 + P_2) * (1-\alpha)$$

on solving,

$$P_0 = \frac{\alpha - 2 * \alpha^2}{\alpha * (2 * \alpha^3 - 4 * \alpha^2 + 1)}$$

$$P_{0'} = \frac{(1-\alpha) * (\alpha - 2 * \alpha^2)}{2 * \alpha^3 - 4 * \alpha^2 + 1}$$

$$P_1 = \frac{\alpha - 2 * \alpha^2}{2 * \alpha^3 - 4 * \alpha^2 + 1}$$

for all $k \geq 2$,

$$P_k = (\frac{\alpha}{1-\alpha})^{k-1} * (\frac{\alpha - 2 * \alpha^2}{2 * \alpha^3 - 4 * \alpha^2 + 1})$$

11.4 Rewards

r_{pool} is defined as ratio of number of attacker blocks entered(A) int to final chain to the sum of all blocks generated by attacker(A_0) and honest miners(H_0).

$$r_{pool} = \frac{A}{A_0 + H_0}$$

$$r_{others} = \frac{H}{A_0 + H_0}$$

$$\frac{r_{pool}}{r_{pool} + r_{others}} = \frac{A}{A + H}$$

$$r_{others} = P_{0'} * (1-\alpha) * \gamma + P_{0'} * (1-\alpha) * (1-\gamma) * 2 + P_0 * (1-\alpha)$$

$$r_{pool} = P_{0'} * (1-\alpha) * \gamma + P_{0'} * 2 * \gamma + P_2 * (1-\alpha) * 2 + \sum_{2}^{\infty} P_i * \alpha$$

$$R_{pool} = \frac{A}{A + H}$$

$$R_{pool} = \frac{\alpha * (1-\alpha)^2 * (4 * \alpha + \gamma(1 - 2 * \alpha)) - \alpha^3}{1 - \alpha * (1 + (2 - \alpha)) * \alpha}$$

Where α is hashing power of attackers pool and γ is the fraction of honest miners who follow attackers after releasing attackers chain.
In the context of Bitcoin, a mining pool is a group of miners who come together to combine their computing power and resources in order to increase their chances of successfully mining new blocks and earning Bitcoin rewards. Here's how a mining pool typically operates:

1. Miners join the pool: Individual miners join a mining pool by connecting their mining hardware (such as specialized computers called ASICs) to the pool's mining server.

2. Solving hash puzzles: The mining pool's server assigns computational tasks to the connected miners, typically in the form of hash puzzles.

3. Work distribution: The pool's server distributes different variations of the hash puzzle to the connected miners.

4. Finding a block: Once a miner in the pool successfully solves a hash puzzle, they broadcast the solution to the pool's server.

5. Distribution of rewards: Rewards are distributed among pool participants based on the proportion of hashing power they contributed.

6. Continuous mining: Miners in the pool continue to receive new hash puzzles and work on solving them to find more blocks and earn rewards.

12 Proof Of Stake

12.1 Sybil attack

In a blockchain, a Sybil attack involves an attacker claims to have more hashing power than what he actually have.
In proof of work Sybil attack cannot be performed because someone cannot fake real time work done by them, but the problem with proof of work is it consumes more energy.
Proof of stake(PoS) is a consensus protocol unlike Proof of Work (PoW), which relies on miners solving complex mathematical puzzles, PoS selects validators to create and validate new blocks based on the amount of cryptocurrency they hold and "stake" as collateral.
Proof of work uses hash of previous block, nonce(IV), timestamp(T), merkle root of the transactions(MR) to solve the puzzle as follows,

$$Hash(Hash(B_k), IV, T, MR, ...) < Threshold$$

Proof of stake uses the hash of previous block, public key(a), balance with a, timestamp and merkle root is not included, but the threshold is varied for miners because people with more stake should have more chance, also timestamp should be near to local time.

$$Hash(Hash(B_k), a, T, ...) < Threshold * balance(a)$$

12.2 Nothing at stake

The "nothing at stake" problem is a concern in Proof of Stake (PoS) blockchains where validators have no financial disincentive to support multiple conflicting forks. Since PoS validators are not required to spend computational resources to mine blocks, they can potentially support all competing chains simultaneously. This lack of economic cost removes the incentive to converge on a single valid chain, leading to a lack of consensus. To mitigate this problem, PoS protocols implement penalties or slashing conditions to discourage validators from supporting multiple forks, ensuring network stability and security.

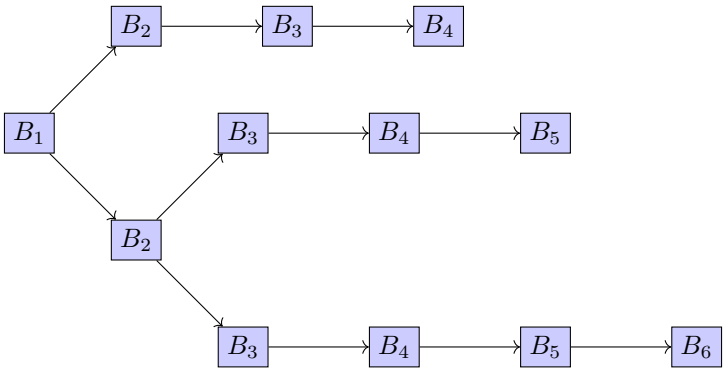


Figure 6: Nothing at stake attack

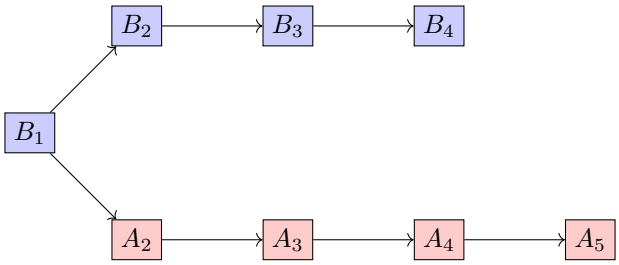


Figure 7: Long range attack killing blue chain after B_1

12.3 Long range attack

A long-range attack is a security concern in blockchain systems, where an attacker attempts to rewrite the entire blockchain's history. By possessing a significant amount of stake or historical data, the attacker creates an alternative chain from an earlier point in time to surpass the current valid chain. This can invalidate previous transactions and compromise the integrity of the network as shown in figure 7. HAs stake goes up so attacker might approach real time in creating blocks i.e creating bloks at every 10 minutes and he can easily kill the entire chain easily.

12.4 Initial distribution problem

The initial distribution problem in Proof of Stake (PoS) refers to the challenge of fairly allocating cryptocurrency among participants at the launch of a PoS blockchain. Concentration of stake by early adopters can lead to centralization and disproportionate control. Strategies like fair token distribution and airdrops are used to promote decentralization and prevent dominance by a few stakeholders.

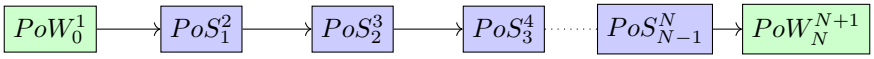


Figure 8: Hybrid solution

12.5 Bribe attack

A bribe attack is a type of attack where an adversary attempts to manipulate the behavior of participants in a blockchain network by offering them incentives or bribes. The attacker aims to persuade participants to act in a way that benefits the attacker's interests. This attack can undermine the integrity and security of the blockchain consensus. Countermeasures include promoting transparency, strong governance mechanisms, and encouraging participants to act in the best interest of the network rather than personal gain.

12.6 Pre computing attack

A pre-computing attack in Proof of Stake (PoS) refers to a potential attack where an adversary tries to gain an unfair advantage by pre-computing a large number of block proposals or validations before they are actually required. By pre-computing blocks as he can change the transaction anytime in previous block, the attacker can quickly gain control over the blockchain and influence the consensus process.

13 Hybrid PoW and PoS consensus

Use proof of work but not too much of energy i.e don't use proof of work in every block, also use proof of stake for consensus protocol. The proposed hybris solution is use PoW after every N blocks in between use Pos as shown in figure 8.
By using this the PoW energy per unit time is changed to $\frac{M}{NT}$ from $\frac{M}{T}$ where M is mining fee per block and T ia average time per creating a block.

14 SLASHER

The concept of "slasher" was proposed by Vitalik Buterin, the co-founder of Ethereum. Slasher is a type of consensus mechanism that was introduced as an alternative to the traditional Proof-of-Work (PoW) algorithm used by Bitcoin. Slasher focuses on penalizing validators who act dishonestly or attempt to manipulate the blockchain.
Reducing mining reward which could potentially reduce the overall computational power and energy consumption required for mining operations.
In slasher each valid block is signed by some group of people selected by proof osf stake beforehand and created by proof of work, so double spend attack is non trivial as attacker should create blocks with PoW and should have signatures.
As shown in figure 9 the blocks are signed by designated signers after the creation of the block, if any person signs multiple blocks at the same level of the fork he will be penalized.

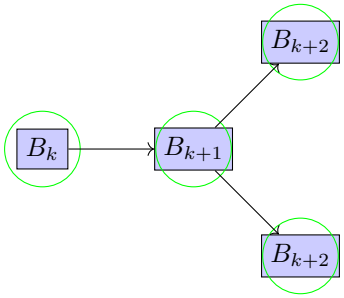


Figure 9: Signing the blocks

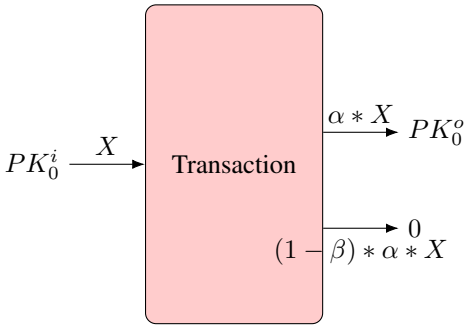


Figure 10: punishment transaction

All the signers selected by PoS has to deposit stake for some duration. The stake is frozen during the time i.e from B_k to B_{k+n} block. If anyone cheats during the time the stake is confiscated by creating a punishment transaction which involves some part to the user who created the transaction, miner who includes the transaction as shown in figure 10. α, β are in range $(0, 1)$ so the part of money paid to shared key 0 is burnt.

14.1 Rules for signing

A group signers are selected for signing B_{K+M} at B_K using proof of stake, assuming that a large number of these signers are honest, a block is valid if and only if it has at least $\frac{2}{3}$ group signatures, the group is called as committee to sign.

14.2 Choosing the committee

If the committee for B_k is selected as B_{k-1} there is a chance of selecting two committee if a fork is present as shown in figure 11, so we go deep into the block chain and select the committee as there is a less chance that there is a long fork from the deep enough block.

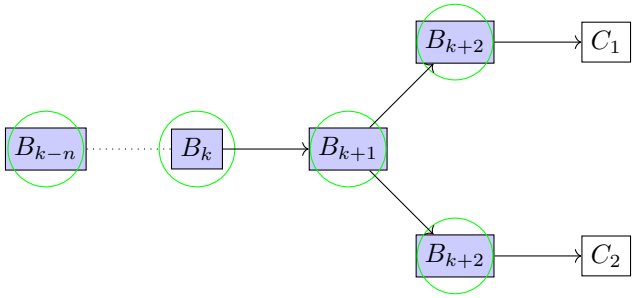


Figure 11: selecting multiple committee at forks

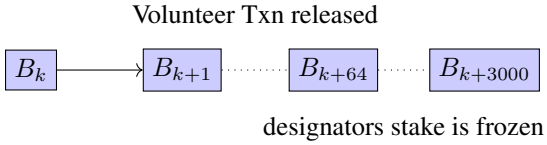


Figure 12: designated signers selection

14.3 Slasher rules

- Blocks are mined using PoW, mining reward decreases by a factor of 50 every year.
- Every block has set of designated signers choosed beforehand.
- A block is valid if it has more than $\frac{2}{3}$ of signatures from the designated signers.

14.4 Potential signers

When B_k is produced the potential signers of block B_{k+3000} are those with address a,

$$Hash(a, Hash(B_k)) < bal(a) * Target$$

we need about 15 potential signers to volunteer to become designated signers, so Target is adjusted such that abou 15 people volunteer every time.

15 Designated signers

Any potential signer of B_{k+3000} can be a designated signer by sending a volunteer transaction between B_{k+1} and B_{k+64}, the deposits are frozen from B_{k+64} to B_{k+6000} of the designated signers, there will be no signers fro first 300 blocks. what if none of the block has $\frac{2}{3}$ signatures or every block has more than $\frac{2}{3}$ signatures then miners should start mining on the parent block by skipping the blocks at the same level.
If they skip n blocks then the new threshold will be

$$Target = Target * 8 * 2^{n-1}$$

16 Anonimity

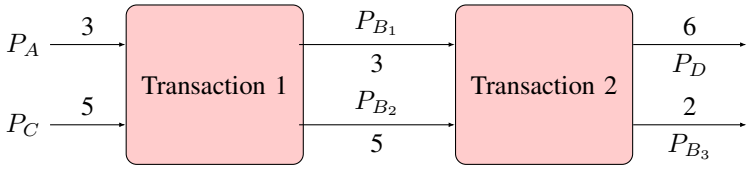
Blockchain provides pseudonymity, where users are identified by cryptographic addresses instead of personal information, but it is not inherently anonymous as transactions can be traced through blockchain analysis techniques.

16.1 Properties of unlinkability

- Hard to link different addresses of the same user.
- Hard to link different transactions of the same user.
- Hard to link sender and recipient of the same transaction.

16.2 De-anonymization

If recipient address is well known then it is easy to track his all transaction as they are part of directed graph. To avoid this users always change their address at each transaction.



In the above transaction show we can easily know the P_{B_1}, P_{B_2} belong to the same user as he is combining them to pay, some wallets by default generates a new change address and keeps it randomly among the outputs to fix this issue, but if there is only one new address among the output then one can de-anonymize them easily.

16.2.1 network de-anonymization

Network deanonymization in a P2P network means figuring out who is behind the network activities. It can be done through analyzing traffic patterns, creating fake identities, or tracking IP addresses.
The Tor network and onion routing are used to enhance privacy by obscuring the origin and destination of network traffic, making it difficult to link Bitcoin transactions to specific individuals.

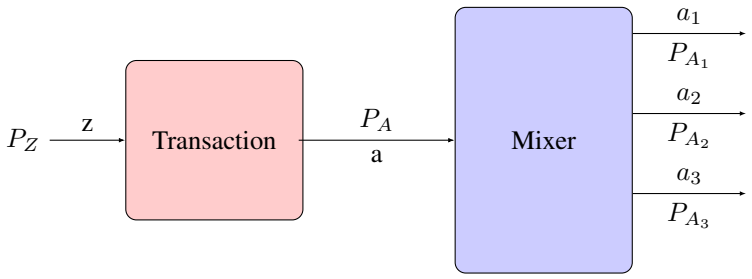
16.3 Tor network & Onion routing

The Tor network is a decentralized network of volunteer-operated servers that routes internet traffic through multiple layers of encryption and relays to protect user privacy and anonymity.
Onion routing is a technique employed by the Tor network where data is encrypted in

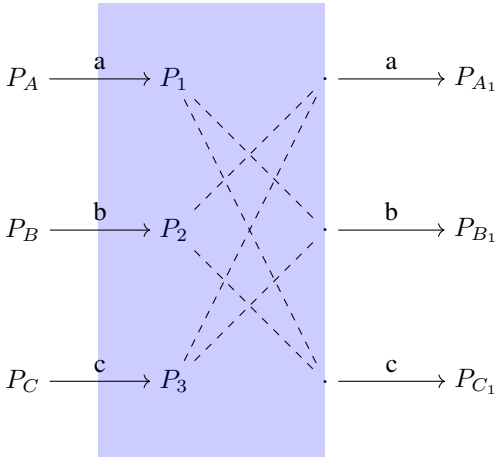
multiple layers (like the layers of an onion) and routed through a series of relays, with each relay peeling off a layer of encryption, making it challenging to trace the source or destination of the traffic. This helps in preventing network deanonymization in Bitcoin by obscuring the true origin and destination of transactions.

17 Mixing

Mixing is done in blockchains to enhance privacy and anonymity. By mixing or combining transactions from different participants, we don't want a mapping between users and their addresses.



In the above picture P_A recieves some money a from P_Z, P_A approaches the mixer to split it's money among the public keys provided by himself, mixer take cares that any person other than A know that theys addresses belong to A. We assume that mixers are honest and pay back the money.
All the money from the users are shuffled and paid to their new addresses provided by them.



In the above figure it shows how a mixer works, as shown if three members appear and the money is paid back as shown above so that there is no link between P_A, P_{A_1}.

17.1 Principles of mixing

- Use series of mixing to avoid side channel attacks, hope at least one mixer provides the anonymity.
- Use the same bitcoin input to the mixer because attacker can track the money taken by all users and find them, the sixe accepted by the mixer is called chunk size.

As the mixer collects its mining fee and looses the transaction fee for creating many of them so the output is less than input, if some user wants series of mixing he has to add some money which is not a good idea because anonymity is lost again.
So the solution for the above situation is "Fees are all or nothing", in this the mixture take everything from 1% of number of the users of the mixer and compensates it to mixing, transaction fee, so there is 99% chance of getting full back and 1% chance of loosing everything.

17.2 Continual mixing

If a client visits a mixing service on days 1, 3, and 5, and creates three new keys before combining them to make a payment, there could be a risk of getting caught if someone has access to information about the client's visits to the mixing service, the solution for this is the clients wallet should go on mixing every day so that attacker cannot correlate those transactions.

17.3 Decentralized mixing

A group of themselves performs the mixing process by jumbling transactions together, enhancing privacy and anonymity using peer to peer network. They use coinjoin method to perform this.

17.3.1 Coinjoin

- Find peers who want to perform mixing.
- Exchange input and output public keys.
- Construct the transaction and send it around themselves to sign it.
- Broadcast it to the miners.

While exchanging input and output address user should use multiple ip address or tor network to send them, both the input and output should not be sent at same time so anonymity is preserved.

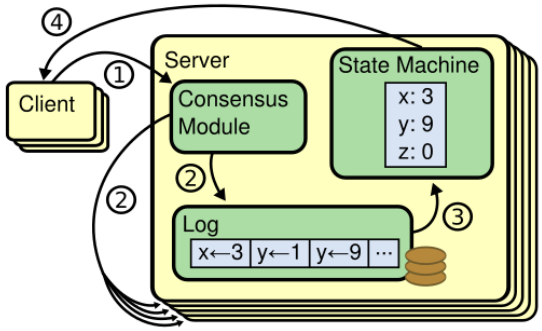


Figure 13: Replicated state machine architecture

18 RAFT

18.1 Replicated state machine

Replicated state machines are typically implemented using a replicated log. Each server stores a log containing a series of commands, which its state machine executes in order. Each log contains the same commands in the same order, so each state machine processes the same sequence of commands. Since the state machines are deterministic, each computes the same state and the same sequence of outputs.
The consensus module on a server receives commands from clients and adds them to its log. It communicates with the consensus modules on other servers to ensure that every log eventually contains the same requests in the same order, even if some servers fail. Once commands are properly replicated, each server's state machine processes them in log order, and the outputs are returned to clients.

18.2 Consensus

Raft implements consensus by first electing a distinguished leader, then giving the leader complete responsibility for managing the replicated log. The leader accepts log entries from clients, replicates them on other servers, and tells servers when it is safe to apply log entries to their state machines. Raft decomposes the consensus problem into two relatively independent subproblems,

- **Leader election**: a new leader must be chosen when an existing leader fails.
- **Log replication**: the leader must accept log entries from clients and replicate them across the cluster, forcing the other logs to agree with its own.

Raft servers communicate using remote procedure calls (RPCs).

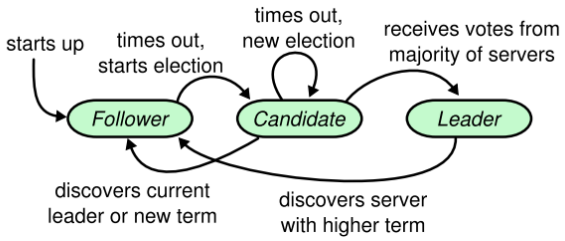


Figure 14: Server states

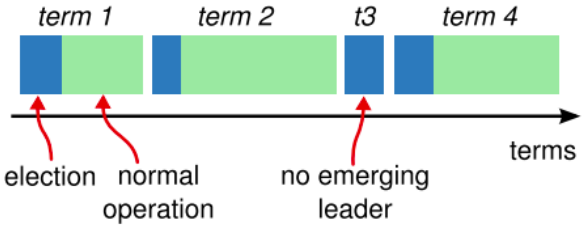


Figure 15: Time is divided into terms

18.3 Leader election

Raft uses heartbeat to trigger leader election; servers start as followers. Followers stay in this state as long as they receive valid RPCs from a leader or candidate. Leaders send periodic heartbeats (AppendEntries RPCs with no log entries) to maintain authority. If a follower experiences an election timeout, it transitions to candidate state, increments its term, and votes for itself. The candidate issues RequestVote RPCs to other servers. The candidate remains in this state until it wins the election, another server becomes leader, or a timeout occurs.
To win an election, a candidate must receive votes from a majority of servers in the cluster for the same term. Each server votes for at most one candidate in a term, following a first-come-first-served basis. The majority rule guarantees only one candidate can win the election for a specific term. When a candidate wins, it becomes the leader and sends heartbeats to establish authority and prevent new elections, if no one wins there will be another election after a timeout, term is increased after every election, term is also put in messages, term also increases if no candidate was elected.

18.4 Logs

Each log entry stores a state machine command along with the term number when the entry was received by the leader, log entry also has an integer index identifying its position in the log.

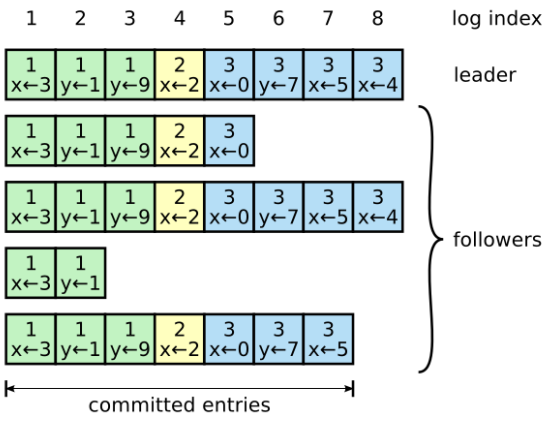


Figure 16: Logs with thier composed of entries

18.5 Logs

18.6 Log replication

After being elected, the leader handles client requests by appending the command to its log as a new entry. It then replicates the entry by sending parallel AppendEntries RPCs to the other servers. Once the entry is safely replicated, the leader applies it to its state machine and returns the execution result to the client,in the event of follower crashes, slow performance, or lost network packets, the leader persists in retrying AppendEntries RPCs indefinitely until all followers successfully store all log entries.
A leader determines when it is safe to apply a log entry to the state machines, referred to as committed. Committed entries are guaranteed to be durable and eventually executed by all available state machines. An entry is considered committed when the leader has successfully replicated it on a majority of the servers. This commitment also applies to preceding entries in the leader's log, including those created by previous leaders.
If any leader fails the new leader elected should have all the committed requests from the start, so while election candidates will send their log, others should vote him if their log is subset of it. Once the new leader is elected with atleast half of votes he updates others log by sending his log.
To ensure coherency and safety, Raft maintains the Log Matching Property by guaranteeing that if two entries in different logs have the same index and term, they store the same command, and the logs are identical in all preceding entries.
During normal operation, the logs of the leader and followers remain consistent, ensuring that the AppendEntries consistency check never fails. However, leader crashes can cause inconsistencies in the logs, as the old leader may not have fully replicated all entries. These inconsistencies can accumulate over multiple leader and follower crashes, resulting in followers having missing or extra entries compared to the new leader.
To synchronize a follower's log with the leader's, the leader identifies the latest agreed-upon entry in both logs, deletes any entries in the follower's log after that point, and

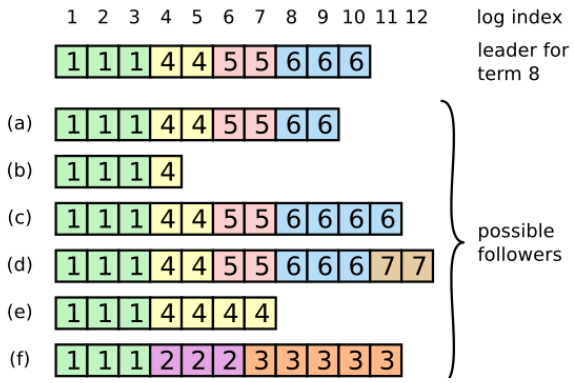


Figure 17: Possible scenarios (a–f) in follower logs when a new leader comes to power: missing entries, extra uncommitted entries, or both.

sends all its own entries from that point onwards to the follower.

18.7 Timing and availability

Safety in Raft must not rely on timing, but availability does. Leader election is critical, and Raft requires that broadcastTime is much smaller than electionTimeout, which in turn is much smaller than MTBF. (broadcastTime $<<$electionTimeout $<<$ MTBF)

19 Byzantine Fault Tolerance

Since malicious attacks and software errors can cause faulty nodes to exhibit Byzantine (i.e., arbitrary) behavior, Byzantine-fault-tolerant algorithms are increasingly important. Let us assume there are n servers and see malicious and rational as Byzantine rest are honest.

19.1 consensus requirements

- **Agreement**: all honest must reach same decision.
- **Termination**: all honest must eventually make a decision.
- **Validity**: if leader is honest, no matter what all other honest ones should follow the leader.

The algorithm offers both liveness and safety provided at most f out of a total of n replicas are simultaneously faulty. This means that clients eventually receive replies to their requests and those replies are correct according to linearizability. The algorithm works in asynchronous systems like the Internet and it incorporates important optimizations that enable it to perform efficiently. After communicating with n-f replicas twice, since f replicas might be faulty and not responding, the minimum overlap between them will

be n-2f in the worst case f faulty replicas may be in this overlap so honest replicas are at least n-3f. so we can proove that

$$f <= \lfloor \frac{n-1}{3} \rfloor$$

20 Impossibility result

It is impossible to have a deterministic protocol that solves consensus in a message passing asynchronous system in which at most one procees may file by crashing.

21 PBFT

PBFT describes a new replication algorithm that is able to tolerate Byzantine faults.

21.1 Quorums

There are n = 3f + 1 nodes which are trying to achieve consensus, where f is the maximum number of byzantine nodes allowed.
The set of 2f + 1 or more nodes which agree on something is called a quorum.

21.2 Properties of quorums

21.2.1 Intersection property

Any two quorums have at least one honest (non byzantine) node in their intersection.
Let us assume A, B are two quorums and if their intersection has no non byzantine node then

$$|A| \geq 2f + 1$$
$$|B| \geq 2f + 1$$
$$|A \cap B| \leq f$$

from the above

$$\begin{aligned} |A \cup B| &= |A| + |B| - |A \cap B| \\ &\geq 2f + 1 + 2f + 1 - f \\ &\geq 3f + 2 \\ &\geq n + 1 \end{aligned} \tag{1}$$

Which is clearly a contradiction so so any two quorums has at least one honest node in intersection.

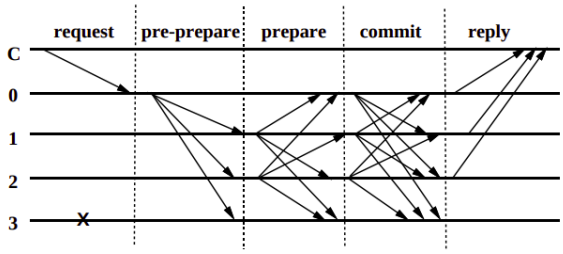


Figure 18: Normal Case Operation

21.2.2 Availability property

There exists always a quorum which has no byzantine in it. If a node hears a quorum(Q_1) agreeing on something, then there cannot be another quorum(Q_2) disagreeing to quorum Q_1, this can be proven from the intersection property.
if $Q_1 \cap Q_2$ contains at least one honest node, as a honest node will not contradict itself this cannot happen. From the availability property we can guarantee liveness because byzantine nodes cannot prevent quorums from forming by being silent.

21.3 PBFT details

PBFT guarantees saftey because replicated service behaves as a centralized implementation that executes operations correctly one at a time, PBFT also guarantees liveness assuming week synchronous.

21.4 Normal Case Operation

In the Normal Case Operation of PBFT, all correct replicas follow the three-phase protocol to achieve consensus on client requests. The stages of PBFT and their corresponding steps:

- **Request**: Client sends a request to the primary replica.
- **Pre-Prepare**: Primary replica assigns a sequence number to the request and multicasts a Pre-Prepare message to other replicas.
- **Prepare**: Replicas verify and multicast Prepare messages to vouch for the correctness of the Pre-Prepare message by observing a quorum.
- **Commit**: Replicas multicast Commit messages after receiving a sufficient number of valid Prepare messages, indicating the request's commitment by observing a quorum.
- **Reply**: Upon receiving f + 1 valid Commit messages, a replica executes the request, sends a Reply message to the client with the result, and considers the request committed.

21.4.1 request

A client c requests the execution of state machine operation o by sending a $\langle request, o, t, c \rangle_{\sigma_c}$ message to the primary. Timestamp t is used to ensure exactly once semantics for the execution of client requests. Timestamps for c's requests are totally ordered such that later requests have higher timestamps than earlier ones.

21.4.2 reply

Replicas send messages with current view numbers to track the primary. The primary multicasts requests to backups atomically.
A replica sends the reply to the request directly to the client. The reply has the form $\langle REPLY, v, t, c, i, r \rangle_{\sigma_i}$ where v is the current view number, t is the timestamp of the corresponding request, i is the replica number, and r is the result of executing the requested operation. The client waits for f+1 replies with valid signatures from different replicas, and with the same t and r, before accepting the result r. This ensures that the result is valid, since at most f replicas can be faulty.

21.4.3 pre - prepare

In the pre-prepare phase, the primary assigns a sequence number, n , to the request, multicasts a pre-prepare message with m piggybacked to all the backups, and appends the message to its log. The message has the form $\langle \langle PRE - PREPARE, v, n, d \rangle_{\sigma_p}, m \rangle$, where v indicates the view in which the message is being sent, m is the client's request message, and d is m's digest. A backup accepts a pre-prepare message provided:

- the signatures in the request and the pre-prepare message are correct and d is the digest for m.
- it is in view v
- it has not accepted a pre-prepare message for view v and sequence number n containing a different digest
- the sequence number in the pre-prepare message is between a low water mark,h , and a high water mark, H.

21.4.4 prepare

Replica's enters the prepare phase by multicasting a $\langle PREPARE, v, n, d, i \rangle_{\sigma_i}$ message to all other replicas and adds both messages to its log. Otherwise,it does nothing.
A replica accepts prepare messages for its log if they meet the criteria of correct signatures, matching view numbers, and valid sequence numbers.
We define the predicate prepared(m, v, n, i) to be true if and only if replica i has inserted in its log: the request m, a pre-prepare for m in view with sequence number n, and 2f prepares from different backups that match the pre-prepare.
If prepared(m, v, n, i) is true then prepared($m^{'}$, v, n, j) is false for any non-faulty replica (including i=j) and any $m^{'}$ such that $D(m) \neq D(m^{'}$. This is true because prepared(m,

v, n, i) and $|R|$ = 3f+1 imply that at least f+1 non-faulty replicas have sent a pre-prepare or prepare for m in view v with sequence number n. Thus, for prepared($m^{'}, v, n, j$) to be true at least one of these replicas needs to have sent two conflicting prepares, i.e., two prepares with the same view and sequence number and a different digest. But this is not possible because the replica is not faulty. Finally, our assumption about the strength of message digests ensures that the probability that $m = m^{'}$ and $D(m) = D(m^{'}$ is negligible.

21.4.5 Commit

Replica i multicasts a $\langle COMMIT, v, n, D(m), i \rangle_{\sigma_i}$ to the other replicas when prepared(m, v, n, i) becomes true.
committed(m, v, n) is true if and only if prepared(m, v, n, i) is true for all i in some set of f+1 non-faulty replicas; and committed-local(m, v, n, i) is true if and only if prepared(m, v, n, i) is true and i has accepted 2f+1 commits from different replicas that match the pre-prepare for m, if committed-local(m, v, n, i) is true for some non-faulty then committed(m, v, n) is true.
This ensures that all nonfaulty replicas execute requests in the same order as required to provide the safety property. After executing the requested operation, replicas send a reply to the client. Replicas discard requests whose timestamp is lower than the timestamp in the last reply they sent to the client to guarantee exactly-once semantics.