Principles of Blockchains
Princeton University,
Professor: Pramod Viswanath

Notes - complete edition

July 22, 2023

Contents

Chapter 1

Introduction

1.1 What are blockchains

Blockchain is a bit different with other subjects, in the sense that it needs to be first defined. It is remarkable and so many people try to define it in their own way. Our definition of blockchain is as follows :

Definition 1.1.1: Blockchain

Blockchains are the technology underlying **decentralized** **digital trust** **platforms**.

Decentralized system

A decentralized system is one where no single entity (person/company) is responsible for the smooth operation of the system. Indeed, blockchains are peer-to-peer systems, where each peer has the same prescribed behavior; no peer is unique. Peers communicate with each other by exchanging messages. Beyond this message exchange, peers function independently of one another.

Trust

Trust is a medium that drives human interactions. Human success is based on flexible cooperation in large numbers. This requires trust. The underlying mechanism of trust dictates how large the size of human cooperation can get.

Types of trust

- **Tribal Trust :**
  This is the traditional version of trust. People who shared similarities in language, generics compositions and customs, organized themselves around tribal societies. It was not very scaled in number or geographical place and was rather local.
- **Institutional Trust :**
  Since the end of WWII, the dominant human organization has been institutional. The underlying mechanism of trust is a set of laws and transparent and rigorous enforcement of the law. “no one is above the law”.
  This notion of trust has enabled human societies to cooperate on a global scale. It was extended in every aspect of human activity, travel, commerce, communication.
  However, the drawback is that the institutions are centralized, and the power to enforce and promulgate the laws is concentrated in the hands of a few individuals. The concentration of power leads to unintended consequences, including corruptibility, autocratic tendencies, and rent-seeking.
- **Distributed trust :**
  The siren song of blockchains is that the scaling and flexibility of institutional trust are possible without its negative aspects, i.e., the creation of a decentralized trust system. This is where blockchains play a significant role.

Figure 1.1: Evolution of Trust over human history

Blockchains are guaranteed to be secure even if some fraction of peers act maliciously. The only requirement is that sufficiently many peers operate according to the prescribed behavior. This is called the honest majority assumption. Thus, any user of a blockchain does not need to trust all peers in the system or even one particular peer. Instead, it just needs to trust that a majority of peers are honest. This is a key feature of blockchains that are not present in other peer-to-peer systems, or indeed, any other system.

Digital platforms

Digital platforms are marketplaces where suppliers and consumers come together and trade. A digital platform is one in which two (or more) parties interact, and some transaction takes place. There are many examples of digital platforms today. For example, Amazon/eBay is a platform for the exchange of goods. It provides a place where sellers can list goods that they are selling, and buyers can choose to buy any listed goods. Among other things, the platform ensures that the transaction takes place securely. It also handles disputes in transactions. Other examples of platforms are Uber/Lyft (for rides), AirBnB (for temporary accommodation), etc. Digital platforms have played a significant role in the global economy in the last decade.For the last decade and beyond, platform companies have been the dominant force in the economy. Here we put the top five, six companies by market cap. They're all platform companies. The aforementioned digital platforms are not truly decentralized in the sense,

2022 September Top US companies by market cap		2011 Top US companies by market cap	
1. Apple	$2.5 T	1. Exxon	$417 B
2. Microsoft	$1.9 T	2. Apple	$321 B
3. Alphabet	$1.4 T	3. Chevron	$215 B
4. Amazon	$1.3 T	4. Microsoft	$213 B
5. Tesla	$0.8 T	5. IBM	$207 B
6. Facebook(???)	$0.4 B	6. Walmart	$204 B

Figure 1.2: Evolution of Trust over human history

they do not offer distributed trust. By using any platform, we implicitly trust the (single) company behind that platform to handle transactions faithfully. The platform holds a lot of power in arbitrating disputes. It can also arbitrarily decide the fees to charge for the service. (Of course, the reputation/popularity of the platform is at stake, which prevents it from behaving arbitrarily).

Blockchains hold the promise to decentralize the digital platforms we see today. From a user's perspective,

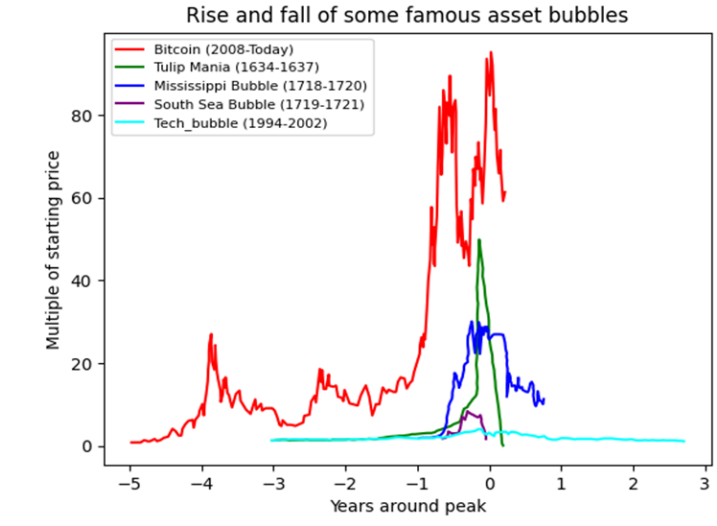

Figure 1.3: Bitcoin and Bubbles

the platform's functionality will be identical. However, the platform will no longer be operated by a single company but rather a multitude of small stakeholders, each running the same blockchain code. The 'trust' factor in the system becomes distributed.

1.2 Bitcoin as the first blockchain

The term blockchain was introduced in late 2008 with the advent of Bitcoin. The inventor of bitcoin is Satoshi Nakamoto, a pseudonym for the person or persons who developed the cryptocurrency, authored the bitcoin white paper, and created and deployed the original bitcoin software. Bitcoin is a cryptocurrency: a decentralized, digital payment platform. As such, cryptocurrencies are one of the simplest applications of blockchains. Today, there is a multitude of blockchain designs for cryptocurrencies and other applications. However, they all retain many of the core design components that were introduced in Bitcoin. Thus, the term 'blockchain' has remained in all of them.

Bitcoin is one among many attempts in history to create a decentralized, digital payment platform (the term 'currency' is to be thought of as a 'token' that is used on this payment platform). Unlike all previous attempts, Bitcoin has stood the test of time. Its popularity, which can be measured by its price in dollars, has grown many-fold over the twelve years since its introduction.

It may seem that the fall and rise in the price of Bitcoin is a bubble, but based on Figure 1.3, the red graph that shows the changes in the price of Bitcoin says that the price of Bitcoin has gone up and down, but other bubbles burst at once and their prices do not rise much. This shows that Bitcoin is stable. Maybe it can be said that this is the mother of all bubbles. A major reason behind the popularity of Bitcoin is its strong security property, coupled with its truly decentralized nature. It is now well understood that as long as 51% of the peers in Bitcoin are honest, the system is secure (the exact conditions are slightly different, but this suffices for the moment). This has been shown theoretically and has also been borne out in practice. Moreover, anyone can freely join and leave the Bitcoin system at any time; the system is **'permissionless'**.

Some properties of Bitcoin and its performance are as follows:

1. **Security**: We will discuss what does it mean that bitcoin is secure in future notes but generally bitcoin is secure as it is almost always accessible and it have never been broken.

2. **Transaction throughputs**: Bitcoin has transaction of 7 tx/s which is noticeably low in amount. (See figure 1.4)

3. **Confirmation Latency** : Bitcoin has a very slow latency which can take hours before something gets confirmed.
4. **Energy consumption**: Bitcoin and it's mining is famous for its large energy consumption. It's that of a medium sized country.
5. **Computation and storage**: Mining bitcoin requires huge resources of computing and storage.
6. **Communication**: Communication requirements are also fairly high. Everybody's transmitting everything and every possible message.

Despite its benefits, there are major drawbacks of Bitcoin, which impact its viability. Some issues can be categorized as scaling issues. For example, the system can only process about seven transactions per second. In contrast, Visa (a centralized digital payment mechanism) processes 50,000 transactions per second. If all people in the world are to switch from Visa to Bitcoin, the system must 'scale' its transaction throughput. Other issues are a lack of desirable properties. For example, if the network is disrupted, transactions that are once 'confirmed' can be reverted. (We say that Bitcoin does not offer 'finality' in confirming transactions).

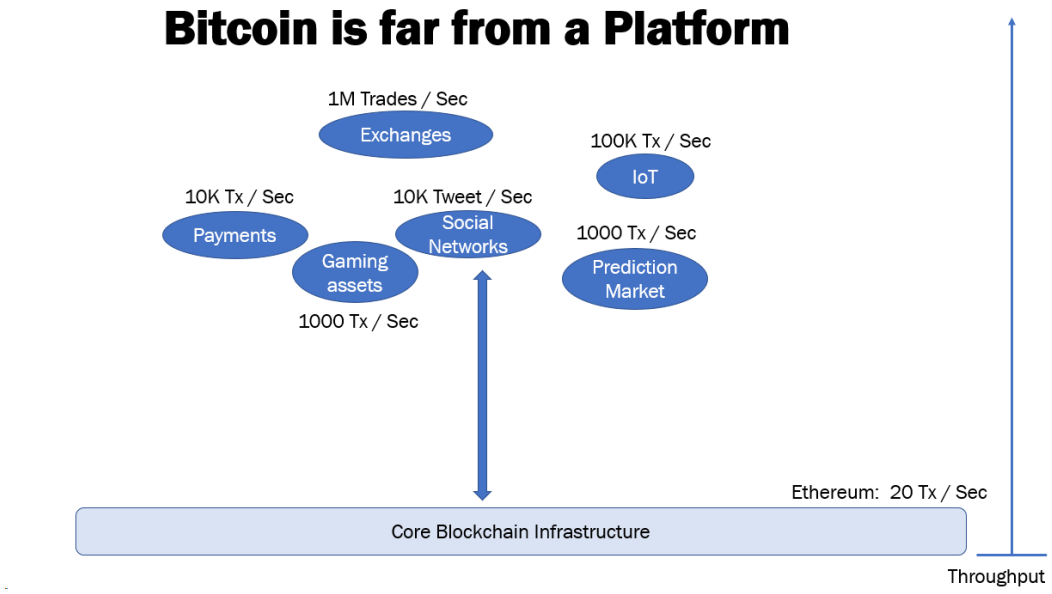

Figure 1.4: Evolution of Trust over human history

Technical components

- **decentralized computer**:
  we are looking at sort of libraries, basic libraries that we are used to, which includes both networking libraries and cryptographic libraries. It mainly contains the followings:
  - Cryptographic data structures
  - Disk I/O, memory/cache and Database management
  - Operating systems
  - Peer to peer networking
  - Consensus and distributed algorithms
- **Virtual machine**:
  - **Reduced instruction set, incentives :**
    A virtual machine uses a simplified instruction set. It also has a decentralized structure that rewards each participant for their contribution, whether it is storage, computation, or data. This creates incentives at the instruction level, similar to how ants benefit from their collective work.

– **General purpose programming language** :
A programming language can be built on top of this system.

Technical challenges

- **Permissionless**:
One of the main goals of designing a decentralized system is to achieve permissionless participation, where anyone can join and contribute to the network without requiring any authorization or intermediaries. However, this also poses a technical challenge, as the system needs to prevent spam and malicious behavior from adversaries who may try to disrupt or compromise the network.

- **Dynamic availability and safety**:
Another goal is to ensure dynamic availability and safety, where the network can reach consensus and maintain its integrity even with changing and unpredictable participation of nodes. This also requires a robust mechanism to deal with potential attacks from adversaries who may try to manipulate or censor the network.

- **Cryptography**:
Cryptography provides some basic tools to address both of these design goals, such as digital signatures, hash functions, encryption, and zero-knowledge proofs. However, cryptography alone is not enough, as adversaries may also use sophisticated techniques to counter or exploit these tools. As the famous quote goes, "The trouble is, the other side can do magic too, Prime Minister." Therefore, designing a decentralized system requires a careful balance of cryptography, incentives, game theory, and distributed algorithms.

1.3 What this course is about

This course covers the **fundamental design principles of blockchains**. We aim to look at good blockchain designs as a full computer rather than isolated elements (we are not going to look only at cryptography parts or algorithms part, …). Basic background in algorithms, probability system programming skills (C/C++) and maturity with nearly all aspects of computer science are required and will be used.

The lectures are divided into three modules:

1. **Understanding Bitcoin**
The first six lectures cover the complete design of Bitcoin. At the end of this module, you will be able to understand the system as a whole and how it functions as a secure, decentralized payment platform. We will introduce various terms pertaining to Bitcoin and cryptocurrencies (e.g., UTXOs, hash pointers, longest chain rule) and cryptography on a "need to know basis." We will study some attacks on Bitcoin and under what conditions Bitcoin is secure under those attacks.

2. **Scaling Bitcoin**
The second module aims to improve the throughput, latency, and resource usage (energy, compute, storage, and communication needs) of Bitcoin while maintaining the same security levels. The goal is to work within the overall architecture of Bitcoin itself and systematically improve various design elements.

3. **Beyond Bitcoin**
The third module covers alternate blockchain protocols that offer properties that are simply not present in Bitcoin. These include accountability, resistance to 51% adversarial attacks, transaction finality, and privacy (the exact meaning of these terms will be made clear later on). We also see how these properties can be combined with Bitcoin-like blockchains via the notion of a 'finality gadget.' This gives us a blockchain with many desirable properties.

This course emphasizes the implementation of blockchains (in software). Thus, the bulk of the evaluation will be based on two projects. The first project involves implementing a Bitcoin client. This will make use of all the concepts learned in the first module of the course. The second project will involve implementing a finality gadget of your choice on top of Bitcoin (from the third module). Rust will be the programming language that is used.

1.4 What is Rust?

Rust is a compiled language such as C and C++ and build for reliability. In Rust we have runtime and compile time safety together and it helps having memory and thread safety. This is very important in blockchain. In general, when a language or model is agreed upon in the blockchain, changing it is indistinguishable from a security attack. Therefore, when the differential method is done, the blockchain cannot be changed or is very difficult to change. Therefore, unlike other applications in the blockchain, it is not easy to add new code and new changes.
For example, the changes that took place in Ethereum took several years because, in addition to the above, it was necessary for all participants to vote, take a stand and move towards the changes. Therefore, the blockchain program must be well written, and for this reason, rust is used.

Chapter 2

Blockchains as Cryptographic Data Structures

Since now we have got familiar with blockchains but now we discuss blockchains in another narrow point of view. We are going to look at blockchains as a data structure with cryptographic properties. We will see what properties this data structure satisfies and that's the key to a blockchain. Remember the two keywords for blockchains, decentralization, and trust. We should be able to link back these properties to trust, and perhaps decentralization.

We will discuss three basic cryptographic primitives :

1. Cryptographic Hash Functions
2. Hash Accumulators, Merkle trees
3. Digital Signatures

The first two primitives are put together to create blockchain that provides trust. The third primitive signatures allow us to go towards decentralization.

2.1 Hash Functions

A hash function is a function that converts a binary string of arbitrary length to a binary string of fixed length.For any piece of data x, the output of the hash function is denoted by $H(x)$ and is called the hash of x.

Example 2.1.1 (Division hashing)

$$y = x \mod 2^{256}$$

Think of x as a binary sequence. This hash function modulate the input with 2^{256} and the output output is a 256 bit sequence, which is the remainder. dividing by a large number of the remainder is uniform between 0 and $2^{256} - 1$. It's a simple deterministic hash function which can be simply computed.

A **collision** happens when there are two inputs, x and x', which get mapped to the same hash, i.e., $H(x) = H(x')$. To minimize collisions, a hash function must distribute its output uniformly and as spread out as possible over the output. space. Therefor, a good hash function should have following properties:

1. Arbitrary sized inputs
2. Fixed size deterministic output
3. Efficiently computable
4. Minimize collisions

Hash functions are widely used not only in blockchains but also in databases, data mining, and information retrieval. They are beneficial for indexing lots of data, especially the ones that are not already numbers e.g. files, images and audio.

2.2 Cryptographic Hash Function

Cryptographic hash functions are quit the same as hash function but with two extra properties:

1. **adversarially collision resistance**
2. **one way function**

The hash of a file should be unique and hard to replicate. An adversary should not be able to create another file that has the same hash as the original one. The previously provided example (2.1) is not adversarially collision resistant since we can simply add 2^{256} to the input and that will have the same output.
In this example if we pick random inputs and compute the hash values, we'll find a collision with high probability long before examining $2^{256} + 1$ inputs. In fact, if we randomly choose just $2^{130} + 1$ inputs, it turns out there's a 99.8% chance that at least two of them are going to collide. The fact that we can find a collision by only examining roughly the square root of the number of possible outputs results from a phenomenon in probability known as the birthday paradox .

The output space of cryptographic hash functions must be large, or else one could easily find a hash collision by iterating over the output space. Typically, they are 128-bit strings, 256-bit strings, or longer.

The key feature of a cryptographic hash function is that it is easy to compute, but difficult to invert. This means that given a binary string x, it is easy to compute $y = H(x)$, but given an arbitrary y', it is difficult and it would take a really long timeto find any x' for which $H(x') = y'$.

SHA-256

Constructing a cryptographic hash function is not easy (in contrast to regular hash functions). The National Institute of Standards and Technology (NIST) sets a standard for these hash functions. One such function is **SHA-256**, where SHA stands for Secure Hash Algorithm, and 256 is the length of the output.
They are built using the Merkle–Damgård construction, from a one-way compression function.

A hash of a certain value acts as a commitment for that value. One can broadcast the hash of the value instead of the value itself. Due to the collision resistance property, one cannot generate an alternate value that matches the same hash value; one is committed to the original value. Thus, publishing the hash of a value is like writing it on a piece of paper and placing it in a sealed envelope.

2.3 Hash Pointer and chains of block

A hash function can also be used as a pointer to certain values when these are stored in a hash table. Note that a hash pointer is nothing more than a hash; the term alludes to the fact that the hash is being used as a pointer. Hash pointer retrieve information and verify the information has not changed. Hash pointers can also be used to build related data structures. Crucially useful for blockchains. In fact, blockchain itself is a hash pointer-based data structure.

In the context of blockchains, a block is a data type that contains a particular header field, called the **hash pointer**, and some **data**.
Using hash pointer, we build a linked list using hash pointers (See figure 2.1). We're going to call this data structure a block chain . Whereas as in a regular linked list where you have a series of blocks, each block has data as well as a pointer to the previous block in the list, in a block chain the previous block pointer will be replaced with a hash pointer. So each block not only tells us where the value of the previous block was, but it also contains a digest of that value that allows us to verify that the value hasn't changed. We store the head of the list, which is just a regular hash-pointer that points to the most recent data block.

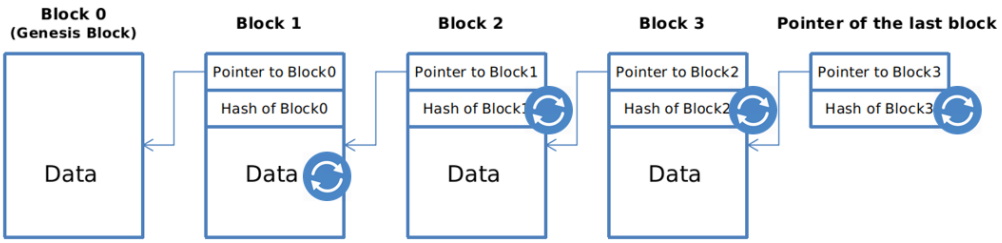

Figure 2.1: A block chain is a linked list that is built with hash pointers instead of pointers

A use case for a block chain is a tamper-evident log . That is, we want to build a log data structure that stores a bunch of data, and allows us to append data onto the end of the log. But if somebody alters data that is earlier in the log, we're going to detect it.

Blockchains are tamper-proof data structures, making them particularly useful in digital trust systems. The system consists of parties that keep their own hash tables as local copies of the data structure. A party can trace the lineage of any block (its parent, grandparent, etc.) using the hash pointers. If a party is missing a block in its hash table, it can ask its peers for the block by using the hash of the block (which it gets from the child block). It can then confirm that the block it gets from its peer is the right one (i.e., it has not been altered) by checking if its hash matches with what it has; this is essentially using the hash as a proof. Similarly, a party can verify if any part of the blockchain it receives has been changed or not.
A party may want to check if a specific data value is part of the blockchain. If the party has the whole blockchain and all its internal data, it can easily prove this. But for a party that only cares about one data value, this is too much work. To make this easier for practical blockchain systems, we use a Merkle tree data structure to store the data in each block. We explain this next.

2.4 Merkle tree

Another useful data structure that we can build using hash pointers is a binary tree. A binary tree with hash pointers is known as a Merkle tree , after its inventor Ralph Merkle. Suppose we have a number of blocks

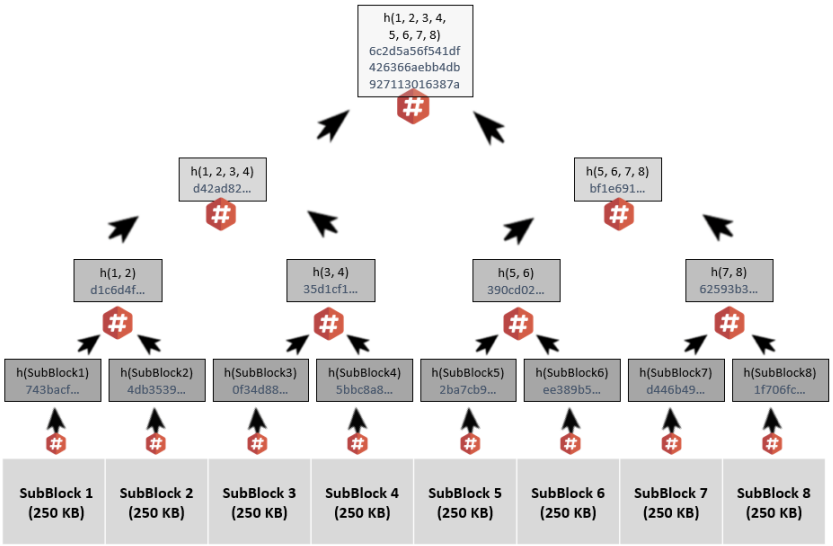

Figure 2.2: A Merkle tree

containing data. We group these data blocks into pairs of two, and then for each pair, we build a data structure that has two hash pointers, one to each of these blocks. We continue doing this until we reach a single block, the root of the tree.

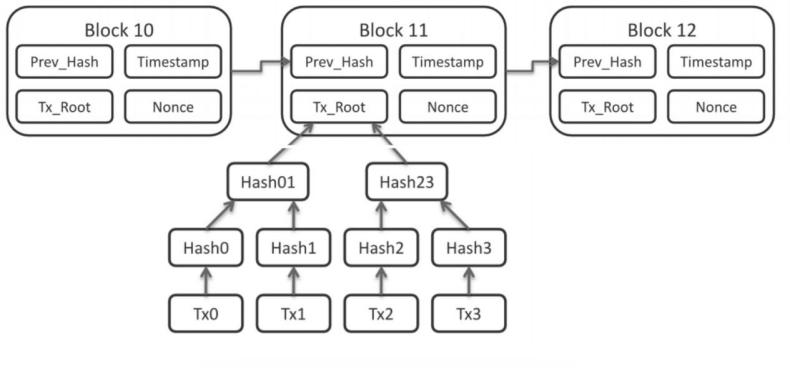

Figure 2.3: Blockchain with markle trees

As you see above, we have a blockchain and data of each block is stored as markle tree. We dont just keep the hash of data, we always keep the hash of markle. At this case header contains the hash of previous block and the root of markle tree.

Proof of membership

Say that someone wants to prove that a certain data block is a member of the Merkle Tree. As usual, we remember just the root. Then they need to show us this data block, and the blocks on the path from the data block to the root. We can ignore the rest of the tree, as the blocks on this path are enough to allow us to verify the hashes all the way up to the root of the tree.

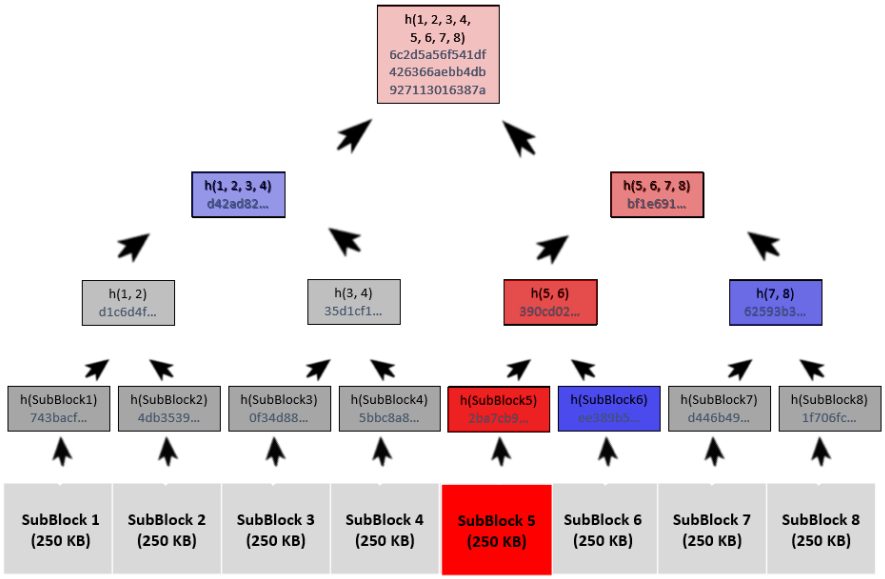

Figure 2.4: Finding a block in Merkle tree

If there are n nodes in the tree, only about log(n) items need to be shown. And since each step just requires computing the hash of the child block, it takes about log(n) time for us to verify it.

Proof of non-membership

. With a sorted Merkle tree, it becomes possible to verify non-membership in a logarithmic time and space. That is, we can prove that a particular block is not in the Merkle tree. And the way we do that is simply by showing a path to the item that's just before where the item in question would be and showing the path to the item that is just after where it would be. If these two items are consecutive in the tree, then this serves as a proof that the item in question is not included. For if it was included, it would need to be between the two items shown, but there is no space between them as they are consecutive.

2.5 Digital Signature

Digital signatures serve as the cryptographic equivalent of handwritten signatures. They enable anyone to verify the sender of a message. In a digital signature scheme, each user is provided with a pair of keys: a secret key known only to the user and a public key shared with everyone. To sign a message, a user employs their secret key, and the resulting signature is sent alongside the message. Another user can verify that the message indeed came from the purported sender by comparing the message and signature to the sender's public key. Thus, the public key becomes the user's identity in the system.

The security of a digital signature scheme lies in the inability of an adversary to forge signatures without knowledge of the corresponding secret key.

It is crucial that each message has a distinct signature. If the same signature could be used for multiple messages, an adversary could simply append the signature to other messages, rendering the scheme ineffective. Typically, users sign the hash of the message they intend to send, ensuring a constant-length bit string for both the signed object and the signature.

To ensure unforgeability, the signature length must be sufficient. Secure signature schemes are standardized by organizations such as NIST.

Bitcoin utilizes the Elliptic Curve Digital Signature Algorithm (ECDSA) as its signature scheme. Elliptic curves are employed in the signing process, but the details are beyond the scope of this discussion. Further information on ECDSA can be found on the Wikipedia page and associated links.

Digital signatures find their first application in upgrading a centralized blockchain system to a decentralized version. Each block in the blockchain is augmented with a digital signature, enabling different users to append blocks and identify themselves through their signatures.

In this decentralized architecture, there are three key questions to address:

1. Who are the eligible users allowed to participate, and how are they selected?
2. When and which block can a user append, and how can others verify this rule in a decentralized manner?
3. Where should a user append a block? In theory, a block can be attached to any other block as viewed by the user.

Blockchain designs vary in how they answer these three questions and the properties they possess. In the next lecture, we will explore how Bitcoin resolves these questions.

The second application of digital signatures involves providing user attribution to the data stored in blocks. While the data is considered an abstract digital entity, there are scenarios where attributing it to users is essential. For example, in cryptocurrencies, the data represents transactions that record the transfer of ownership of coins. By using signatures, user ownership of coins can be established during the creation stage and subsequent ownership transfers can be verified.

2.6 Summary

Hash functions map any value or data to a fixed-size bit string. The hash of a value is typically unique and serves as its identifier. Hash functions play a vital role in constructing tamper-proof data structures like blockchains and Merkle trees.

Hashes provide commitments to data, ensuring its integrity and authenticity. Additionally, Merkle trees serve as accumulators and enable proof of membership for individual elements within a committed set.

Digital signatures function similarly to handwritten signatures, providing authentication in communication. In a blockchain system, all exchanged messages are signed to ensure their authenticity.

We previously introduced ledgers as ordered lists of data values. The blockchain and Merkle tree structures are sufficient to create a centralized ledger, with a central authority responsible for writing to the ledger and multiple parties reading from it.

Digital signatures play a crucial role in transitioning from a centralized ledger to a decentralized one.

To achieve a functional decentralized ledger, three important questions need to be addressed:

1. Who can participate in the ledger, and how are participants chosen?
2. How can users append blocks to the ledger, and how can this process be verified in a decentralized manner?
3. Where should a user append their block? In a decentralized ledger, blocks can be attached to any existing block as perceived by the user.

Chapter 3

Proof of Work and Nakamoto Consensus

Still it is note worthy to remember the two keywords for blockchains, decentralization, and trust. What we have discussed so far was that blockchain data structure enables a tamper-evident and tamper-resistant ledger but only a single party has the privilege to write into the ledger. We saw that this data structure has trust built into it. In this lecture we will go through the idea of decentralization and we will see that how bitcoin deals with decentralization via the Nakamoto consensus protocol.

3.1 Decentralized Blockchain & Nakamoto consensus

We saw the idea of signature on the previous lecture. The identity of the signer is so called the public key, in blockchain or bitcoin terminology is simply called address. In a bitcoin wallet, the address simply reports the public key of the wallet. There is also a secret key corresponding to each public key. Giving away secret key would be tantamount to giving away the access to be able to let other people sign for you .

A coin has a unique identity that is linked to its owner. The owner is the person who received the coin in the last transfer transaction, which can be traced back to the original coin creation transaction or the Genesis block.In blockchains, there are special people who have the authority to create new coins. We will discuss who they are, how they got this power, and what are the principles and rules of coin creation in blockchains. This is related to the field of tokenomics, which studies the design and economics of tokens.

A signature can also be on a block itself. If there are several people who are writing to a database, then it may make sense to know who has signed it and who has entered the data. This data structure, or ledger, can be updated by a fixed and known group of parties who may not fully trust each other. They have a common interest in maintaining the ledger, but they need to follow certain rules to add new blocks to it. This could be consortium.

Unlike some blockchains that limit the number of participants, Bitcoin is open and unlimited. Anyone can create as many identities as they want by generating public and private keys. Bitcoin is a free and decentralized system.

3.1.1 Distributed consensus

When different people can update the ledger, who keeps track of all the changes? One way is to have everyone store the whole ledger, but this is very inefficient and costly. We will see how we can relax this assumption and make the system more scalable.They follow a protocol that lets them check and verify each other's blocks. This is how they ensure the ledger's integrity and security. Consensus or agreement is at the heart of trust.

Byzantine fault tolerance (BFT) is a property of a distributed system that allows it to reach a consensus among its components, even if some of them are faulty or malicious. BFT is important for ensuring the reliability and security of a distributed system, especially in scenarios where there is no central authority or trust among the

participants. BFT is also relevant for blockchain systems, which are distributed networks that store and process transactions without relying on a central authority. Blockchain systems use consensus protocols to ensure that all nodes in the network agree on the same state of the ledger, despite the presence of faulty or malicious nodes. You could use one of these BFT protocols to come up with agreement among the participants on a block.

Bitcoin consensus protocol is so different from BFT protocols in some ways:

- It's decentralized truly in the sense of permissionless.
- It makes less pessimistic network assumptions.

Now there will be a question that **how do people come to consensus?**
Since in case of bitcoin, one can have several identities, voting can not be an answer like it is for democracy. We need a way to have consensus without voting. This was somehow assumed impossible before Nakamoto.

3.1.2 Network Assumptions

People in a consensus need to communicate with each other, in other words wee need a network among the participants. Bitcoin assumptions for this concept are as follows:

- Any node can broadcast to all nodes into the network and it's a fully connected network.
- Every broadcast message reaches every node albeit with some delay which is about 10 minute for bitcoin.

These concepts and ideas will be discussed more on future lectures.

3.1.3 Leader election: Oracle

One way to quickly decentralize is to elect a leader. The leader's job is to basically put the block together and and put it out for everybody to share it. This sharing the block and creating it is called proposal, and this leader is said to be a proposal. Adding a new block to the ledger is a special role that requires some rules and restrictions. Otherwise, there would be too much conflict and confusion among the nodes. They need to agree on the same ledger state. A proposal is simply identified with a public key and not with IP address or network (node).
The ledger is updated in regular intervals, called rounds. Blocks are not created randomly or too frequently (e.g. every second). For example, in Bitcoin, the protocol specifies that a new block is added every 10 minutes. A node creates a new block by collecting all the new and valid transactions that it has seen on the network, and that are not already in previous blocks. This block is the node's proposal for the next ledger state.

A transaction is valid if it meets two criteria:

1. The sender of the coin must have owned the coin in a previous block in the ledger
2. The sender must not have spent the coin twice in different transactions. This ensures that the coin is not duplicated or forged.

In a glance here are the 4 main activities that a proposal do:

1. constitutes a block with transactions
2. validates transactions
3. includes hash pointer to previous block
4. signs the block

A malicious proposer could try to cheat by signing a block when it was not their turn, or by inserting fake or invalid transactions in the block. However, this can be detected and prevented by the other nodes, who can verify the signature and the transactions. The proposer cannot fool the system by being dishonest or lazy. Everyone can verify the validity of transactions by using the ledger, which is a chain of blocks. Each block has a Merkle root that preserves the order of the transactions in the block. The ledger gives a complete history of all transactions ever made, and allows anyone to track the current state of the coins and their owners. This is how validation is done.

There are two kinds of messages that are broadcasted in the network: transactions and proposals. Transactions are sent by anyone who wants to transfer coins, and they are stored in a temporary memory pool by the nodes. Proposals are created by special nodes who have the right to make new blocks, and they contain some transactions from the memory pool. The proposer can choose which transactions to include in the block, as long as the block size is one megabyte.

A satoshi is the smallest unit of a bitcoin, equivalent to 0.00000001 BTC. There are 100 million satoshi in one bitcoin.

3.2 Proof of Work

the method to elect a proposer that Bitcoin shows here in general is called proof of work. The goal of the competition is to select one of the participants to be the proposer. The competition proceeds like this. Proof of work requires miners to solve complex mathematical problems using their computing power, which consumes a lot of energy. The problems are hard to solve but easy to verify, and the probability of finding a solution is proportional to the amount of work done. The first miner who finds a valid solution gets to create a new block of transactions and receive a reward in Bitcoin. The new block is then broadcasted to the rest of the network and added to the chain of previous blocks, forming the blockchain.

The mathematical problem is actually a hash puzzle. Hash puzzles are a game in which one tries to find a nonce (an integer) such that

$$H(nonce, data) < T$$

where T is the target difficulty level. If you can find a nonce, that's the proof of what is work here. We include nonce inside the block. Threshold is chosen such that a block is mined successfully on average once in 10 minutes. The process of searching for a nonce that solves the hash puzzle is called mining.

Properties of Proof of Work

1. Random miner selected at each time
2. Independent randomness across time and across miners
3. Probability of successful mining proportional to fraction of total hash power
4. Sybil resistance
5. Spam resistance
6. Tamper proof – even by the proposer!

The chance of winning is proportional to how much hash power this miner brings to the competition. Hash power relates to the number of hashes I can try per second, which is directly proportional to how much GPUs energy i can bring.
Sybil resistance is the ability of a system to prevent or deter malicious actors from creating multiple fake identities or nodes to manipulate or attack the system. For example, in a peer-to-peer network, a sybil attacker could create many fake nodes to isolate, censor, or deceive honest nodes, or to gain more influence or voting power.

3.3 Forks and Longest Chain Protocol

Forks and the longest chain rule are related to how the Bitcoin network reaches a consensus on the state of the ledger, which is a chain of blocks that contains all the transactions ever made.

A fork is a situation where there are two or more competing versions of the ledger, each with a different block at the end. A fork can happen when two miners find a valid block at the same time, or when some nodes do not receive or accept a new block. A fork can also be caused by malicious nodes who try to create fake or invalid blocks.

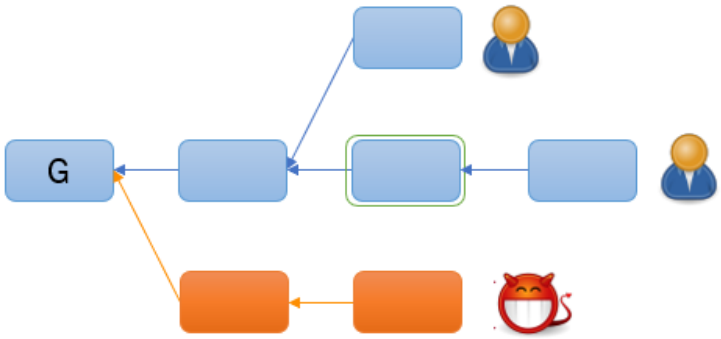

Figure 3.1: Forks in Blockchain

The longest chain rule is a way of resolving forks and choosing the valid version of the ledger. The longest chain rule states that the nodes should always follow and extend the chain of blocks that has the most accumulated proof-of-work, which is a measure of how much energy and effort was spent to create the blocks. The longest chain rule ensures that the majority of the network agrees on the same ledger, and that any forked or orphaned blocks are discarded.

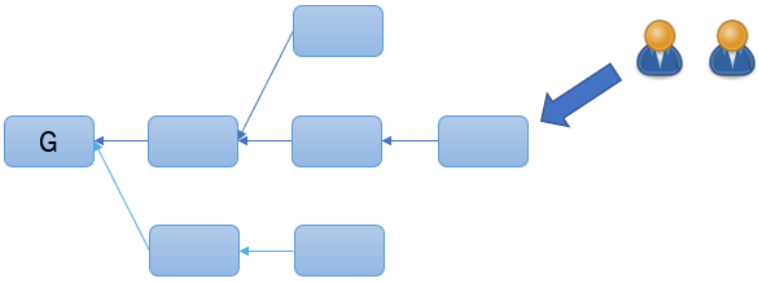

Figure 3.2: The longest chain rule

3.4 Adversarial users

The preceding description outlines the fundamental elements of the Nakamoto consensus protocol, with additional details to be covered in the forthcoming lecture. It is crucial for all users to strictly adhere to the protocol's guidelines. However, in real-world scenarios, there might be some users who deviate from the protocol, and these individuals are known as adversarial users or malicious users.
The primary objective of adversarial parties is to disrupt the system in any way possible. One particular attack they may employ to undermine the append-only property of the ledger is referred to as the **private attack**.
In the private attack scenario, the adversarial users engage in mining new blocks, but they refrain from broadcasting these blocks to the network. This secretive approach allows them to create an alternative chain, or "fork," in the blockchain that remains hidden from honest users. While the adversarial users continue to mine privately, honest users carry on with their mining activities, unaware of the existence of the private blocks.
Due to the inherent randomness in the mining process, the adversarial users might get lucky and quickly build a few private blocks in succession, let's say five blocks. Meanwhile, the honest users, during the same period, mine only three blocks on the public blockchain.

In this situation, the private (adversarial) chain becomes longer than the public (honest) chain because it has five blocks compared to the honest chain's three. Now, the adversarial users release their private chain to all other users, revealing the blocks they kept hidden.
As per the Nakamoto consensus protocol's longest-chain rule, honest users must adopt the chain with the greatest length. Since the private chain is now longer than the public chain, all honest users are compelled to give up their chain and switch to the adversarial chain. As a result, the last three blocks, which were part of the honest chain, are effectively erased from the ledger.
This action of adopting the longer chain effectively invalidates the transactions contained in the last three blocks of the honest chain, allowing the adversarial users to carry out a double-spending attack. By creating a longer chain with more proof-of-work, the adversarial users successfully overwrite the transaction history on the blockchain, causing a disruption and undermining the append-only property of the ledger. This highlights the importance of consensus mechanisms in preventing such attacks and maintaining the integrity and security of decentralized ledgers.

Figure 1.3 illustrates the fork in the blockchain resulting from a private attack:

1. **Scenario:**
   A group of malicious users launches a private attack on the blockchain system.

2. **Fork in the Blockchain:**
   The attack causes a fork in the blockchain, resulting in two competing chains of blocks.

3. **Block Representation:**

   - Dashed blocks: These represent the privately held blocks generated by the adversarial users.
   - Solid blocks: These represent the publicly visible blocks mined by honest users and part of the main blockchain.

4. **Private vs. Public Chain:**

   - Private Chain: Comprises the dashed blocks, which remain hidden from the public network.
   - Public Chain: Consists of the solid blocks and is the main blockchain known to honest users.

5. **Objective:**
   The adversarial users aim to create a longer private chain compared to the public chain by exploiting the longest-chain rule in the Nakamoto consensus protocol.

6. **Revealing the Private Chain:**
   Once the malicious users perceive that their private chain is longer, they reveal it to the entire network.

7. **Switching to the Longer Chain:**
   Following the longest-chain rule, honest users must adopt the longer chain, abandoning the public chain they were initially working on.

8. **Impact on the Ledger:**
   The switch to the adversarial chain effectively invalidates the transactions contained in the blocks of the public chain that are no longer part of the main blockchain.

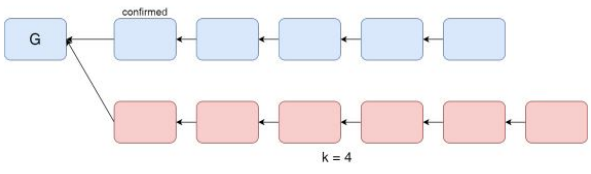

Figure 3.3: An Illustration of the private attack

This private attack undermines the integrity of the blockchain and demonstrates the significance of robust consensus mechanisms in preventing such disruptions and maintaining the security of decentralized ledgers.

The successful execution of private attacks can severely undermine trust in the blockchain system. One particularly impactful demonstration of this is the double-spend action enabled by the private attack. Let's explore this scenario using Figure 1.4:

1. **The Initial Transaction:** A transaction (tx) is embedded in a block B. For instance, let's assume the transaction involves the payment of seven Bitcoins to Tesla, which has recently announced plans to accept Bitcoins for their cars.
2. **Confirmation of the Transaction:** Upon seeing the transaction (tx) embedded in a block on the longest chain, Tesla confirms the transaction and releases the car to the source of the transaction. This action "confirms" or completes the transaction (tx) based on the longest chain, which Tesla considers valid.
3. **Launching the Private Attack:** After the transaction (tx) has been confirmed, an adversary observing this development decides to launch a private attack. The adversary releases two or more blocks it had been secretly mining in private.
4. **Impact of the Private Attack:** As a result of the private attack, the block B containing the confirmed transaction (tx) is no longer part of the longest chain. Consequently, the transaction (tx) is no longer included in the ledger maintained by the network's participants.
5. **Double-Spending Opportunity:** Now that the transaction (tx) has been removed from the ledger, the source of the transaction is free to double-spend the seven Bitcoins. This means they can use the same seven Bitcoins to initiate another transaction, effectively spending them twice.
6. **Trust Demolished:** The successful double-spending attack seriously undermines the trust embodied by the blockchain system. It compromises the integrity of the ledger and erodes confidence in the security of the network.

One way to address this vulnerability is by delaying the confirmation of transactions. By waiting for more blocks to be added to the chain after the transaction is initially embedded, the likelihood of a successful double-spending attack through a private attack is reduced. This delay allows the network to have more confidence in the validity of transactions before confirming them, enhancing the security and trustworthiness of the blockchain system.

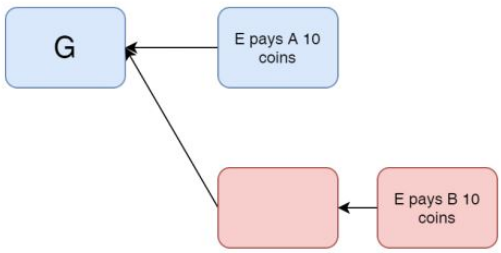

Figure 3.4: An Illustration of the double spend attack.

3.5 The k-deep confrmation rule

In the Nakamoto consensus protocol, occasional changes at the end of the ledger is impossible to avoid. This chancges can be made by adversarial or by network delays.
Malicious users can collaborate and launch private attacks, as discussed earlier. By creating a longer chain in secret and releasing it to the public network, they can rewrite the last block of the ledger. This undermines the integrity of the ledger and poses a significant challenge to maintaining trust in the blockchain system.

Even without adversarial users, forks can occur in the blockchain due to network delays. When multiple blocks are mined simultaneously, they can propagate through the network at different speeds, leading to temporary forks. Honest users may work on different branches of the blockchain, resulting in competing chains.
Forks introduce the possibility of the longest-chain switching. As honest users continue mining and adding blocks to their respective branches, one of the branches might eventually become longer, leading to a shift in the longest-chain rule. When this happens, the blockchain system must adapt to the new longest chain, resulting in a rewriting of the last portion of the ledger.
Indeed, the solution to address the issue of occasional changes at the end of the ledger in the Nakamoto consensus protocol is simple and intuitive. By treating a block as confirmed only if it is buried deep enough under other blocks, the blockchain system can enhance its resilience against forks and ensure the immutability of the ledger. Here's a breakdown of the solution:

1. **Confirmation Depth:**
   Users should consider a block as confirmed only if it is buried deep enough under a certain number of subsequent blocks. Let's denote this number as "k."

2. **Consequences of Confirmation:**
   Confirming a block, say block B, means that the portion of the ledger up to that block is now considered immutable and secure. This implies that all transactions contained in the blocks preceding B are considered final and trustworthy.

3. **Delayed Confirmation:**
   The solution suggests that it is not advisable to confirm very recent blocks immediately. New blocks are vulnerable to being overwritten due to forks in the blockchain. By delaying the confirmation of recent blocks, the system allows more time for the network to converge to a single valid chain.

4. **Protection Against Forks:**
   By requiring a minimum depth of k blocks to confirm a specific block, the system provides protection against forks. Blocks that are not part of the longest chain, and therefore part of competing forks, will not be confirmed until a sufficient number of additional blocks are added to the chain, making it less likely for the confirmation to switch between forks.

5. **Increasing Security Over Time:**
   As more blocks are added to the blockchain and the depth of confirmations increases, the security and finality of transactions improve. Deeper confirmations provide greater confidence that the ledger's history is unchangeable.

This approach aligns with the practical behavior observed in many blockchain systems, where users and applications typically wait for a certain number of block confirmations before considering a transaction as fully settled. The higher the confirmation depth (larger k value), the more secure and reliable the transaction becomes.
By implementing this simple confirmation mechanism, blockchain systems can maintain the integrity of the ledger and safeguard against the potential disruptions caused by forks and private attacks, contributing to a more robust and trustworthy decentralized network.
The k-deep confirmation rule, as described earlier, is based on the belief that any changes in the longest chain will typically occur towards the end of the chain. The prefix of the longest chain at a specific time t, obtained by dropping the last k blocks, will continue to remain a prefix of the longest chain at future times as well. It is essential to note that this belief holds for most practical scenarios, but it is not guaranteed with absolute certainty for any finite value of k.
Certain conditions, such as adversarial users investing in significant computing power, disrupting block communication among honest users, or having an extraordinary stroke of luck, can potentially overturn a deeply buried block. However, as the value of k increases, the likelihood of such events happening diminishes significantly.
Lecture 6 demonstrates that the probability of such events decreases exponentially with k, under the assumption that the honest miners control a majority of the total hash power (i.e., more than 50%). This assumption is critical in the Nakamoto consensus protocol, as it ensures that the honest miners collectively have more computational power than adversarial users, making it more improbable for adversarial users to consistently outpace the honest chain in mining blocks.
In summary, the k-deep confirmation rule, although not entirely foolproof, provides a practical and effective approach to enhance the security and reliability of the blockchain system. By requiring a certain depth of confirmations (larger k value) before considering a block as truly confirmed, the system minimizes the risk of chain

reorganizations due to forks and private attacks. The larger the value of k, the lower the likelihood of such disruptive events happening, reinforcing the blockchain's robustness when honest miners have a majority of the hash power.

3.6 Variable mining difculty

In PoW blockchains, adapting to the vast variation in mining power is essential for maintaining stability and security. One way to achieve this is through a difficulty adjustment algorithm. In the case of Bitcoin, the following three core ideas are used in its difficulty adjustment algorithm:
(a) Varying the Difficulty Target: The difficulty target for block mining is adjusted based on the average inter-block time from the previous epoch, which consists of 2016 blocks. If the average time is greater than the desired inter-block time (e.g., 10 minutes), the difficulty is reduced to make mining easier. Conversely, if the average time is less than the target, the difficulty is increased to make mining harder.
(b) Using the Heaviest Chain Rule: Instead of considering the longest chain, the blockchain system determines the valid chain based on the total sum of block difficulties. The chain with the most accumulated proof-of-work (highest difficulty) is considered the heaviest chain and is used to determine the valid ledger.
(c) Mild Difficulty Adjustment: The difficulty is adjusted only mildly at the end of each epoch, and the adjustment is bounded by a factor of 4. This prevents abrupt and drastic changes in the difficulty, ensuring a smoother transition during different mining periods.
While this algorithm may seem straightforward, simpler approaches can lead to dangerous security vulnerabilities. For example, using only the heaviest chain rule (b) and allowing miners to choose their own difficulty can create significant problems.
An initial analysis might suggest that the heaviest chain rule does not provide an advantage for manipulating difficulty. However, this lack of advantage only holds in expectation, and extremely difficult adversarial blocks can introduce high variance, thwarting confirmation rules that confirm deeply-embedded blocks with non-negligible probabilities proportional to the attacker's mining power[1]..
For instance, suppose honest miners adopt the initial mining difficulty defined in the genesis block, with an expected inter-block time of 10 minutes (using 10 minutes as the unit of time). The difficulty unit is defined as the time it takes to mine a honest chain with k blocks. If the adversarial mining power is half of the honest mining power (or 1/3 of total mining power), the adversary can mine a single block as difficult as k honest blocks within k units of time. The adversarial mining process follows a Poisson point process with rate 1/2k, and the number of adversarial blocks mined in k units of time follows a Poisson distribution Poiss(1/2).
Hence the success probability of this attack would be:

$$P(\text{attack succeeds}) = P(\text{Poiss}(1/2) \geqslant 1) = 1 - e^{-1/2} \approx 39.3\%$$

which is a constant independent of k, therefore any k-deep confrmation rule will fail.
Consider a more detailed difficulty adjustment rule involving only (a) and (b), where the adversary can perform a difficulty-raising attack. The attack exploits the ability to create an epoch with extremely close-together timestamps, causing the difficulty adjustment rule to set the difficulty significantly higher for the next epoch. The attacker then utilizes the high variance in mining to execute the attack. Here's a description of this attack:

1. **Adversarial Timestamps:**
   The adversary can manipulate timestamps in their private blocks, allowing them to create an epoch with timestamps that are extremely close together.

2. **Difficulty Setting:**
   The difficulty adjustment rule (a) calculates the difficulty for the next epoch based on the average inter-block time from the previous epoch. If the timestamps in the adversary's private blocks indicate an extremely fast block generation rate, the difficulty for the next epoch will be set very high.

3. **Difficulty-Raising Attack:**
   Let B be the first block of the second epoch in the adversary's private chain, with difficulty X. The chain difficulty of block B is 2016 + X, as it includes the difficulty of the previous 2016 blocks.

4. **Honest Chain Mining Time:**
   Mining an honest chain with chain difficulty 2016 + X, on average, takes 2016 + X units of time.

5. **Attack Preparation Time:**
   Considering the same adversary, it takes, on average, 4032 units of time for them to complete the first epoch in their private chain.

6. **Attack Execution:**
   To succeed in the attack, the adversary needs to mine block B within X − 2016 units of time, which happens with a probability:

$$P(\text{attack succeeds}) = P(\text{Poiss}\left(\frac{X-2016}{2X}\right) \geqslant 1) = 1 - e^{-(X-2016)/2X} \approx 1 - e^{-1/2} \approx 39.3\%$$

The attack's success probability is approximately 39.3%. If the adversary can accomplish this, they can create a difficulty spike for the next epoch, making it exceedingly difficult for honest miners to keep up with the fast mining rate. This high variance in mining can then be exploited by the attacker, allowing them to perform reorganizations, invalidate deeply-embedded blocks, and potentially double-spend.
This difficulty-raising attack highlights the importance of a careful and sophisticated difficulty adjustment algorithm in PoW blockchains. The Bitcoin difficulty adjustment algorithm takes into account a combination of factors, such as average inter-block time, to prevent such attacks and maintain a stable and secure block generation rate over time. By carefully adjusting the difficulty at regular intervals and avoiding abrupt changes, the system can mitigate the risks associated with extreme variations in mining power and ensure the resilience of the blockchain network.
Correct, if X 2016, the success probability of the difficulty-raising attack remains significant, and it becomes more challenging to rely solely on a k-deep confirmation rule. This complex attack, which is only thwarted by the full protocol employing (a), (b), and (c) together, highlights the importance of the comprehensive Bitcoin difficulty adjustment algorithm. Let's formally describe the Bitcoin difficulty adjustment algorithm:
Consider a chain of v blocks with timestamps $(r_1, \ldots, r_v)$. The Bitcoin difficulty adjustment algorithm uses the following fixed parameters:

- τ (set to 4 in Bitcoin): The factor by which the difficulty is allowed to be adjusted mildly every epoch.
- Φ (set to 2016 in Bitcoin): The length of an epoch in the number of blocks.
- Λ_0 (set to 2 weeks in Bitcoin): The expected duration of an epoch.

The difficulty adjustment algorithm aims to keep the average inter-block time close to the target block time (e.g., 10 minutes for Bitcoin) over the long term.

The algorithm operates as follows:

1. Calculate the Time Difference:

$$D = r_v - r_1$$

2. Calculate the Expected Epoch Duration:

$$\Lambda = \frac{\Lambda_0 \cdot D}{\Phi \cdot \tau}$$

3. Adjust the Difficulty:

$$\text{If } D < \frac{\Lambda}{2}, \text{ increase the difficulty threshold by a factor of } \tau$$
$$\text{If } D > 2\Lambda, \text{ decrease the difficulty threshold by a factor of } \tau$$

4. Keep the Difficulty within Bounds:

   Ensure that the difficulty adjustment is limited to a maximum increase or decrease of a factor of τ

By adjusting the difficulty threshold based on the actual epoch duration and comparing it to the expected epoch duration, the algorithm maintains a relatively stable and predictable inter-block time over the course of a long duration. This prevents abrupt difficulty spikes or drops caused by significant changes in mining power and helps maintain the security and stability of the Bitcoin blockchain.

The full protocol, which employs (a), (b), and (c) together, effectively mitigates various types of attacks and ensures the integrity of the blockchain system, preventing malicious actors from manipulating the difficulty to their advantage.

The target calculation function D : Z$*$ $\rightarrow$ R is defned as

$$D(\epsilon) = T_0$$

where T_0 is the initial target as defined in the genesis block, and Φ', Λ, and T correspond to the last block, duration, and target of the last completed epoch, respectively, i.e., $\Phi' = \Phi \left\lfloor \frac{v}{\Phi} \right\rfloor$, $\Lambda = r_{\Phi'} - r_{\Phi'-\Phi}$, and $T = D(r_1, \ldots, r_{\Phi'-1})$. A full and beautiful analysis of the Bitcoin protocol is provided in [2].

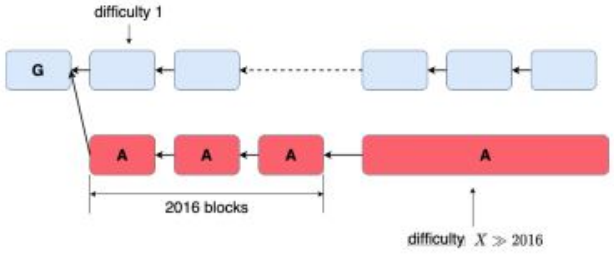

Figure 3.5: The difculty rising attack. The adversary raises the difculty to extremely high in the second epoch by faking timestamps.

3.7 Bitcoin is Permissionless

In the context of Bitcoin, participants have the freedom to generate multiple (secret key, public key) pairs for themselves, allowing them to create multiple identities. This feature is encouraged for privacy reasons, making Bitcoin a permissionless system. However, creating multiple identities does not grant an individual more influence in the system. The actual representation and power in Bitcoin are determined by the computational (mining) power, which can only be increased through a capital investment in hardware and resources. This property ensures that the system is resistant to Sybil attacks, where an adversary could gain an advantage by creating numerous fake identities.

The PoW mining process in Bitcoin serves multiple important purposes simultaneously:

(a) Sybil-Resistance: By tying mining power to computational resources and capital investment, Bitcoin prevents individuals from easily gaining control of the system through the creation of multiple identities.

(b) Randomized Block Proposer Election: PoW mining introduces an element of randomness to the selection of the miner who gets to propose the next block. The probability of being chosen as the block proposer is proportional to the miner's computational power. This process ensures that block proposals are distributed across different miners, enhancing the decentralization of the network.

(c) Adjusting Inter-Block Duration: As discussed earlier, the PoW difficulty adjustment algorithm regulates the inter-block time, maintaining a relatively stable and predictable block generation rate.

In contrast to Bitcoin's permissionless nature, some other blockchain designs implement external mechanisms to ensure that each entity has only a single key, creating a permissioned system. In such permissioned blockchains, separate mechanisms are used for achieving goals (b) and (c) mentioned above, while still maintaining Sybil-resistance through other means.

In subsequent lectures, we will explore some of these other blockchain designs and understand how they address different aspects of consensus, security, and decentralization. Each design may have its trade-offs and may be better suited for specific use cases depending on the requirements of the application.

References

[1] Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013, 2013*

[2] Garay, J., Kiayias, A., & Leonardos, N. (2020). Full analysis of Nakamoto consensus in bounded-delay networks. Cryptology ePrint Archive, Report 2020/277. https://eprint.iacr.org/2020/277

Chapter 4

Peer to Peer Network and Bitcoin system

4.1 Blockchains and Networking

A blockchain network is a system that provides ledger and smart contract services to applications without relying on a centralized server or authority. A centralized server can be a single point of failure or a target for censorship, which can compromise the security and availability of the network. Therefore, a blockchain network has the following networking requirements:

- The key primitive of the network is to **broadcast blocks and transactions to all nodes**.
  Blocks are data structures that contain transactions, which are records of value transfers or contract executions. Transactions are validated and appended to the ledger by consensus algorithms. Broadcasting ensures that all nodes have the same view of the ledger and can verify its integrity.
- The network also needs to be **robust and resilient to node failures or attacks**.
  Some nodes may go offline due to various reasons, such as power outages, network disruptions, or malicious attacks. The network should be able to tolerate a certain percentage of node failures without affecting its functionality. Moreover, the network should allow new nodes to join and synchronize with the existing nodes, as well as handle node churns and partitions.

4.1.1 Types of Network Architecture

There are two types of network architecture:

1. **Client Server** :
   This is a type of network architecture where one or more servers store most of the data and resources, and the clients access them through requests. The server is the central authority that controls the network, and the clients are dependent on the server for their functionality.
2. **Peer to Peer** :
   This is a type of network architecture where each node acts as both a client and a server, and there is no central authority or hierarchy. The nodes share their data and resources directly with each other, without relying on a server.

The need for robustness implies that we do not want a client-server relationship; we settle for a peer-to-peer (P2P) network where each node has identical behavior.

4.1.2 Overlay Networks

An overlay network depicts the connections between nodes, and is represented as a graph. It abstracts out the physical network switches and routers and defines virtual links between nodes. Two nodes that are connected by a link can exchange messages directly. Those that are not connected by a link must find a path connecting them on this overlay network in order to communicate messages.

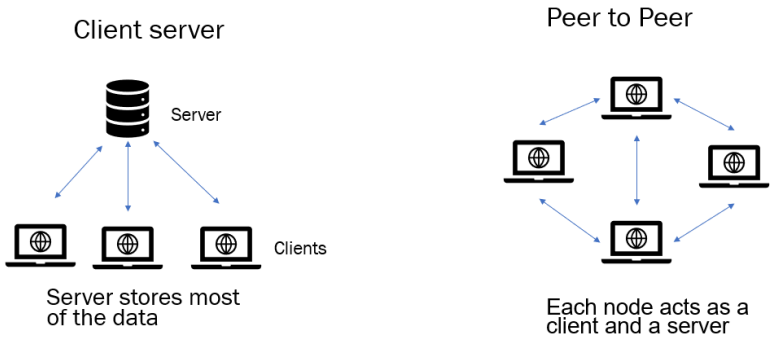

Figure 4.1: Types of Network Architecture

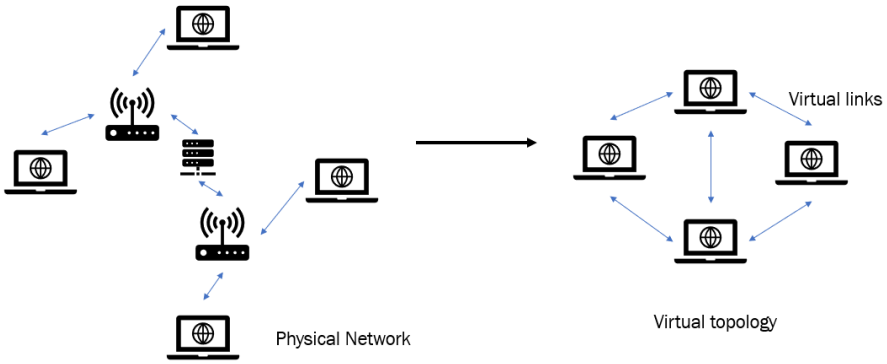

Figure 4.2: Overlay Network

There are different ways to classify overlay networks, but one common way is to distinguish between two types of overlay networks

- **structured** : These are overlay networks that have a specific topology or organization, such as a ring, a tree, or a grid. Structured overlay networks, like CHORD, assign an identifier to each node and uses that to construct welldefined routing rules. These networks are excellent for routing sending point to point messages, a use case not required for Bitcoin. Structured networks are suitable for broadcast, too; any message transmission takes O(logN) hops on CHORD with O(logN) connections per node.
- **unstructured** : These are overlay networks that have no specific topology or organization, and the nodes are connected randomly or based on some criteria, such as proximity, interest, or availability. Unstructured networks like d-regular graphs have no node identifiers; a node connects to d other nodes randomly.

Routing point to point messages is not feasible because it takes O(logN) hops and requires many peer queries to find the path from point A to point B. However, broadcast is very effective using gossip and also takes O(logN) hops. Therefore, the number of hops needed is equal to the structured network with the extra advantage of O(1) peer connections. That is why the Bitcoin network uses an unstructured d-regular overlay network.

4.1.3 Gossip and Flooding

Gossip and Flooding are two techniques for disseminating information in a network of nodes. Gossip is a technique that randomly sends the information to some of its neighbors, while Flooding sends the information to all of its neighbors. Both techniques can spread the information exponentially and reach all nodes in O(logN) time.
Gossip and Flooding can be used to achieve fast and reliable information dissemination in a network, but they differ in terms of efficiency, overhead, and robustness.

4.1.4 Expander Graph

Expander graph is a type of graph that has the property of being well connected but sparse.

Definition 4.1.1: Sparse Graph

A sparse graph $G(V, E)$ is a graph that has a small number of edges compared to the number of vertices, such that $|E| = O(|V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices.

An Expander graph is a sparse graph that is also well connected, such that for any subset A of vertices, the number of vertices outside A that have at least one neighbor in A, denoted by $|A|$, is large.

A gossip message starts with $A(0)$ as the broadcasting node with $|A(0)| = 1$. In the next hop, it will reach $\partial A(0)$ with $|A(1)|$ being at least $(1+\epsilon)$ times $|A(0)|$. This process repeats and we get $|A(k)|$ being at least $(1+\epsilon)^k$ times $|A(0)|$ for any k. Therefore, the number of steps to get to half the number of nodes is proportional to the logarithm of the number of nodes. It can be proven that the other half of the nodes can also be reached in O(logN) time.

4.2 Bitcoin Network

Bitcoin is a peer-to-peer network that uses TCP protocol to exchange data. In Bitcoin network:

- The codebase limits the number of outgoing connections to eight and the number of incoming connections to 117.
- The network has a high churn rate (rate at which users join or leave the system); therefore, the node must be prepared to connect to new peers.
- The node maintains a large list of nodes running Bitcoin in the form of their (IP, port) pair and connects to one of them randomly when a slot becomes available.
- The node ensures that the peers it connects to are selected randomly.

4.2.1 Peer discovery

A node starts its list of peers by connecting to a group of DNS seed nodes. These are special nodes that provide a list of IP addresses and ports of active nodes in the network. A node can query a DNS seed node to get this list and then try to connect to some of the nodes in the list.
The seed nodes are not very distributed; so it is not wise to depend entirely on the peer list given by them. After connecting to the first set of peers, a node requests their peer list using **getAddr** and **Addr** messages. The node updates its peer list frequently by swapping peer lists with its peers.

4.2.2 Block transmission

The process of transmitting blocks and transactions involves the following steps:

1. A node sends an **inv** message to its peers, which is an inventory message that tells them what blocks and transactions are available.
2. When a peer receives an **inv** message, it checks if it already has the block or the transaction in its local storage. If not, it sends a **getData** message to the node to get those blocks and transactions.
3. Optionally, the peer can request only the headers of the blocks first, using a **getHeaders** message, before asking for the full blocks .
4. This header-first block transmission can reduce the bandwidth use and prevent invalid blocks from being accepted .

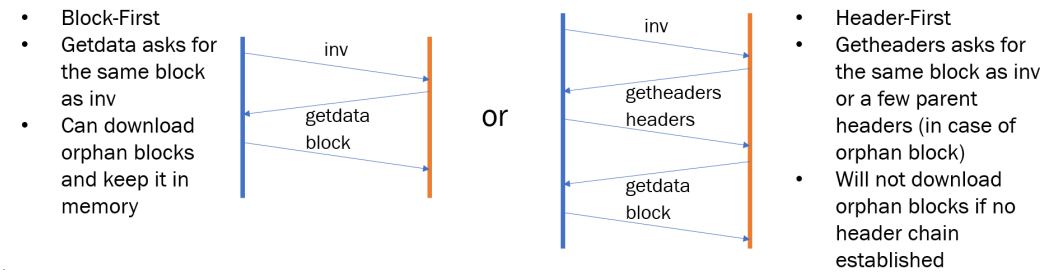

Figure 4.3: Block transmission in blockchain network

4.2.3 Data Broadcast

Now we explain how data, such as transactions, are broadcasted in a peer-to-peer network

- Each node maintains a non-persistent memory to store unconfirmed transactions, which are transactions that have not been included in a block yet. This memory is called the mempool.
- When a node receives a new transaction, it sends an inv message to its peers, which contains the transaction id (txid). The inv message is a way of checking if the peer already has the transaction in its mempool.
- If not, the peer sends a getdata message to request the full transaction data from the node. The node then sends the transaction data (tx) to the peer.
- Some unconfirmed transactions might be removed from the mempool due to various reasons, such as low fees, expiration time, or double spending.

4.2.4 Compact Blocks

Here we discuss two methods of relaying blocks in a peer-to-peer network:

1. **legacy relaying** :
   Legacy relaying is the traditional way of sending the full block data to each node.
2. **compact block relaying** :
   Compact block relaying is a protocol that reduces the amount of bandwidth and latency required to transfer a block that confirms many transactions that the nodes already have in their mempools.

There some differences between the two methods in terms of the messages exchanged between the nodes .Here are the advantages of compact block relaying over legacy relaying:

- Compact block relaying uses shortened transaction identifiers (txids) instead of full transaction data, which reduces the size of the block data.
- Compact block relaying allows nodes to request only the transactions they are missing from their mempools, which reduces the redundancy of sending transactions twice.
- Compact block relaying has two modes: low bandwidth mode and high bandwidth mode.
  Low bandwidth mode minimizes the bandwidth usage by asking for permission before sending a compact block.
  High bandwidth mode reduces the latency by sending a compact block as soon as possible without asking for permission.
- Compact block relaying can verify the proof of work (PoW) from the block header before requesting the full block data, which prevents invalid blocks from being accepted.

There is a trade-off between capacity and propagation delay in a peer-to-peer network. **Capacity** (C) is the maximum amount of data that can be transmitted in a given time, measured in bits per second (bps). **Propagation delay** (D) is the time it takes for a signal to travel from one node to another, measured in seconds. The end to end delay, which is the total time it takes for a block to reach all nodes in the network, increases with the increase

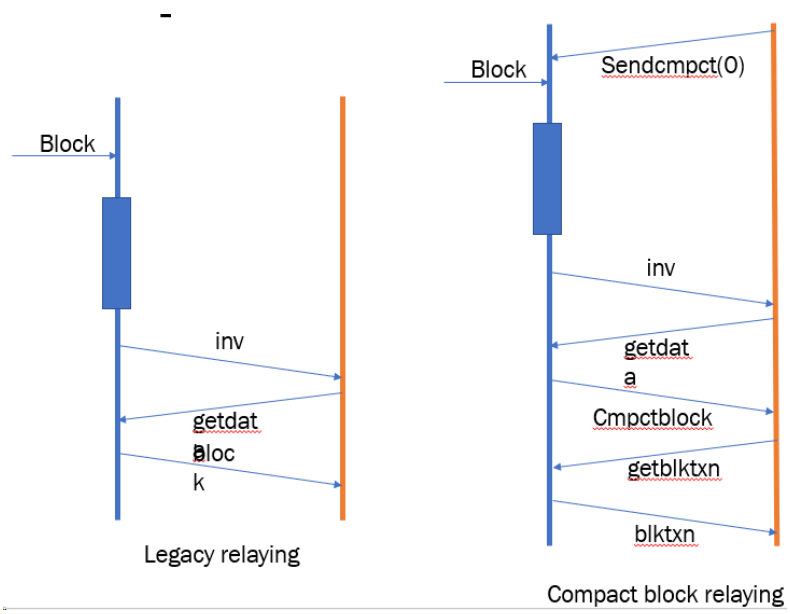

Figure 4.4: Compact Block

in block size. This is because larger blocks take longer to transmit and verify, which can cause congestion and orphaning. Therefore, there is a trade-off between increasing the capacity and reducing the propagation delay of the network. This increase in delay affect the system in following ways:

- Wasted Hash power
- Forking

4.2.5 Disadvantages of current p2p network

- **Efficiency:**
  It requires a lot of communication between nodes. The total communication is proportional to the number of nodes (N) and the average degree of connectivity (d). This means that as the network grows, the communication overhead also increases, which can affect the performance and scalability of the system.
- **Privacy:**
  It can link the transaction source to the IP address of the node that broadcasts it. This means that anyone who monitors the network traffic can potentially identify the sender and receiver of a transaction, which can compromise their anonymity and privacy.
- **Security:**
  It allows for plausible deniability for forking. Forking can cause inconsistency and double spending. Plausible deniability means that a node can claim that it did not receive a valid block from another node, even if it did, and create a fork on purpose. This can be used as an attack strategy to disrupt the consensus and integrity of the system.

4.3 Geometric Random Network

The current network topology is based on random IP addresses, which are not related to geographic distances. This means that nodes may be connected to peers that are far away from them, which can increase the latency and bandwidth usage of the network.

A better network topology would be based on a **geometric random network**, which is a network where nodes are placed according to some geometric criteria, such as proximity or similarity. This way, nodes can connect to peers that are closer or more relevant to them, which can improve the efficiency and performance of the

network. The challenges are creating a self-adapting network topology based on measurements, such as latency, bandwidth, or trust. This means that nodes can dynamically adjust their connections based on the changing conditions and preferences of the network.

4.3.1 Perigee

Perigee is a self-adaptive network topology algorithm for peer-to-peer networks. Perigee is a decentralized algorithm that selects neighbors based on past interactions. It aims to retain neighbors that relay blocks fast and disconnect from neighbors that do not relay blocks fast. It also explores unseen neighbors to discover new potential connections.

'Perigee is motivated by the multi-armed bandit problem, which is a problem of finding the optimal strategy for choosing among several options with uncertain rewards. Perigee tries to balance between exploration and exploitation, which means finding new neighbors and using existing neighbors, respectively. Perigee can improve the efficiency and performance of the peer-to-peer network by reducing the latency and bandwidth usage.

Algorithm

The Perigee Algorithm works as follows:

- It assigns scores for each subset of neighbors based on how fast they relay blocks.
- It retains the subset of neighbors with the best score and disconnects the node that is not in the subset.
- It forms a connection to a random neighbor to explore new potential connections.

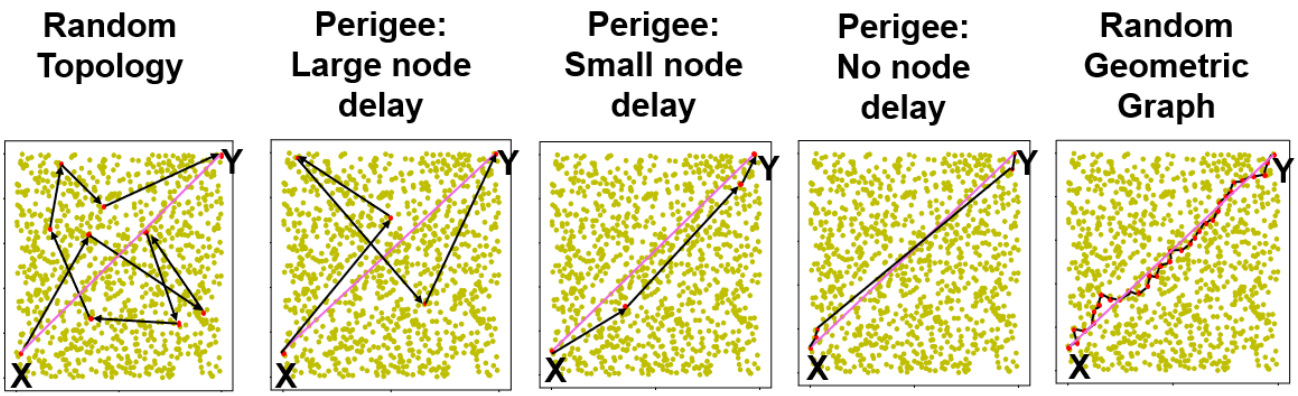

Figure 4.5: Performance of Perigee

4.4 Efficient Networking

Efficient networking is the ability to transmit and receive data in a fast, reliable, and scalable way. There are three aspects of efficient networking:

1. **Trusted networks** :
   Trusted networks are networks where nodes can trust each other to relay valid and timely data. Trusted networks can improve the efficiency of the network by reducing the overhead and redundancy of data transmission. However, trusted networks also pose some risks, such as centralization and censorship.

2. **Privacy** :
   Privacy is the ability to protect the identity and activity of nodes from external observers. Privacy can improve the efficiency of the network by reducing the exposure and vulnerability of nodes. However, privacy

also poses some challenges, such as linking transaction source to IP address and plausible deniability for forking.

3. **Security** :
   Security is the ability to prevent and resist attacks from malicious nodes or adversaries. Security can improve the efficiency of the network by maintaining the integrity and consistency of the data. However, security also poses some difficulties, such as eclipse attacks and plausible deniability for forking.

4.4.1 FRN (Fast relay network)

FRN (Fast relay network) is an example of a trusted network model, which is a hub and spoke model where miners connect to FRN servers. FRN servers are trusted servers that are fast and efficient in relaying blocks and transactions. FRN servers can reduce the latency and bandwidth usage of the network by filtering and compressing the data. However, FRN servers also pose some risks, such as centralization, censorship, and single point of failure.

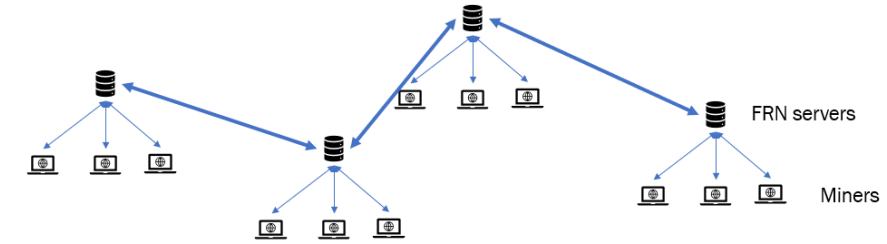

Figure 4.6: FRN

4.4.2 Trusted Networks: Falcon

Falcon network routing is based on the idea of cut through routing, which is a method of forwarding data packets as soon as their headers are verified, without waiting for the whole packet to arrive. This can reduce the latency and bandwidth usage of the network, as well as the risk of forking and double spending. Falcon network routing uses a set of trusted servers, called FRN servers, to relay data packets between nodes.

Cut through routing is the way of sending data packets from point a to point b, where each node verifies the header of the packet and forwards it to the next node without waiting for the whole packet to arrive. The diagram shows that cut through routing has lower latency than legacy routing, which is represented by the distance between the green lines and the red lines in the diagram. The diagram also shows that cut through routing uses FRN servers to relay data packets between nodes.

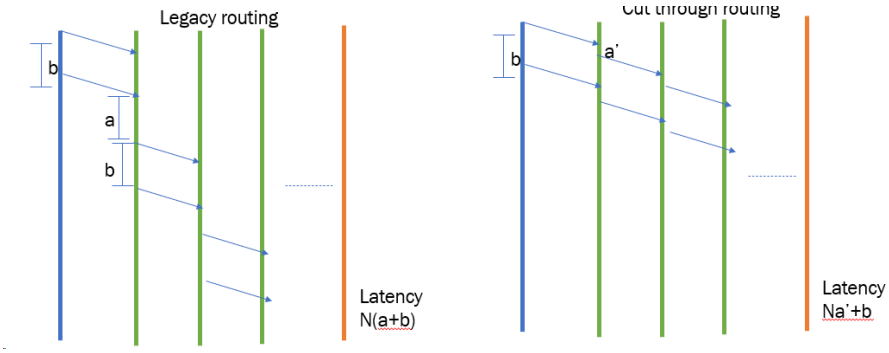

Figure 4.7: Legacy routing vs. Cut through routing

4.5 Network Anonymity and Privacy

Blockchains are systems where the data structure of the blockchain is shared and visible to all participants. For example, in a cryptocurrency, the blockchain records every transaction that ever happened in the currency. However, this also means that the blockchain can contain sensitive data of users, such as their identities and balances. This can raise privacy concerns for users who do not want their data to be public. Therefore, privacy is an important issue in blockchains. How can users achieve privacy in blockchains?

Bitcoin and most other cryptocurrencies use pseudonymous identifiers to protect the privacy of users. Each user has one or more pseudonym, which is a public key that they use to participate in the system. The blockchain only shows the transaction patterns of each pseudonym, not the real identity of the user. However, this system is not very secure, because there are ways to link the pseudonyms to the user's real identity, especially if there is extra information available. These are called de-anonymization attacks, and they can expose the user's sensitive data.

4.5.1 Network De-anonymization Problem

he network de-anonymization problem is the challenge of finding the real identities of users who participate in a network that uses anonymization techniques. Anonymization techniques are methods that hide the personal information of users, such as their names and locations, by using random or fake identifiers. Anonymization techniques are supposed to protect the privacy of users, but they can be broken by attackers who have access to some extra information or techniques. The network de-anonymization problem can be modeled using a graph, where nodes represent users and edges represent connections between them. The attacker's goal is to guess which node in the anonymized graph corresponds to which user in the original graph. The attacker can use different methods, such as eavesdropper adversaries or botnet adversaries, to collect and analyze data from the network. Eavesdropper adversaries are attackers who listen to all or most of the network traffic and use metadata and topology information to infer the source of a message. Botnet adversaries are attackers who control a set of compromised nodes that participate in the network normally and share information with each other. The network de-anonymization problem is a difficult and important problem for network security and privacy.

There are two main types of attackers in network de-anonymization:

1. **Eavesdropper attackers** :
   Eavesdropper attackers are attackers who connect to all or most of the nodes in the network and listen to their communications. They look like normal nodes to the honest nodes, who relay messages to them as usual. Eavesdropper attackers can collect metadata, such as timestamps and IP addresses, from the messages they receive. They can also use information about the network topology, such as the connections between nodes, to guess the source of a message. Eavesdropper attackers usually do not send any messages; they only observe and analyze the messages they receive.

2. **Botnet attackers** :
   Botnet attackers are attackers who control a set of compromised nodes that act like normal nodes in the network. They can accept and relay messages from other nodes, but they also share information with each other. Botnet attackers can have limited visibility into the network, depending on how many nodes they control and how they are connected. Botnet attackers can also inject fake or modified messages into the network to confuse or mislead other nodes.

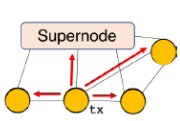

Figure 2: Eavesdropper adversary. A well-connected supernode eavesdrops on relayed communications.

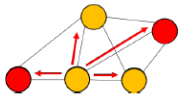

Figure 3: Botnet adversary. Red nodes represent corrupt "spy nodes", which use observed metadata to infer the transaction source.

Figure 4.8: Eavesdropper adversary (Left graph) - Botnet adversary (Right graph)

De-anonymization algorithms are methods that can break the privacy of users who participate in a network that uses anonymization techniques. Anonymization techniques are ways of hiding the personal information of users, such as their names and locations, by using random or fake identifiers. However, these techniques can be defeated by attackers who have access to some extra information or techniques. De-anonymization algorithms are based on the idea of centrality or symmetry, which means that the source of a message is usually located at the center of a disc-shaped region on the network graph. The attackers can use metadata and topology information to infer the shape of the disc and identify the central node with a high probability. These algorithms show that diffusion, which is a common way of spreading messages on the network, is not very good at protecting the anonymity of users at the network level. Therefore, there is a need for alternative spreading protocols that can protect the anonymity of users.

4.5.2 Dandelion

One of the main insights from studying how information spreads is that we need to make the communication protocol asymmetric, so that the adversary cannot easily figure out the message. This is the idea behind the Dandelion P2P network protocol, which has two stages:

1. a **stem stage (or anonymity stage)** :
   In the stem stage, each node sends each message along a random direction. This stage lasts for a random number of hops
2. a **fluff stage (or diffusion stage)** :
   in the fluff stage, the message is diffused to the whole network by broadcasting it

The stem stage provides anonymity, while the fluff stage ensures fast and reliable delivery. We will describe the

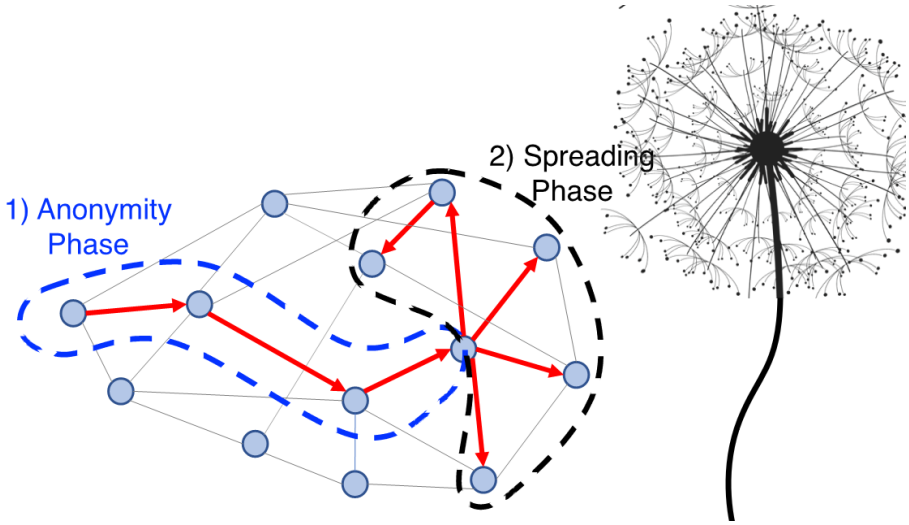

Figure 4.9: Spreading Protocol: Dandelion

Dandelion protocol in more detail next. Dandelion works in epochs that are not synchronized; each node changes its epoch when its internal clock reaches a random value (in practice, this will be a few minutes). Within an epoch, the main algorithmic components of Dandelion are:

1. **Anonymity graph:**
   The random walk happens on a layer of the P2P graph called the anonymity graph. This layer should be either a random cycle graph (i.e., a graph with two edges per node) or a graph with four edges per node. This graph with four edges per node is created from the P2P graph by having each node pick (up to) two of its outgoing edges, without repeating, randomly as Dandelion relays. This does not make an exact graph with four edges per node, but a close one. Each time a node moves to the next epoch, it chooses new Dandelion relays.
2. **Sending of a node's own transactions:**
   Each time a node creates a transaction, it sends the transaction in stem phase along the same random edge

on the anonymity graph. If the anonymity graph is a cycle, there is only one edge per node; otherwise, the node must pick one of its two edges.

3. **Relaying of other nodes' transactions** :
   Each time a node gets a transaction in stem phase from another node, it either passes the transaction on or broadcasts it. The decision to broadcast transactions is pseudo-random, and is based on a hash of the node's own identity and epoch number. Note that the choice to broadcast does not depend on the transaction itself—in each epoch, a node is either a broadcaster or a passer for all passed transactions. If the node is not a broadcaster in this epoch (i.e.,it is a passer), then it passes transactions pseudo-randomly; each node assigns each of its incoming edges in the anonymity graph to an outgoing edge in the anonymity graph (with repetition). This assignment is chosen at the start of each epoch, and determines how transactions are passed.

4. **Robustness mechanism**:
   Each node keeps track, for each transaction in stem phase that it sends or passes on, whether the transaction comes back as a transaction in fluff phase within some random time. If not, the node begins to broadcast the transaction.