Principles of Blockchains
Princeton University,
Professor: Pramod Viswanath

Notes - complete edition

July 30, 2023

Contents

Chapter 1 **Introduction** — Page 3

1.1 What are blockchains 3
1.2 Bitcoin as the first blockchain 5
1.3 What this course is about 7
1.4 What is Rust? 8

Chapter 2 **Blockchains as Cryptographic Data Structures** — Page 9

2.1 Hash Functions 9
2.2 Cryptographic Hash Function 10
2.3 Hash Pointer and chains of block 10
2.4 Merkle tree 11
2.5 Digital Signature 13
2.6 Summary 13

Chapter 3 **Proof of Work and Nakamoto Consensus** — Page 15

3.1 Decentralized Blockchain & Nakamoto consensus 15
Distributed consensus — 15 • Network Assumptions — 16 • Leader election: Oracle — 16
3.2 Proof of Work 17
3.3 Forks and Longest Chain Protocol 17
3.4 Adversarial users 18
3.5 The k-deep confrmation rule 20
3.6 Variable mining difculty 22
3.7 Bitcoin is Permissionless 24

Chapter 4 **Peer to Peer Network** — Page 26

4.1 Blockchains and Networking 26
Types of Network Architecture — 26 • Overlay Networks — 26 • Gossip and Flooding — 27 • Expander Graph — 27
4.2 Bitcoin Network 28
Peer discovery — 28 • Block transmission — 28 • Data Broadcast — 29 • Compact Blocks — 29 • Disadvantages of current p2p network — 30
4.3 Geometric Random Network 30
Perigee — 31
4.4 Efficient Networking 31
FRN (Fast relay network) — 32 • Trusted Networks: Falcon — 32

4.5 Network Anonymity and Privacy 33
Network De-anonymization Problem — 33 • Flooding Protocol — 34 • Anonymity in Botnet adversarial model — 35 • Dandelion — 36

Chapter 5 **Bitcoin system** Page 38

5.1 Basic requirements for a banking system 38
5.2 Bitcoin and Satoshi 38
5.3 Transactions 38
Addressing — 39 • Transaction inputs and outputs — 39 • Signatures on transactions — 40 • UTXO — 40 • Cryptocurrency wallets — 41 • Transaction fees — 42 • Coinbase transactions — 42 • Transaction mempool — 43
5.4 Validating a block 45
The state of the system — 45 • Validating blocks — 46 • Light nodes and stateless clients — 47
5.5 Smart contracts 47
Scripts in Bitcoin — 48

Chapter 6 **Safety of Bitcoin** Page 49

6.1 Introduction 49
6.2 Mining as a Poisson process 49
6.3 Nakamoto consensus protocol model 51
6.4 Formal defnitions of safety 52
6.5 Private attack 52
6.6 Analysis of the private attack under zero delay 53
6.7 Private attack is the worst-case attack 54
6.8 Safety analysis under bounded network delay 55
6.9 Nakamoto blocks 56
6.10 Importance of synchronous network 58

Chapter 7 **Liveness of Bitcoin** Page 59

7.1 What is Liveness 59
7.2 Chain Growth 60
7.3 Chain Quality 61
7.4 Selfish Mining 62
7.5 Fruitchains 62
Cryptographic Sortition — 62 • Fruitchain Protocol — 63 • Short time scale optimal CQ — 64

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,

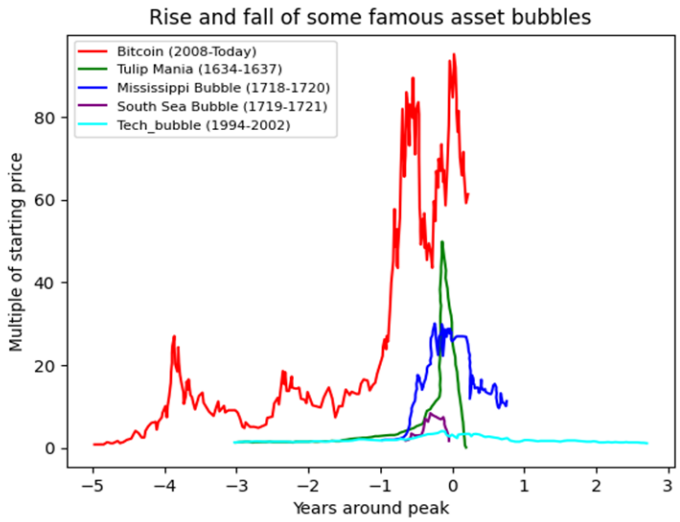

Figure 1.3: Bitcoin and Bubbles

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).

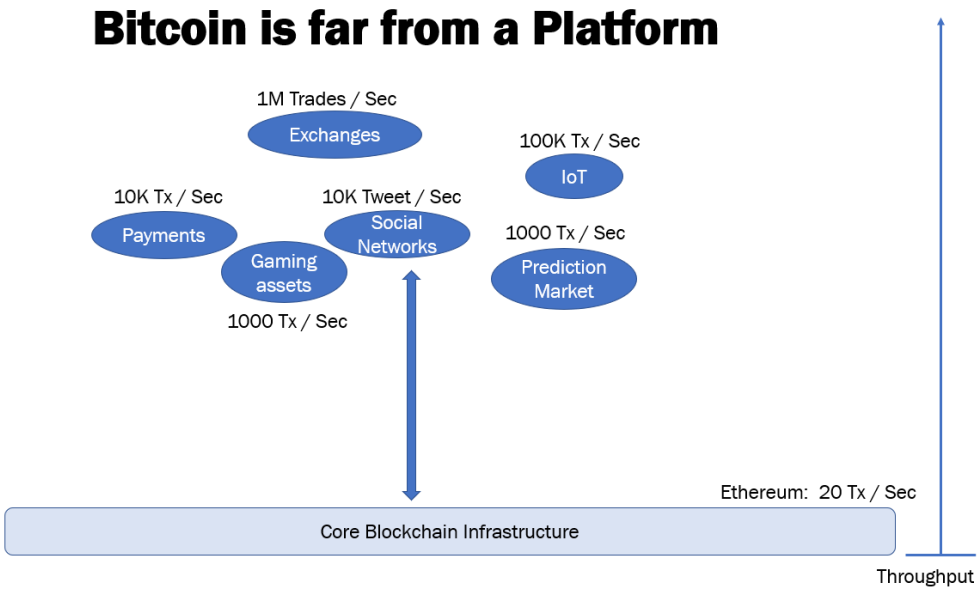

Figure 1.4: Evolution of Trust over human history

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.

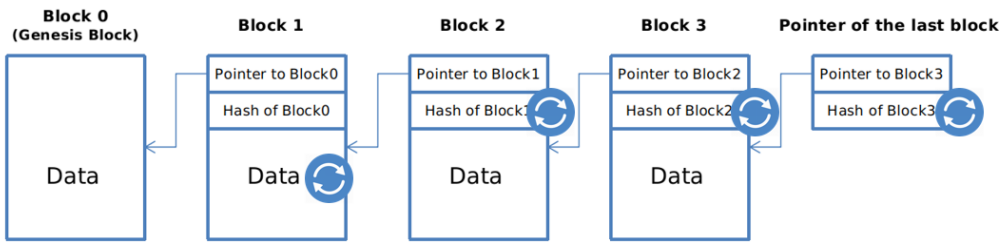

Figure 2.1: A block chain is a linked list that is built with hash pointers instead of pointers

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

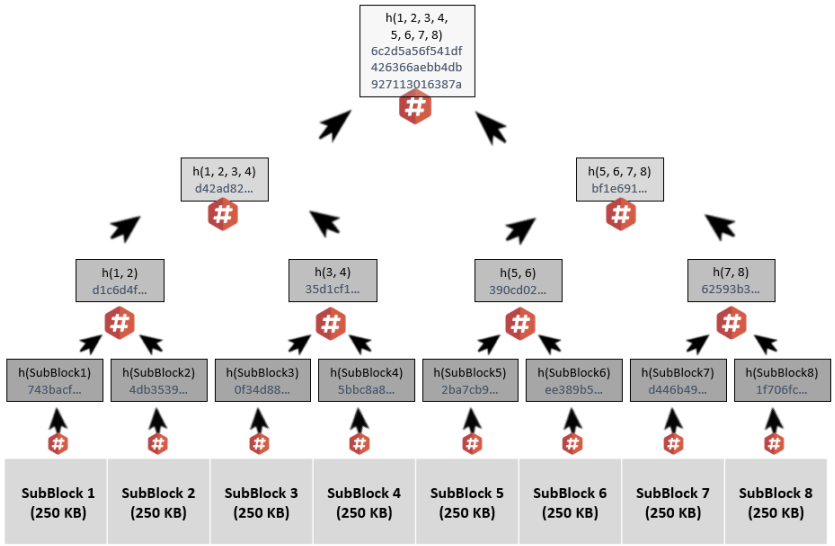

Figure 2.2: A Merkle tree

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.

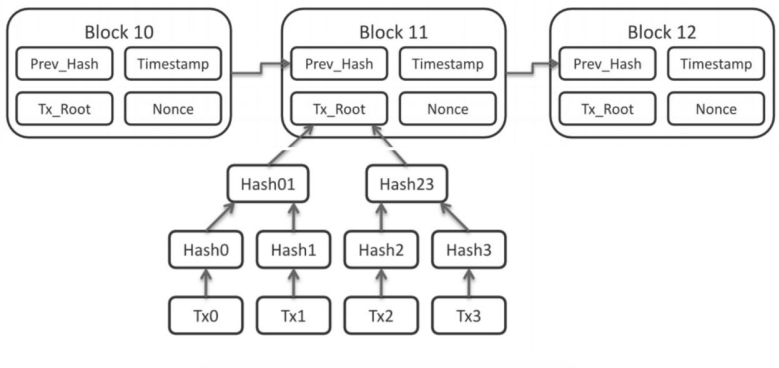

Figure 2.3: Blockchain with markle trees

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.

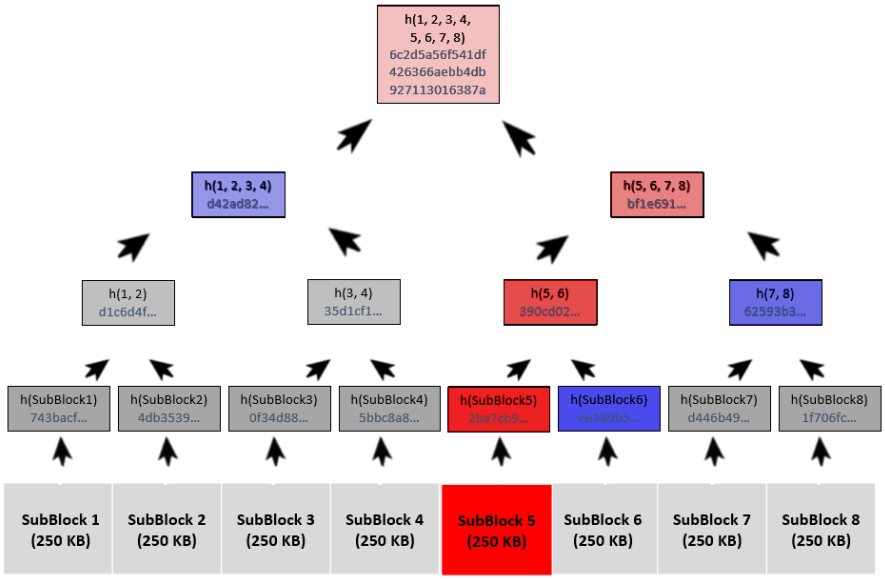

Figure 2.4: Finding a block in Merkle tree

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.

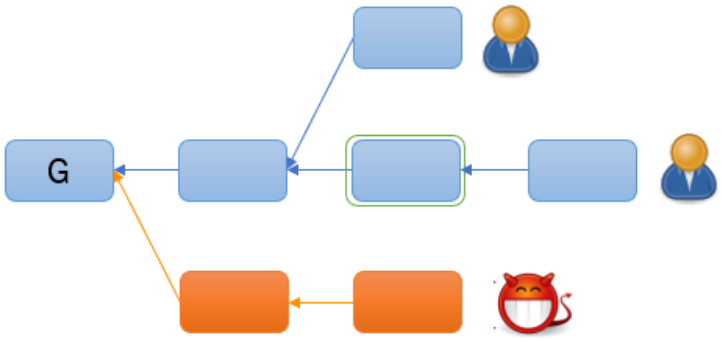

Figure 3.1: Forks in Blockchain

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.

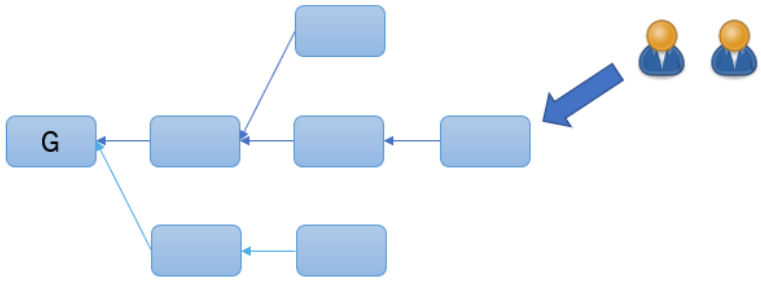

Figure 3.2: The longest chain rule

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.

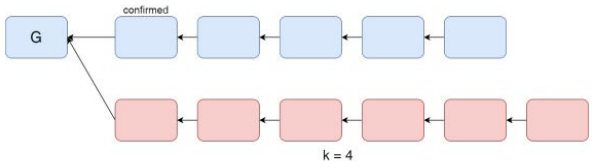

Figure 3.3: An Illustration of the private attack

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.

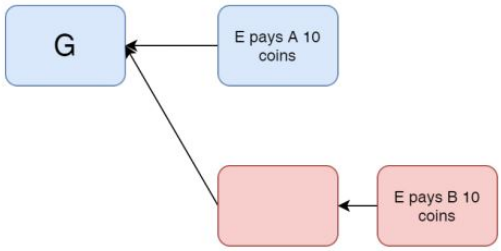

Figure 3.4: An Illustration of the double spend attack.

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∗ → 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].

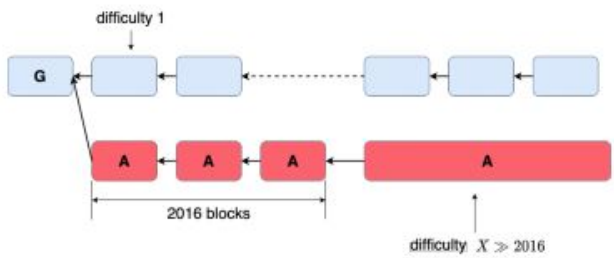

Figure 3.5: The difculty rising attack. The adversary raises the difculty to extremely high in the second epoch by faking timestamps.

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

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

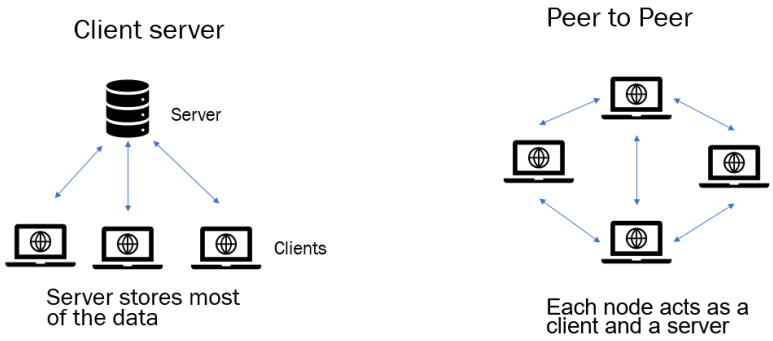

Figure 4.1: Types of Network Architecture

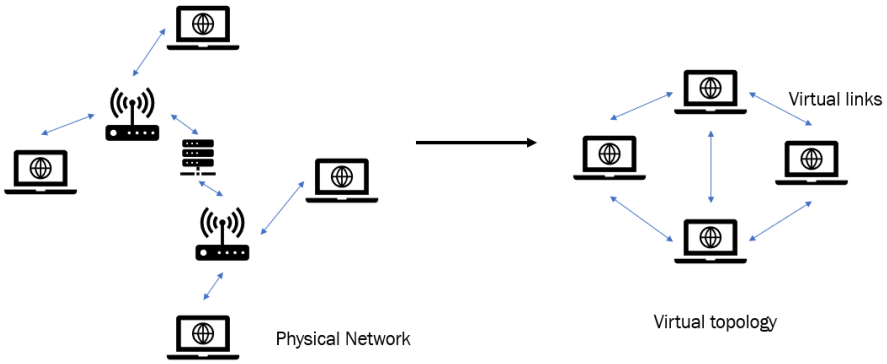

Figure 4.2: Overlay Network

- **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 .

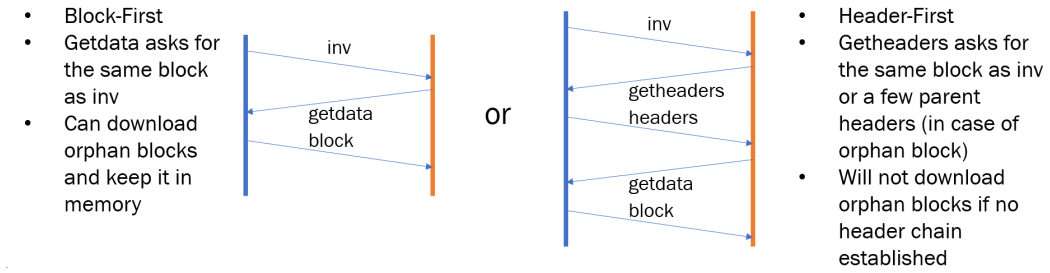

Figure 4.3: Block transmission in blockchain network

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

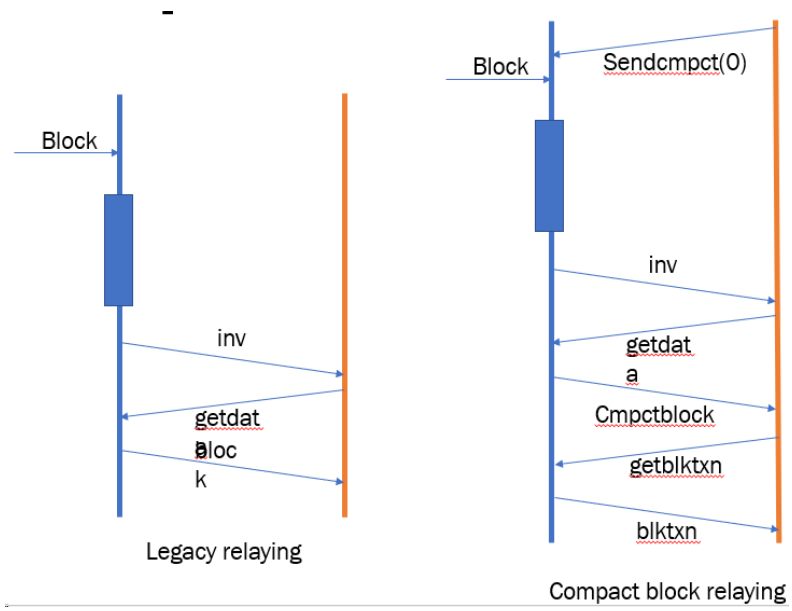

Figure 4.4: Compact Block

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.

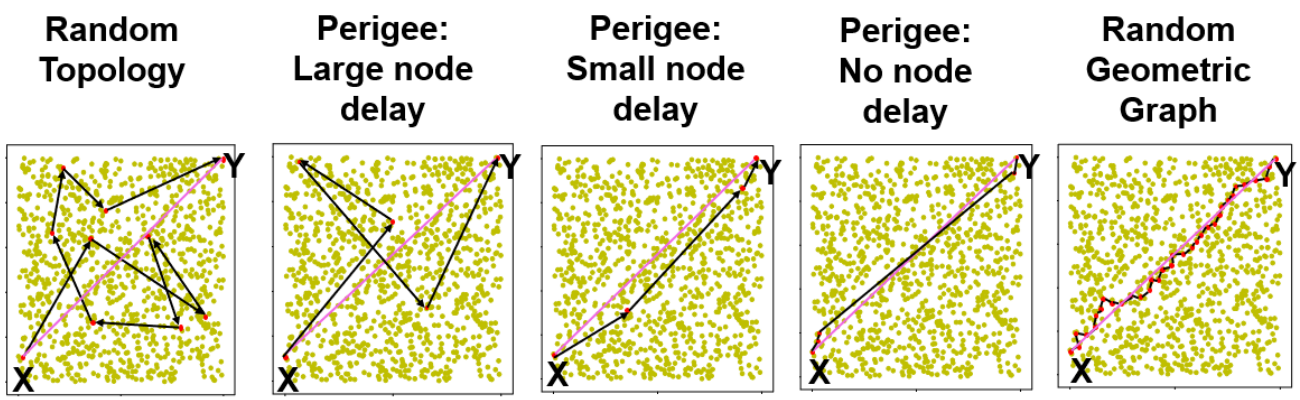

Figure 4.5: Performance of Perigee

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.

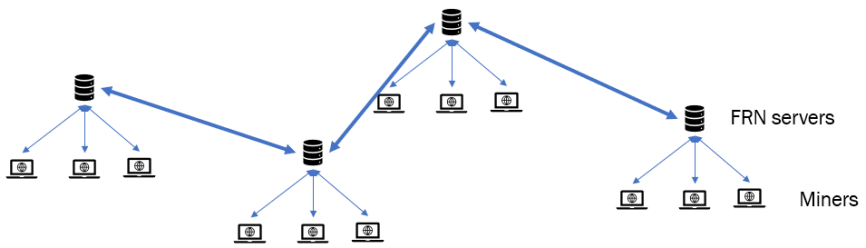

Figure 4.6: FRN

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.

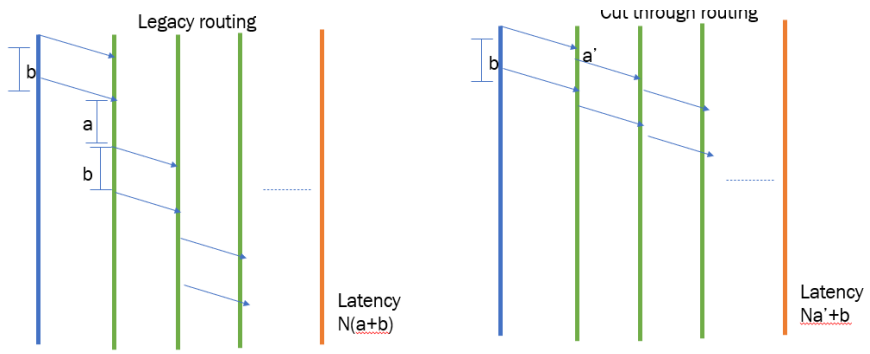

Figure 4.7: Legacy routing vs. Cut through routing

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 Flooding Protocol

Flooding protocols are methods of sending packets to all nodes in a network by having each node rebroadcast the packets it receives.Flooding protocols can be useful for applications such as firmware updates, routing information, and multicast forwarding. However, flooding protocols can also cause problems such as network congestion, packet loss, and high energy consumption.

There are two flooding protocols:

1. **Trickle** :
   Trickle uses a **polite gossip** policy to control the send rates of nodes. Trickle divides time into intervals and each node maintains a counter of how many consistent packets it has heard from its neighbors. If the counter is below a threshold, the node transmits its packet with a high probability. Otherwise, the node transmits its packet with a low probability. This way, Trickle ensures that nodes hear enough packets to stay consistent, but not too many to cause redundancy.

2. **Diffusion** :
   Diffusion uses an exponential backoff mechanism to reduce the send rates of nodes. Diffusion assigns each node an exponent value based on its hop distance from the source node. The exponent value determines how long the node waits before transmitting its packet. The closer the node is to the source, the smaller the exponent value and the shorter the waiting time. The farther the node is from the source, the larger the exponent value and the longer the waiting time. This way, Diffusion reduces the number of collisions and retransmissions in the network, and improves the reliability and efficiency of packet delivery.

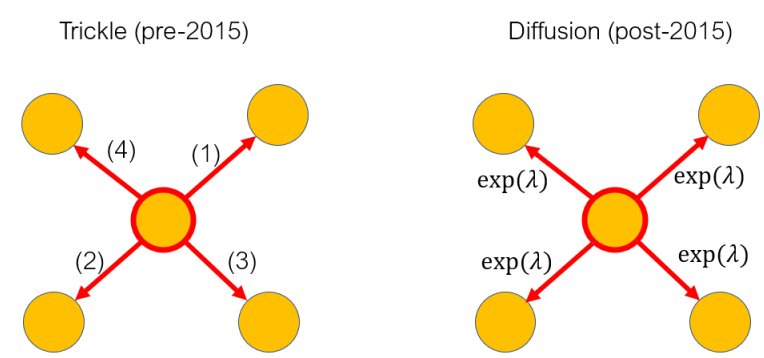

Figure 4.9: Flooding Protocols

Figure 4.10 shows that Trickle has a higher probability of detection than Diffusion for any given number of eavesdropper connections. This means that Trickle is more effective in identifying and reporting eavesdropper connections than Diffusion.
Diffusion does not have (significantly) better anonymity properties than trickle.

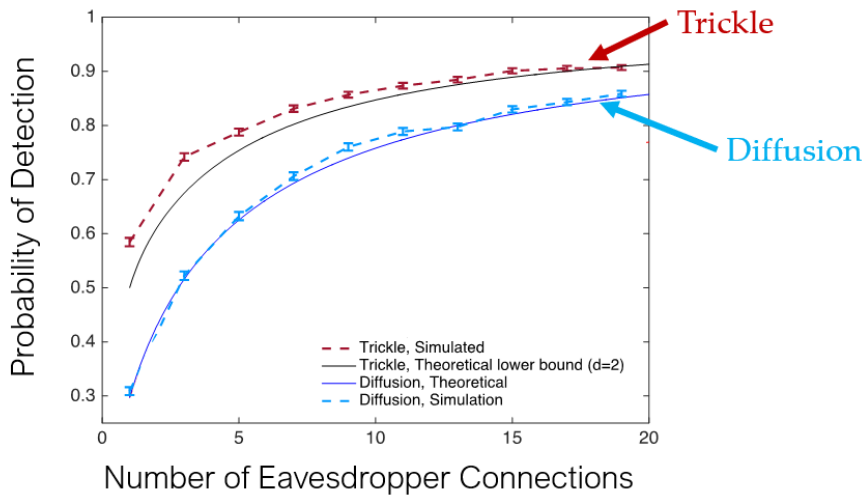

Figure 4.10: Performance of Trickle and Diffusion

4.5.3 Anonymity in Botnet adversarial model

Anonymity is the degree to which a user's identity and transactions are hidden from other users and observers. Anonymity can be measured by two factors: recall and precision. Recall is the probability that a user's transactions are detected by an observer, while precision is the accuracy of the observer's detection. Recall can be calculated

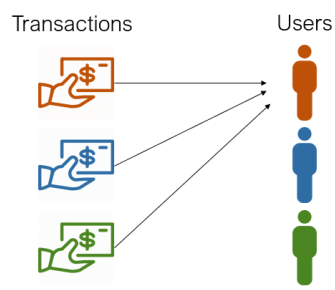

Figure 4.11: Mapping M

using the following formula

$$\frac{1}{n}\sum_{v} 1\{M(v' \text{ s tx}) = v\}$$

where n is the number of honest users in the network, v is a user index, $M(v' \text{ s tx})$ is the mapping function that assigns a user's transactions to a user index, and $1\{M(v' \text{ s tx}) = v\}$ is an indicator function that returns 1 if the mapping is correct and 0 otherwise.

The formula for calculating precision is :

$$\frac{1}{n}\sum_{v} \frac{1\{M(v' \text{ s tx}) = v\}}{\#\text{ tx mapped to } v}$$

Our goal is to design a distributed flooding protocol that minimizes the maximum precision and recall achievable by a computationally-unbounded adversary.

Figure 4.12 implies that we can't have high precision-low recall and vice versa.

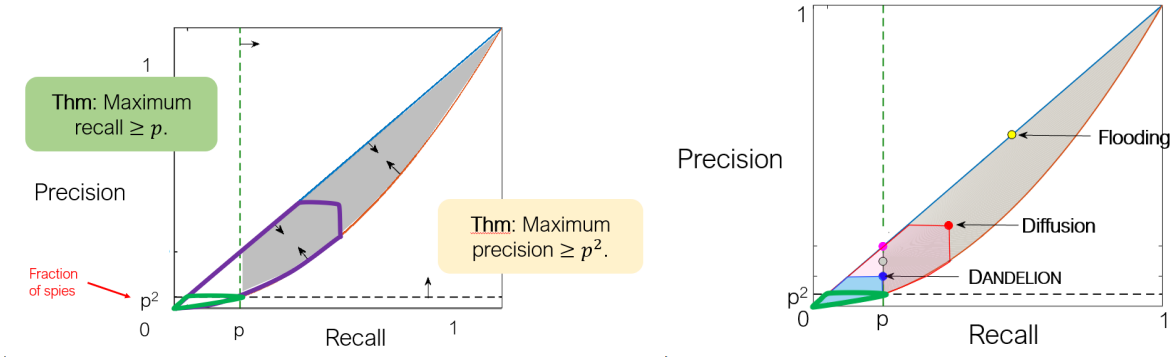

Figure 4.12: Fundamental Limits

4.5.4 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

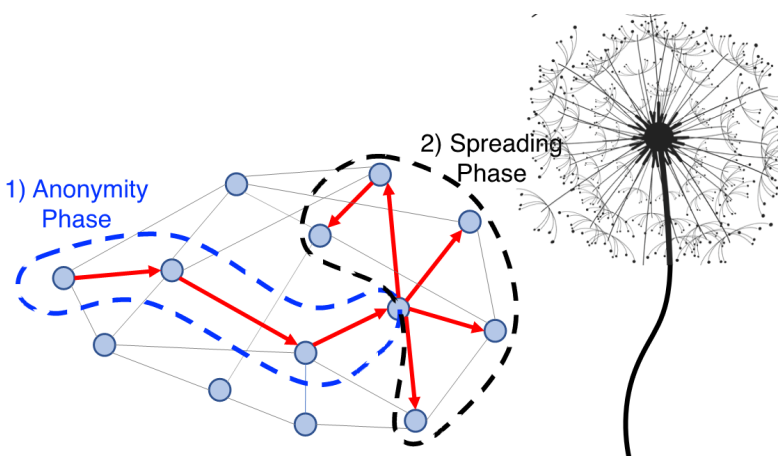

Figure 4.13: Spreading Protocol: Dandelion

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.

Dandelion spreading can achieve the lowest possible recall for any given fraction of spies in the network.

Theorem 4.5.1 Recall of Dandelion

Dandelion spreading has an optimally low maximum recall of

$$p + O\left(\frac{1}{n}\right)$$

.

The theorem implies that no algorithm can reduce the accuracy of detection by an adversary below this bound.

Theorem 4.5.2 Precision of Dandelion

Dandelion has a nearly-optimal maximum precision of

$$\frac{2p^2}{1-p}\log\left(\frac{2}{p}\right) + O\left(\frac{1}{n}\right) \qquad \text{for } p < \frac{1}{3}$$

Figure 4.14 is a timeline chart that shows the evolution of privacy technologies in cryptocurrencies.

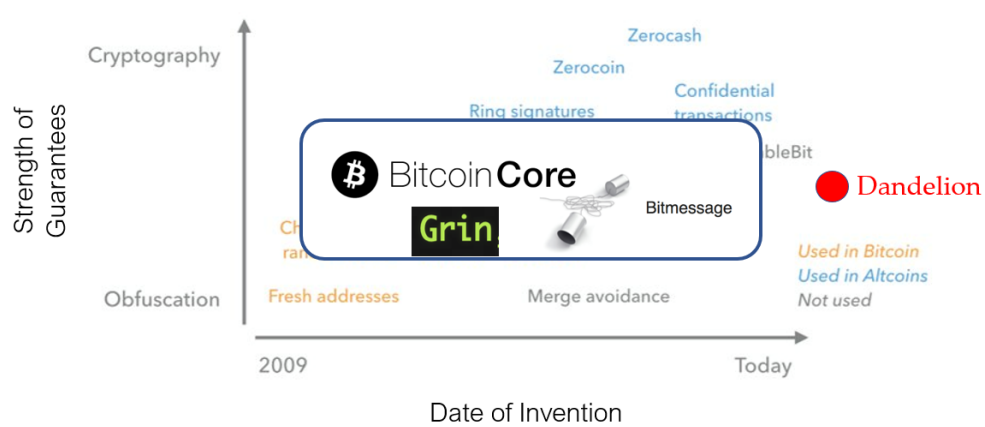

Figure 4.14: Evolution timeline

Chapter 5

Bitcoin system

5.1 Basic requirements for a banking system

In any currency/banking system, there are some basic requirements that the system must provide for. We list them here:

1. There should be a unit of currency/money.
2. There should be a standard way of keeping accounts, i.e., keeping track of how much money each person owns, and transferring money between accounts.
3. No user should be able to create new money from thin air. Put differently, there should be a fixed amount of money in the system at any given time, and new money should be introduced in a systematic manner.
4. A user should not be able to spend more money than he/she owns. There should be a way to verify whether or not this happens.
5. One user should not be able to spend someone else's money (at least, not without their permission).

Let us see how the Bitcoin system provides these features.

5.2 Bitcoin and Satoshi

1. The basic unit of currency in the Bitcoin system is Bitcoin, and the smallest denomination is called a Satoshi, which is equal to 10^{-8} Bitcoins.
2. All transactions in Bitcoin must be an integer multiple of a Satoshi.
3. Just like other currencies, there is an exchange rate between Bitcoin and the dollar. As of July 22, 2023, one Bitcoin is worth 30,000 US dollars. However, the exchange rate between Bitcoin and dollars is very volatile due to various factors, including greed and speculation.
4. The classification of Bitcoin as a currency (like the US Dollar) or a store of value (like gold) has been widely debated. The mainstream view is that Bitcoin is a combination of both.
5. Economically valuing Bitcoin, both in the short and long terms, is an active area of research.
6. Understanding the economic aspects of Bitcoin, both as a currency and a store of value, involves studying various aspects of the Bitcoin system.

5.3 Transactions

In ordinary parlance, the term **transaction** refers to an exchange of something of value. In the context of Bitcoin and cryptocurrencies, a transaction is simply a message that specifes the transfer of money from one entity to another. In fact, transactions are the data-values that get recorded on the blockchain. The blockchain as a ledger is therefore an ordered list of transactions. From this publicly verifable ledger, any user can detect whether transactions are made according to certain rules or not, thereby lending credibility to the ledger and the currency.

5.3.1 Addressing

1. **Bitcoin Address and Its Generation:**
   - In Bitcoin, the notion of traditional accounts is replaced with addresses.
   - An address is simply the hash of a user's public key. It is a unique alphanumeric string that serves as a destination for receiving bitcoins.
   - Addresses are also used to determine where bitcoins will be sent in a transaction.
   - New pairs of public and private keys, and thus new addresses, can be generated at will by a single user.
2. **Receiving vs. Spending Coins:**
   - To receive coins, a user only needs to share their Bitcoin address with others. The address serves as a public identifier for receiving funds.
   - However, to spend coins, a user must also reveal the corresponding public key associated with the address.
   - This is because spending requires providing proof of ownership through a digital signature, which is created using the user's private key.
   - The idiosyncrasy lies in the fact that while an address (hash of public key) is publicly visible and used for receiving, the associated public key is only revealed during the spending process for cryptographic verification.

5.3.2 Transaction inputs and outputs

1. **Transaction Inputs:**
   - Each transaction input represents the amount of Bitcoin being spent from a specific address.
   - When a user initiates a transaction, they reference one or more previous unspent transaction outputs (UTXOs) from the blockchain that they have the right to spend.
   - These UTXOs serve as the inputs to the new transaction and determine the source of the funds being spent.
   - Each input includes a reference to the UTXO's transaction ID and its output index, along with a cryptographic signature to prove ownership.
2. **Transaction Outputs:**
   - Each transaction output represents the amount of Bitcoin being received by a specific address.
   - When a transaction is created, it typically includes multiple outputs, each specifying the amount of Bitcoin and the recipient's address.
   - These outputs determine the destinations of the funds being transferred in the transaction.
   - Each output locks the specified amount of Bitcoin to the recipient's address using a locking script that can only be unlocked with the corresponding private key.
3. **Balance Consistency:**
   - In every valid transaction, the total amount of Bitcoin being spent (sum of inputs) must equal the total amount being received (sum of outputs).
   - This ensures that the transaction preserves the overall balance of the Bitcoin system, i.e., no new bitcoins are created, and no bitcoins are lost during the transaction process.

5.3.3 Signatures on transactions

1. **Signing Transactions for Safety:**
   - Each transaction in Bitcoin must be signed by the users who are spending money. This process is a fundamental safety feature that prevents unauthorized access to someone's funds.
   - For every transaction input, the user creating the transaction must sign it using the corresponding private key associated with the address from which the funds are being spent.
   - By signing the transaction, the user proves ownership of the private key and authorizes the transfer of funds.
2. **One-to-One Correspondence: Public and Private Keys:**
   - As mentioned earlier, each address in Bitcoin has a one-to-one correspondence with a public key, and each public key has a corresponding private key.
   - When a user wishes to spend Bitcoins associated with a particular address, they create the transaction and then sign it using the private key linked to that address.
3. **Verification of Signatures:**
   - After a user signs a transaction, they broadcast it to the network along with the corresponding public key (not the private key).
   - Anyone who sees the signed transaction can verify its authenticity by checking whether it was signed with the private key associated with the public key provided.
   - By doing so, other users can validate that the transaction was indeed signed by the rightful owner of the address (and the coins in that address).
4. **Multiple Addresses, Multiple Signatures:**
   - In transactions that spend Bitcoins from multiple addresses, there must be signatures corresponding to each of these addresses.
   - Each input requires a valid signature to prove ownership and authorization for spending the funds associated with that particular address.

5.3.4 UTXO

The UTXO (Unspent Transaction Output) model is a key feature of the Bitcoin system that ensures the security and validity of transactions. Let's summarize the important points about the UTXO model and how it facilitates transactions with multiple inputs and outputs:

1. **UTXO Model for Validating Transactions:**
   - In the UTXO model, every transaction input must be a transaction output of a previous transaction, linking the spending of funds to their source.
   - When a new transaction output is created, it is considered unspent until it gets consumed as an input in a future transaction when the funds are spent.
   - A valid transaction can only include unspent transaction outputs (UTXOs) as its inputs, providing proof that the address indeed has sufficient funds to spend.
2. **Preventing Double-Spending:**
   - By keeping track of UTXOs at all times, honest users can validate every new transaction against this set to prevent double-spending attempts by dishonest users.
   - To spend their money, users must provide a valid chain of ownership through the UTXO history, ensuring that each input is a previously unspent output.
3. **Multiple Transaction Outputs (Outputs and Change):**

- A transaction may have multiple outputs, allowing users to send funds to multiple recipients in a single transaction.
- For example, if a user owns an address with 2 Bitcoins in an unspent output and wants to spend 1 Bitcoin, the transaction will have two outputs: one for the recipient and one for themselves (the change).
- The change output sends the remaining 1 Bitcoin back to the user's address or a new address.

4. **Multiple Transaction Inputs:**
   - Users can include multiple transaction inputs in a single transaction to combine funds from different unspent outputs for larger transactions.
   - For instance, if a user owns an address with 1 Bitcoin from two separate unspent outputs and wants to pay 2 Bitcoins to another address, they can include both unspent outputs as inputs in the transaction.

The UTXO model's design ensures that each transaction is valid, and its structure allows for flexibility in handling various scenarios, such as spending from multiple sources and sending funds to multiple recipients in one transaction. The adoption of the UTXO model is one of the key factors that contribute to the security and integrity of the Bitcoin blockchain.

5.3.5 Cryptocurrency wallets

Cryptocurrency wallets play a crucial role in simplifying the process of managing and transacting with cryptocurrencies like Bitcoin. Here are the key points about cryptocurrency wallets:

1. **Complexity of UTXO Management:**
   - Keeping track of UTXOs and measuring them for each transaction can be complex and challenging for regular users.
   - Additionally, to maintain anonymity, users often generate new addresses regularly, which requires careful handling of private keys and secure storage.
2. **The Role of Cryptocurrency Wallets:**
   - Cryptocurrency wallets are software applications that handle many of the complex tasks in the background, making it easier for users to transact in cryptocurrencies.
   - Wallets manage UTXOs, track balances, create new addresses, and handle cryptographic signatures for transaction verification.
3. **Anonymity and Security:**
   - Wallets ensure that new addresses (and the corresponding keys) are generated securely without revealing private keys.
   - For maintaining anonymity, wallets often offer features like generating new addresses for each transaction, making it harder to link transactions to a specific user.
4. **Secure Storage and Usage:**
   - Wallets provide a secure storage solution for private keys, protecting them from unauthorized access or theft.
   - Users must be cautious and responsible for securing their wallet's private keys since compromising them could result in the loss of funds.
5. **Trust in Wallet Software:**
   - Using a cryptocurrency wallet requires placing trust in the software's functionality and security.
   - Using a cryptocurrency wallet requires placing trust in the software's functionality and security.

5.3.6 Transaction fees

Transaction fees play a crucial role in the Bitcoin network and serve as an incentive for miners to include transactions in the blocks they mine. Here are the key points about transaction fees:

1. **Total Value in Inputs and Outputs:**
   - In a Bitcoin transaction, the total value in the inputs (UTXOs being spent) must be equal to or greater than the total value in the outputs (newly created UTXOs).
   - The difference between the total value in inputs and outputs is known as the transaction fees.
2. **Transaction Fees as Miner Incentives:**
   - The transaction fees are claimed by the miner who successfully includes the transaction in a block and adds that block to the blockchain.
   - Miners compete to add transactions to their blocks because the fees they collect are an additional reward on top of the block reward (newly minted Bitcoins).
3. **Transaction Prioritization and Speed:**
   - Transactions with higher fees are more attractive to miners because they earn more rewards for including them in their blocks.
   - Miners prioritize transactions with higher fees, leading to faster inclusion of these transactions in the blockchain.
   - Transactions with lower fees may take longer to be confirmed since they are lower in the priority list.
4. **Automatic Fee Calculation:**
   - Wallets automatically calculate transaction fees based on a particular fee rate, measured in Satoshi per kilobyte (Satoshi/kB).
   - The fee rate determines the amount of Satoshi to be paid per kilobyte of transaction data.
   - Higher fee rates result in faster confirmation times, while lower rates may lead to delayed confirmations.
5. **Dynamic Fee Variation:**
   - Transaction fees can vary over time due to fluctuations in network demand and block space availability.
   - During periods of high network congestion, transaction fees may increase as users compete for limited block space.

5.3.7 Coinbase transactions

Coinbase transactions are a crucial mechanism for introducing new Bitcoins into the Bitcoin system and rewarding miners for their efforts in processing transactions and adding blocks to the blockchain. Here are the key points about coinbase transactions:

1. **Introduction of New Bitcoins:**
   - New Bitcoins are introduced into the system as a reward for miners who successfully mine a new block.
   - Every block includes a special transaction known as the "coinbase transaction," which allows the miner to claim a fixed number of newly created Bitcoins.
2. **Block Reward Halving:**
   - Initially, when Bitcoin was launched, the block reward for miners was 50 BTC per block.
   - Approximately every 210,000 blocks (about four years), the block reward is halved through an event known as "halving."
   - This means that after each halving, miners receive half the number of Bitcoins as the previous reward.
3. **Current Block Reward:**

- As of now, there have been three halvings, and the current block reward for miners is 6.25 BTC per block.
- The block rewards will continue until the year 2140 when the total supply of Bitcoin will reach its cap.

4. **Fixed Supply of Bitcoin:**
   - The total supply of Bitcoin is capped at 21 million coins.
   - This fixed supply ensures that there will never be more than 21 million Bitcoins in circulation.
   - Approximately 19.4 million coins are already in circulation, and the remaining Bitcoins will be gradually introduced through coinbase transactions until the cap is reached.
5. **Incentive Mechanism:**
   - Coinbase transactions, along with transaction fees, serve as incentives for miners to actively participate in the network and secure the blockchain through the proof-of-work process.
   - In the early years of Bitcoin, coinbase transactions formed a significant portion of the rewards for miners, but over time, the contribution of transaction fees has increased.

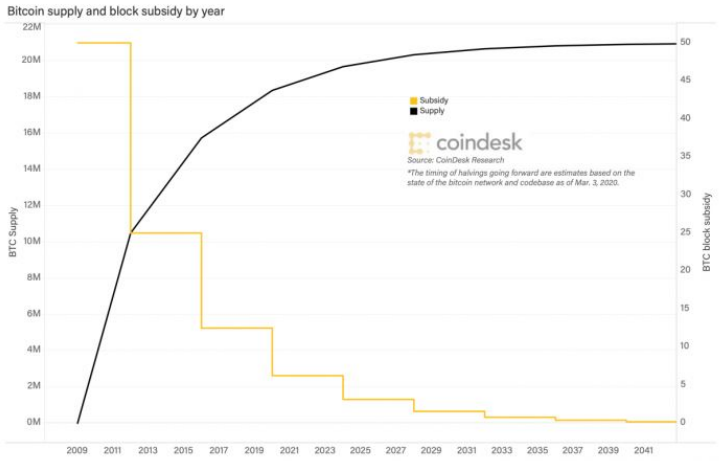

Figure 5.1: Bitcoin block reward in bitcoins. Figure sourced from here

The combination of coinbase transactions and transaction fees ensures the proper functioning and security of the Bitcoin network. Miners are motivated to continue mining, securing the network, and processing transactions as they receive both block rewards and transaction fees as incentives for their efforts. The gradual reduction in block rewards through halvings also creates a deflationary monetary policy, which further contributes to Bitcoin's scarcity and potential value appreciation over time (Figure 1.1, 1.2).

5.3.8 Transaction mempool

The transaction mempool plays a crucial role in the process of how Bitcoin transactions are propagated, verified, and eventually included in blocks. Here are the key points about the transaction mempool and its significance:

1. **Transaction Propagation:**
   - When a user wants to make a Bitcoin transaction, they create a transaction message, sign it using their private key, and broadcast it across the Bitcoin network.
   - The transaction message is a few kilobytes in size and contains information about the transaction inputs, outputs, and the corresponding digital signatures.
   - Miners in the network receive these transactions and verify the signatures before adding them to their memory pool, known as the "mempool."

Figure 5.2: Bitcoin rewards in dollars. Figure sourced from here

2. **The Mempool:**
   - The mempool is a temporary storage area where miners keep pending transactions that they have received and verified.
   - Transactions in the mempool are waiting to be included in a block and added to the blockchain.
   - Miners prioritize which transactions to include in their working block based on factors like the transaction fee rate (higher fee with smaller size).
3. **Block Size Limit:**
   - Each block in the Bitcoin blockchain has a maximum size limit, currently set at 1 megabyte (MB) in the Bitcoin network.
   - Miners aim to maximize the total transaction rewards in their block while adhering to this size limit.
4. **Transaction Fee Rates:**
   - Miners prioritize transactions with higher transaction fees because they want to maximize their potential reward for including transactions in a block.
   - Transactions with higher fees are more likely to be included in blocks sooner, incentivizing users to offer higher fees if they want their transactions processed quickly.
5. **Confirmation Latency:**
   - There is a certain latency between the time a transaction is issued and when it is confirmed on the blockchain. This latency is influenced by two factors: (a) the time it takes for a transaction to be included in a block (can be reduced by offering a higher transaction fee), and (b) the time it takes for that block to be buried deep enough in the longest chain for the transaction to be considered confirmed.
   - Users may choose to trade off latency for security, deciding whether to use higher transaction fees for faster confirmation or lower fees for potentially longer confirmation times.
6. **Blockchain Design Trade-offs:**
   - The trade-off between latency and security is specific to the Bitcoin blockchain design.
   - Other blockchain designs may have different approaches and trade-offs, which will be explored in future lectures.

The mempool and the prioritization of transactions based on fees are essential components that allow the Bitcoin network to efficiently process and confirm transactions while incentivizing miners to maintain the security of the blockchain through the proof-of-work process (Figure 1.3).

5.4 Validating a block

5.4.1 The state of the system

The state of the blockchain system, also known as the UTXO set (Unspent Transaction Output set), is a critical component in ensuring the integrity of the cryptocurrency network. Here are the key points about the state of the system and its significance in Bitcoin:

1. **Verifying Transaction Double-Spending:**
   - One of the essential tasks in the Bitcoin system is to prevent double-spending, where a user attempts to spend the same coin or UTXO more than once.
   - To verify the validity of a new transaction, users need to check if the transaction's inputs (UTXOs) have not been spent previously.
2. **Unspent Transaction Outputs (UTXOs):**
   - The state of the system is represented by the set of UTXOs, which includes all transaction outputs that have not been spent in the blockchain.
   - UTXOs are created when transactions are included in the blockchain and are considered valid until they are spent by new transactions.
3. **Separate from the Mempool:**
   - The state of the system is distinct from the mempool, which contains pending transactions yet to be confirmed.
   - The state consists of UTXOs that have been confirmed and permanently recorded in the blockchain.
4. **Role in Transaction Validation:**
   - When miners construct new blocks from transactions in the mempool, they only include valid transactions by checking the information in the state (UTXO set).
   - Transactions that attempt to spend already spent UTXOs are rejected as invalid.
5. **Summary of Ledger Entries:**
   - The state contains a summary of all the confirmed ledger entries in the blockchain until the current point in time.
   - This summary is crucial for validating new transactions and ensuring their consistency with previous transactions.
6. **Growth and Size Challenges:**
   - The UTXO set grows over time as more transactions are confirmed and new UTXOs are created.
   - As of now, the UTXO set size is quite large, over 5 GB, and it continues to increase as Bitcoin gains popularity.
   - The size of the UTXO set presents scalability challenges, especially concerning memory requirements and specialized hardware for storage and management.

Overall, the state of the system (UTXO set) plays a vital role in maintaining the integrity of the Bitcoin network, preventing double-spending, and ensuring that new transactions are consistent with previous ones. Managing the growth of the UTXO set is an important aspect of addressing scalability concerns in the blockchain system.

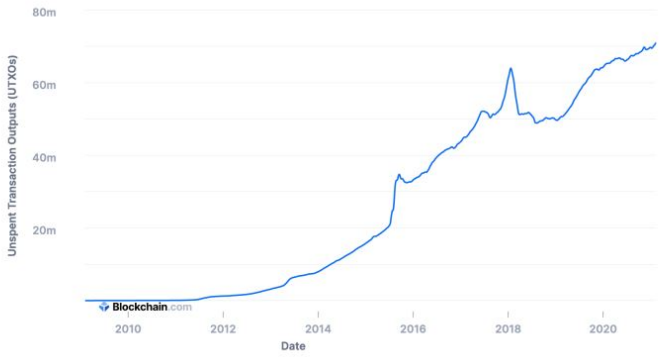

Figure 5.3: Number of UTXOs over time in Bitcoin. Figure sourced from here

5.4.2 Validating blocks

Validating blocks is a crucial step for miners in the Bitcoin network to ensure that the newly received or constructed block adheres to the rules and integrity of the blockchain. Here are the important points about block validation:

1. **Proof-of-Work (PoW) Check:**
   - The first check performed by a miner is to verify whether the proof-of-work presented in the block meets the required threshold.
   - The proof-of-work is a computationally intensive puzzle that miners must solve to create a new block, and it ensures the security and immutability of the blockchain.
   - If the proof-of-work is valid, the miner can proceed with further checks.
2. **Transaction Consistency Check:**
   - After validating the proof-of-work, the miner needs to verify whether the transactions included in the block are consistent with the current state (UTXO set).
   - This consistency check is crucial to prevent any attempts at double-spending or invalid transactions.
   - The miner must confirm that each input in every transaction is a valid unspent transaction output (UTXO) according to the current state.
3. **Updating the State:**
   - If all the transactions in the block pass the consistency check, the miner accepts the block, removes the relevant transactions from its mempool, and updates the state (UTXO set).
   - Updating the state involves deleting the spent transaction outputs (UTXOs) that were used as inputs in the block's transactions.
   - Additionally, the newly generated unspent transaction outputs (UTXOs) are added to the state, representing the funds transferred in the block.
4. **Preemptive Validation in Block Construction:**
   - Before a miner starts mining a new block, it performs the same validation steps preemptively on the transactions it intends to include in the block.
   - This ensures that the constructed block will be accepted by the network once it is mined, saving computational effort.

In summary, block validation is a critical process to maintain the integrity of the blockchain. Miners verify the proof-of-work and the consistency of transactions with the current state (UTXO set). Once validated, a new block is added to the blockchain, and the state is updated to reflect the changes made by the included transactions.

5.4.3 Light nodes and stateless clients

Light nodes and stateless clients are important concepts in blockchain networks, especially for enhancing scalability and allowing users with limited resources to participate in the system. Here's an explanation of these concepts:

1. **Light Nodes / Light Clients:**
   - Light nodes, also known as light clients, are participants in the blockchain network who do not store the full state of the system.
   - Instead, they only verify the proof-of-work on blocks to ensure their validity, and they may selectively verify specific transactions that concern them.
   - By avoiding the storage of the entire state, light nodes require much less computation and storage power compared to full nodes, making them more accessible to devices with limited resources.
2. **Stateless Clients:**
   - Stateless clients are an extension of light nodes that aim to validate transactions and blocks without maintaining the full state of the blockchain at all times.
   - Instead of storing the entire state, stateless clients download only the necessary state information on-demand to validate a specific block or transaction.
   - Once the validation is complete, the stateless client can delete the downloaded state information to save storage space.
   - To securely handle the state information, it is stored in the form of an accumulator, a data structure that we discussed in Lecture 2 of the blockchain course.
3. **Advantages of Stateless Clients:**
   - The adoption of stateless clients can significantly enhance the scalability of blockchain networks. It allows more users to participate without the need for extensive resources to store the full state.
   - By reducing the storage requirements for clients, the network becomes more inclusive and accessible to a wider range of devices, including mobile phones and low-powered computers.
   - Stateless clients can still contribute to the network's security by validating transactions and blocks, albeit without maintaining a persistent copy of the state.
4. **Implementation in Ethereum:**
   - The concept of stateless clients is actively pursued by the Ethereum blockchain network as a scalability solution.
   - Ethereum aims to transition from a stateful design (where clients store the full state) to a stateless design, using stateless clients to improve the network's performance and reduce resource requirements.

In conclusion, light nodes and stateless clients are innovative approaches to make blockchain networks more scalable and inclusive. By allowing participants to interact with the network without storing the full state, these concepts address the challenges of resource-intensive storage and computation in traditional blockchain architectures.

5.5 Smart contracts

In the context of blockchain technology, smart contracts are self-executing contracts with the terms and conditions written directly into code. They are programmed to automatically execute and enforce the terms of the contract without the need for a trusted third party. Smart contracts run on blockchain networks, and their execution is validated and recorded on the blockchain, making them transparent and immutable.
In traditional settings, when two parties want to exchange money subject to certain conditions, they would create a written contract and rely on a trusted intermediary or authority to enforce the contract's terms. With smart contracts on a blockchain, this intermediary is eliminated, and the trust is decentralized across the network.
In summary, smart contracts are a powerful innovation that leverages blockchain technology to enable secure, automated, and transparent execution of agreements without the need for intermediaries. While Bitcoin's smart contract capabilities are limited, other blockchain platforms like Ethereum have expanded the possibilities of smart contracts to revolutionize various industries and applications.

5.5.1 Scripts in Bitcoin

In the context of Bitcoin, smart contracts are referred to as "scripts." A script is a sequence of opcodes included in every transaction output, specifying the conditions that must be satisfied to spend the coins from that output. By using different combinations of opcodes, Bitcoin scripts can enforce various conditions beyond the default ones, providing a level of programmability.

The default condition in a regular transaction output requires the spender to provide a public key that matches the address mentioned in the output and sign the transaction with the corresponding private key. This ensures that only the rightful owner of the coins can spend them.

However, Bitcoin scripts allow for additional conditions to be specified. Some examples include:

1. **Timelocks:** A transaction can be configured to be unspendable until a certain block height or a specific time is reached. This can be useful for time-sensitive transactions or implementing time-based conditions.

2. **Multisig:** A script can require multiple signatures from different private keys to spend the coins. This allows for escrow-like arrangements, where funds are held until all parties involved in the transaction provide their signatures.

3. **Hashed Timelock Contracts (HTLCs):** HTLCs are a specific type of smart contract that combines time locks and hash locks. They are useful for setting up secure escrow arrangements for exchanging assets or payments between parties, ensuring that both parties fulfill their obligations before the funds are released.

The combination of these conditions in Bitcoin scripts enables the creation of more sophisticated smart contracts, even though Bitcoin's scripting language is not as powerful as the one used in platforms like Ethereum. While Bitcoin's script language is intentionally limited for security reasons, it still provides enough flexibility to implement useful smart contract functionalities, especially for simple financial transactions and basic programmability.

Chapter 6

Safety of Bitcoin

6.1 Introduction

The security of a blockchain system, such as Bitcoin, is of utmost importance to ensure the integrity and trustworthiness of the network. When analyzing the security of a blockchain, it is crucial to consider the presence of adversaries who may try to disrupt the system for their own gain. In the context of blockchain security, adversaries are often modeled as powerful entities with significant resources, including more information, computational power, and control over the network compared to honest participants.
Bitcoin incorporates several security measures to protect honest users from adversarial attacks. These measures include:
Proof-of-Work Verification: Honest nodes verify the proof-of-work in new blocks, ensuring that miners have expended sufficient computational effort to create a valid block. This prevents adversaries from easily generating new blocks and taking control of the blockchain.
Transaction Verification: Honest nodes validate signed transactions, ensuring that only rightful owners can spend their funds. Transactions that do not follow the rules are ignored.
UTXO Validation: Honest nodes keep track of the unspent transaction outputs (UTXOs) and verify that each transaction's inputs reference valid UTXOs. Invalid transactions are not accepted.
While these sanity checks are straightforward and easy for honest participants to perform, attacks at the level of the Nakamoto consensus protocol, which determines the longest chain and the agreed-upon version of the blockchain, are more complex and subtle. Adversaries can attempt to disrupt the consensus protocol in ways that are not immediately evident to honest players.
Formal mathematical models are used to analyze the security of the Nakamoto consensus protocol. These models treat the mining process as a stochastic process, where new blocks are generated at random intervals of time. The use of stochastic modeling allows for probabilistic analysis of the protocol's security guarantees. The security guarantees in this context are probabilistic because they depend on the randomness inherent in the mining process.
By applying formal mathematical methods and probabilistic analysis, researchers can gain insights into the security properties of the Nakamoto consensus protocol. This includes understanding the likelihood of successful attacks by adversaries and the resilience of the blockchain system to such attacks.

6.2 Mining as a Poisson process

A Poisson process is a stochastic process where events occur at random intervals, following the exponential distribution. In this case, the events correspond to the mining of new blocks. The intervals between any two block arrivals are statistically identical and independent of each other.
If we denote the time between consecutive block arrivals as X, then X follows an exponential distribution with parameter . The probability that the time between two block arrivals is greater than or equal to t is given by:

$$P(X \geqslant t) = e^{-\lambda t} \quad \text{for all } t \geqslant 0.$$

This probability is often used to analyze the behavior of the Poisson process and to estimate the time it takes for the next block to be mined, given the current rate . The exponential distribution is particularly useful

in modeling Poisson processes, as it captures the randomness and memorylessness properties of the arrival times in the process.
Indeed, a Poisson random variable Y with parameter has a probability mass function given by:

$$P(Y = k) = \frac{e^{-\lambda}\lambda^k}{k!} \quad \text{for all } k \geqslant 0.$$

In a Poisson process, the number of events (block arrivals in this case) in an interval of length T follows a Poisson distribution with parameter T, assuming the average rate remains constant. Furthermore, the number of events in disjoint intervals of time are statistically independent.
If we consider small intervals of time (T 1), the probability of having one event (block mined) in the interval is approximately T, and the probability of having no events (no block mined) is approximately 1 - T. This approximation is justified because T is very small, given the average rate is generally small when expressed in terms of the number of blocks per unit of time.
This is where the analogy with coin tosses comes in. A Poisson process can be emulated by counting the occurrence of "successes" (in this case, block arrivals) in a sequence of independent Bernoulli trials (coin tosses), where the probability of success (head) is very small (approximately equal to T).
In modeling the mining process as a Poisson process, we assume that the total number of hashes computed (or nonces tried out) per unit time remains roughly constant, even if the number of miners and their individual computation power may vary. This assumption allows us to treat the mining process as a random process, where new blocks are found at random intervals.
Let's recap some important modeling assumptions about the mining process in the context of the Nakamoto consensus protocol:

1. **Constant Mining Rate ( ):** We assume that the mining rate (the average number of blocks mined per unit time) is a constant, denoted by . In Bitcoin, the average block time is about 10 minutes, so is approximately 1 block every 600 seconds. In Ethereum, the block time is much faster, so the value of is higher.

2. **Poisson Process:** The mining process is modeled as a Poisson process, where new blocks are found at random intervals following an exponential distribution. The Poisson process assumption holds because the probability of successfully mining a block in a given second is very small (0.03 in the example given), and the mining process is essentially like flipping a biased coin (trying out nonces) repeatedly, where each flip is independent of the others.

3. **Independent and Constant Difficulty:** We assume that the difficulty level of the hash puzzle for proposing a block remains constant over some period of time. The difficulty level determines the number of leading zeros required in the hash output, and a high difficulty makes finding a valid block more challenging. In reality, the difficulty is adjusted regularly to maintain a constant mining rate, and this ensures that the time to mine a block remains relatively constant over time.

4. **Uniform Distribution of Valid Nonces:** Valid nonces (hash inputs) are assumed to be uniformly distributed among all possible nonces. This is a common assumption in modeling proof-of-work systems like Bitcoin.

5. **Idealized Network and Synchronization:** The model assumes an idealized network where all nodes receive blocks and transactions immediately, and there are no network delays or partitions. In practice, real-world networks introduce latency and possible partitions, affecting block propagation and communication.

6. **Honest Majority:** We assume that the majority of miners in the network are honest, following the protocol correctly. This is a common assumption in security analysis, as most blockchain systems rely on the assumption that a majority of computational power is controlled by honest participants.

By making some of these modeling assumptions and giving the adversary some extra power, the security analysis can provide guarantees that hold even in practical scenarios where real-world considerations might deviate from the idealized assumptions. The goal is to design a protocol that remains secure and resilient in the face of adversarial attacks, even when the adversary has considerable computational power and control over the network.

6.3 Nakamoto consensus protocol model

In the model, we assume the presence of a single adversary with a fraction of the total computing power (hash power) denoted by β. The remaining computing power, $1-\beta$, is distributed roughly equally among the many honest parties participating in the protocol.
To be more precise:

- The adversary's computing power is β times the total computing power of all users in the system.
- The computing power of honest users is divided equally among them, with each honest user controlling a very small fraction $\frac{1-\beta}{N}$, where N is the total number of honest users.

This means that the adversary mines blocks at a rate of (Poisson process of rate), while honest users mine blocks at a combined rate of $(1-\beta)\lambda$ (Poisson process of rate $(1-\beta)\lambda$). The processes of the adversary and the honest users are independent of each other.
This assumption gives the adversary a significant advantage in terms of mining power, as it controls a substantial portion of the total computing power. The adversary's higher mining rate ($\beta\lambda$) compared to that of honest users ($(1-\beta)\lambda$) allows them to have a higher probability of finding new blocks and building their own chain. However, we still assume that $\beta < \frac{1}{2}$, which means that the honest users collectively have more mining power than the adversary. This assumption is crucial to ensure that the Nakamoto consensus protocol remains secure even in the presence of an adversary.
In the model, we assume that each honest player (party) stores a single blockchain at all times, which is the longest chain they have heard until that specific time. Let's denote the chain held by party h at time i as C_i^h.
Here's a more detailed explanation of the notation:

- i R+: This represents a continuous-time parameter, which denotes the specific point in time during the protocol execution. The time parameter i can take any non-negative real value.
- C_i^h: This notation represents the chain held by party h at time i. The chain C_i^h is a sequence of blocks, starting from the genesis block (the first block) up to the last block at time i that party h has seen and believes to be part of the longest chain.

In this model, honest players continuously update their chain to be the longest chain they have seen so far. This implies that if a party hears of a new block that extends the current longest chain, they update their chain accordingly to include this new block. As a result, the chain C_i^h held by each honest party h may vary over time as new blocks are discovered and added to the blockchain.
The longest chain rule states that honest players always work on extending the longest chain they know, assuming it is the valid chain. As more blocks are mined and added to the blockchain, the chain held by each honest party h (C_i^h) keeps growing as they update it to follow the longest chain.
In the model, when an honest party hears of multiple chains with the same (maximum) length, they choose one of them arbitrarily as C_i^h. This choice is made by the adversary, which means that an honest user may swap its chain when it hears of an equally long chain, not just a strictly longer one. Additionally, if two honest parties both hear of two chains of equal (maximum) length, they may adopt different chains due to this arbitrary tie-breaking power given to the adversary.
To define the notion of the prefix of a chain, we say that chain C1 is a prefix of chain C2 if all blocks in C1 are also present in C2. Formally, we denote this by C1 C2. To represent the prefix of a chain, we use the notation C^h, which represents the prefix chain obtained by dropping the last k blocks from chain C. If chain C has less than or equal to k blocks, C k will be considered as the genesis block.
The k-deep confirmation rule in Bitcoin implies that a node treats all but the last k blocks in its longest chain as confirmed. In the notation of the model, the blocks in C^h are confirmed by user h at time i. Once we confirm a block, we also confirm all the transactions in it. The confirmation rule is applied locally by each user, which means that a transaction confirmed by one user may not be confirmed by another. However, it is desirable that a transaction confirmed by one user becomes confirmed by all other users relatively quickly and remains confirmed forever after. This desirable property is called a safety property in blockchains.
In the context of safety, it is important to ensure that once a block is confirmed (included in the longest chain) by a significant number of honest parties, it is highly unlikely to be reversed or replaced by another chain. In other words, a confirmed block's position in the blockchain should be secure and unlikely to be changed. This property is crucial for maintaining the consistency and integrity of the blockchain, especially in the presence of adversarial

behavior. Ensuring safety is an essential aspect of blockchain protocols to maintain the trust and reliability of the system.

6.4 Formal defnitions of safety

In the Nakamoto consensus protocol, safety is a crucial property that ensures the consistency and integrity of the blockchain. We define safety in two different ways: the common prefix property and individual block safety.
Definition 1 (Common Prefix Property):
The k-common prefix property holds during an execution of the protocol if any block that is committed by one honest user appears in every honest user's chain thereafter. In mathematical terms, for all pairs of times $i_1 \quad i_2$ and for all pairs of honest users h_1, h_2: $C^k C_2$, where $C_1 C_{i_1}^{h^1} and C_1 C_{i_2}^{h^2}$.
Definition 2 (Individual Block Safety): In an execution, a block B present in some honest user's chain is safe if, after it has been committed by any honest user, it remains a part of all honest users' chains. In mathematical terms, if block B is committed by some user h at time t, then for all tt and for all honest users h, $BC_{t'}^{h}$.
Note that the common prefix property subsumes individual block safety. For simplicity, the focus is on individual block safety alone.
These definitions represent desired security properties of the protocol. To ascertain whether such properties hold or with what probability they hold, further calculations and analysis are required. Security guarantees or theorems are typically given with the form that the k-common prefix property holds with a probability approaching one as k approaches infinity. These guarantees are made under specific assumptions, including a fixed mining rate, bounds on the adversary's computing power, and bounds on network delays. Importantly, these guarantees are provided while assuming that the adversary can act arbitrarily, meaning no specific assumptions are made about the adversary's strategy.
To understand how the adversary can disrupt safety, it is essential to consider possible adversarial actions. In particular, the focus is on an adversary attempting to disrupt the individual safety of a specific block B. By understanding the various strategies the adversary can employ, a comprehensive analysis of the protocol's security can be performed.

6.5 Private attack

In the simple model of the blockchain system with one adversarial user and many honest users, the adversary aims to violate the safety of a specific block B by performing a private attack. The adversary's goal is to first confirm block B by some (or all) honest users and then dislodge it from the longest chain in the future.
One possible attack is the private attack, previously introduced in Lecture 3. Here's a more detailed description of the attack:

1. The adversary identifies block B's parent, denoted as B, and immediately mines a conflicting block on B right after B is mined. The adversary continues mining privately, extending this hidden chain.

2. Meanwhile, the honest users, unaware of the private chain, continue mining on the longest chain below block B.

3. The honest and adversarial chains are independent of each other and follow Poisson processes with rates $(1\beta)\lambda$ and $\beta\lambda$, respectively. (Recall that represents the fraction of hash power controlled by the adversary.)

To successfully perform the private attack and violate the safety of block B, the adversary must reveal its private chain at the right time. If the adversary reveals the private chain while it is still shorter than the longest honest chain, the honest users will simply ignore the adversary's chain, and the attack will have no effect.
Therefore, the adversary must wait until its private chain is at least as long as the honest chain containing block B (see Figure 1). This way, when the adversary reveals its chain, it will create a fork from a block preceding B and eventually produce a chain of equal or longer length than the longest chain containing B.
Timing is critical for the success of the private attack. If the adversary reveals its chain too early, even if it displaces block B, the attack will not lead to a safety violation. The adversary must wait until the honest chain is long enough, at least k deep, for the attack to have the desired impact and violate the safety of block B.

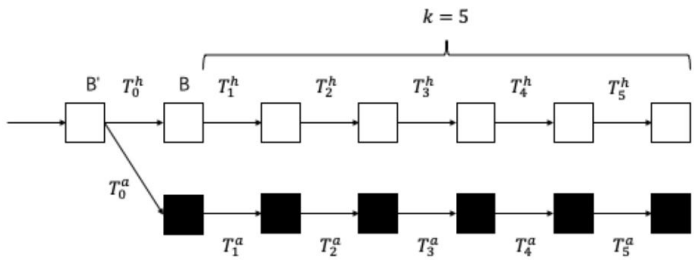

Figure 6.1: Private attack on block B with k = 5.

6.6 Analysis of the private attack under zero delay

Suppose now that the network delay $\delta = 0$. So any node mined successfully by an honest node reaches all other nodes instantaneously. One immediate effect of this is that the honest nodes always have access to the latest mined block and thus can always mine on the tip of the longest public blockchain (by potentially switching their mining upon receiving a new block). Denote the time intervals between the honest and adversarial blocks in the chains of length $k+1$, following block B by $(T_0^h, T_1^h, ...T_k^h)$ and $(T_0^a, T_1^a, ...T_k^a)$ respectively (see Figure 1, where $k = 5$). Then the private attack is successful exactly when the total time to mine $k+1$ blocks by the adversary is faster than the time the honest miners take to mine their $k+1$ blocks, i.e.,

$$\sum_{i=0}^{k} T_i^h \geqslant \sum_{i=0}^{k} T_i^a$$

Note that the times are all independent of each other and $T_0h, ..., T_kh$ are identically distributed as exponential random variables with mean $(\frac{1}{1-\beta\lambda})$ and $T_0^a, ..., T_k^a$ are identically distributed as exponential random variables with mean $\frac{1}{\beta\lambda}$. By the law of large numbers

$$\frac{1}{k+1}\sum_{i=0}^{k} T_i^h - T_i^a - > \frac{1}{(1-\beta)\lambda} - \frac{1}{\beta\lambda}.$$

The limiting value is positive (i.e., the private attack is successful) exactly when

$$\frac{1}{(1-\beta)\lambda} > \frac{1}{\beta\lambda}.$$

or simply $\beta > \frac{1}{2}$, i.e., the adversary controls a majority of the hash power. This calculation shows that in the limit of k very large, the safety of any block B in the longest chain of Bitcoin will continue to remain in the longest chain with probability approaching one. In general there is a tradeoff between a confirmation depth of k and the resulting safety of a block B that has k blocks mined under it. The probability of "deconfirmation", i.e., the block B gets dislodged from the longest chain can be calculated as follows (here $s > 0$ is a parameter to be chosen later):

$$P(\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(s\sum_{i=0}^{k}(T_i^h - T_i^a) > 0) = P(e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)} > 1) \tag{6.1}$$

$$\leqslant E[e^{s\sum_{i=0}^{k}(T_i^h - T_i^a)}] = E[\prod_{i=0}^{k} e^{s(T_i^h - T_i^a)}] = \prod_{i=0}^{k} E[e^{s(T_0^h - T_0^a)}] \tag{6.2}$$

$$= (\frac{\beta\lambda}{\beta\lambda + s})^{k+1}(\frac{(1-\beta)\lambda}{(1-\beta)\lambda - s})^{k+1} \tag{6.3}$$

because of the factorization of the moment generating function of independent random variables. Choosing $s = (1\frac{2\beta\lambda}{2})$ minimizes the exponent and we see that the probability of deconfirmation of a block B is upper bounded

by $e^{c(k+1)}$ where the exponent is given by: $c = \log_e(4\beta(1-\beta)) > 0$. So the probability of deconfrmation decays exponentially in k, as also illustrated in Figure 2. Turns out this calculation was also conducted by Nakamoto themself, albeit somewhat incorrectly, which is included in Figure 2 as a comparison.

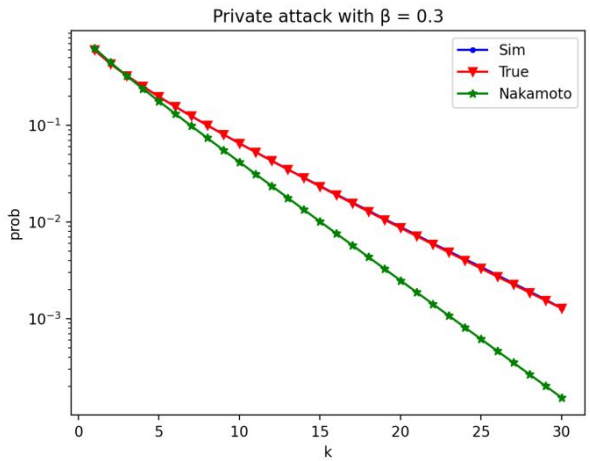

Figure 6.2: Private attack with = 0.3.

6.7 Private attack is the worst-case attack

The balance attack is another type of attack on the safety of the blockchain, where the adversary mines and releases blocks in a way that balances the heights of two chains. This prevents any one chain from stabilizing, causing constant switching between the two chains and preventing the ledger from stabilizing.
In the case of zero network delay (where all miners instantaneously learn of any newly mined block), it has been observed that whenever any attack on safety is successful, the private attack is also successful. In this context, safety refers to the k-deep confirmation rule, which means that blocks deeper than k blocks in the longest chain are considered confirmed. Here are some key observations related to this:

1. To launch a safety attack (a balancing attack), at least two chains of length $k+1$ must be present: one chain to confirm a block B (of depth k) and another chain to de-confirm block B by switching the longest chain.

2. Therefore, the total number of blocks mined by both the adversary (A_k) and the honest nodes (H_k) during the attack must be at least $2k+2 : A_k + H_k 2k + 2$.

3. Due to zero network delay, two honest nodes can never mine at the same level of the blockchain. Honest nodes always mine on the tip of the longest chain, preventing parallel mining at the same level.

4. For two chains to have a length of at least $k+1$ (to launch the safety attack), the number of adversarially mined blocks must be larger than the number of honest mined blocks: $A_k H_k$.

From these observations, it follows that $A_k k + 1$, which means that the adversary has successfully mined at least $k+1$ blocks by the time of the safety attack on the k-deep confirmation rule.
The significance of this result is that, in the case of zero network delay, any attack that successfully violates safety (such as a balance attack) requires the adversary to have already mined at least $k+1$ blocks. And with this number of blocks, the adversary could have used them to launch a private attack instead.
This observation is crucial in understanding the relationship between different attack strategies and provides insights into the security of the Nakamoto consensus protocol under various conditions. It shows that the private attack is a powerful and general attack strategy, and other attacks, like the balance attack, are limited in their effectiveness compared to the private attack (Figure 3).

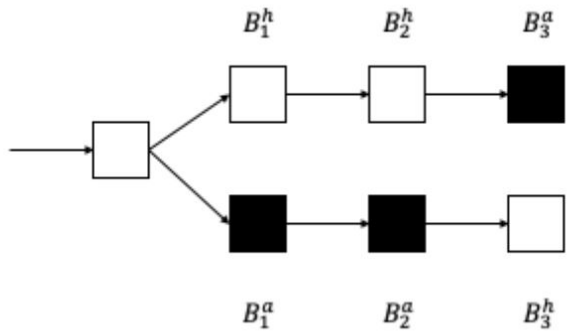

Figure 6.3: Illustration of a balance attack.

6.8 Safety analysis under bounded network delay

In the δ-synchronous network model with propagation delay, there is a finite amount of time δ within which all blocks reach all participants definitively. The adversary can deliver blocks to different participants at a time of its choosing, as long as the δ-synchrony condition is met. This introduces natural forking in the blockchain, even without adversarial intervention.

In the context of Bitcoin, with an inter-block arrival time of ten minutes, natural forking is not very prevalent because it is unlikely that honest nodes will mine on stale blocks. However, in other blockchains like Ethereum, where the inter-block arrival time is fourteen seconds, natural forking is more common due to the shorter block time.

The term $(1-\beta)\lambda\delta$ represents the expected number of blocks that are being mined "in parallel" by the honest nodes. Here, $(1-\beta)\lambda$ is the rate at which honest blocks are mined, and δ is the network delay. This means that during the time λ, the honest nodes may mine multiple blocks, but only one of these blocks will succeed in extending the longest chain. The other blocks mined in parallel are considered "wasted" because they will not become part of the final longest chain.

For example, if the honest fraction of hash power $(1-\beta)\lambda$ is 0.9 and the network delay δ is 10 seconds, then the expected number of parallel blocks mined by honest nodes during this time is 0.9 blocks. Out of these 0.9 blocks, only one block will eventually be included in the longest chain, and the remaining $0.9-1=-0.1$ block is considered "wasted" mining effort.

In blockchains with shorter block times and higher natural forking rates, a larger fraction of honest mining power may be "wasted" due to natural forking. This natural forking phenomenon is an inherent characteristic of the Nakamoto consensus protocol in blockchain systems and affects the efficiency of the mining process.

Thus the rate of growth of the honest chain is now reduced by a fraction $\frac{1}{1+(1-\beta)\lambda\delta}$ to

$$\frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\beta}$$

Now an adversary launching a private attack is successful if its rate of growth (of a private chain) is larger than that of the (reduced) growth rate of the honest chain:

$$\beta\lambda > \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\beta} \tag{6.4}$$

Figure 4 illustrates the minimum hash power the adversary needs to succeed in its private attack, i.e., to replace a certain number of blocks on the longest chain with its own longer chain. The adversary can execute the private attack by mining blocks in secret and then revealing them at a strategic moment to create a longer chain that overtakes the honest chain.

Previously, we saw that when the network delay (δ) is zero, the private attack is a worst-case attack in a very general setting. This means that for every sample path of the mining process and for every confirmation depth (k), the private attack is successful whenever any attack is successful. However, in cases where the network delay

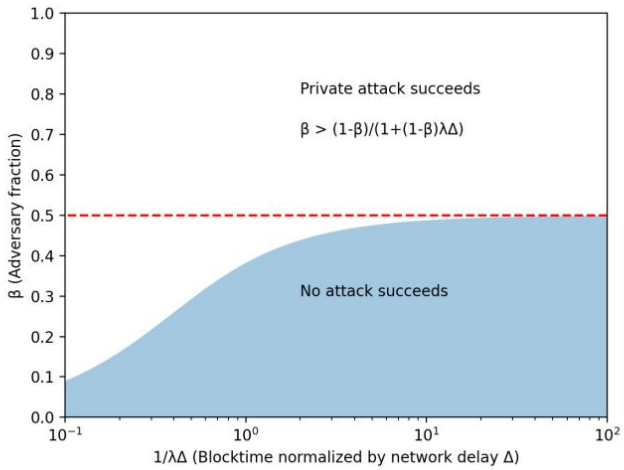

Figure 6.4: Minimum hash power needed by the adversary to succeed in its private attack. It turns out that this is also the security threshold for the adversary to successfully launch any safety attack.

is non-zero ($\delta > 0$), the private attack may not always be the worst-case attack.
Figure 5 shows an example where the private attack is not the worst-case attack. In this scenario, the adversary can use a different strategy and still succeed in disrupting the safety of the blockchain, even though the private attack fails. This implies that the private attack is not always the most effective attack in the presence of network delay.
However, despite this, it turns out that the private attack is still a worst-case attack in the sense of the minimum required honest hash power to ensure safety against all attacks. In other words, if the blockchain system can withstand the private attack with a certain level of honest hash power, it can also withstand any other attack with the same level of honest hash power. This safety guarantee holds with probability approaching one as the confirmation depth (k) approaches infinity.
The reason why the private attack is still a worst-case attack in terms of the required honest hash power is due to the inherent properties of the Nakamoto consensus protocol. The protocol is designed such that an attacker needs to have a majority of the total hash power to overpower the honest nodes and control the longest chain. Therefore, the same condition derived for the success of the private attack also holds for any other attack, and the private attack remains a worst-case scenario for determining the minimum required honest hash power to ensure safety in the blockchain system.

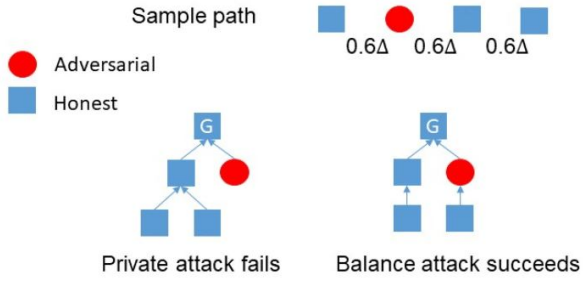

Figure 6.5: Attacks aimed to create a fork with length 2

6.9 Nakamoto blocks

The analysis of attacks on the blockchain system becomes more complicated when the network delay (δ) is non-zero, and the entire blocktree can be arbitrary due to multiple adversarial actions at different time instances.

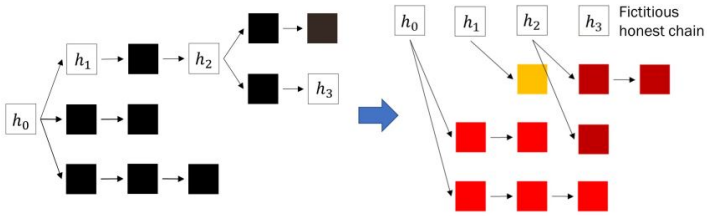

Figure 6.6: By blocktree partitioning, a general attack is represented as multiple adversarial chains simultaneously racing with a fictitious honest chain. Note that this fictitious chain is formed by only the honest blocks, and may not correspond to the longest chain in the actual system. However, the longest chain in the actual system must grow no slower than this fictitious chain.

However, we can still gain insights by partitioning the complex blocktree into subtrees, each rooted at an honest block and consisting entirely of adversarial blocks (public or private).
By doing this, we can view the general attack as initiating multiple adversarial subtrees that race with a single fictitious chain consisting of only honest blocks (Figure 6). Each adversarial subtree's growth rate is upper-bounded by the growth rate of the adversarial chain used in the private attack. Therefore, if the private attack is unsuccessful, we know that the growth rate of each adversarial subtree must be less than that of the fictitious honest chain.
Under this condition, there must exist honest blocks, referred to as Nakamoto blocks, that have the special property that none of the past adversarial subtrees can ever catch up to the honest chain after the honest chain reaches that particular block (Figure 7). These Nakamoto blocks play a crucial role in stabilizing the blockchain. When a Nakamoto block occurs and occurs frequently, it guarantees the safety of the protocol.
The presence of Nakamoto blocks ensures that the prefix of the longest chain up to that block remains immutable in the future. In other words, once a Nakamoto block is included in the blockchain, it becomes a solid anchor that protects the entire prefix of the longest chain before that block from any adversarial reorganization attempts. This stability ensures that the blockchain system maintains its safety guarantees, and honest users can have confidence in the integrity of the ledger.
When Nakamoto blocks occur regularly, the blockchain's safety is effectively guaranteed, even in the presence of adversarial attacks. The analysis helps us understand how certain key blocks in the blockchain, such as Nakamoto blocks, play a critical role in maintaining the system's security and ensuring that honest users' transactions remain protected and valid.

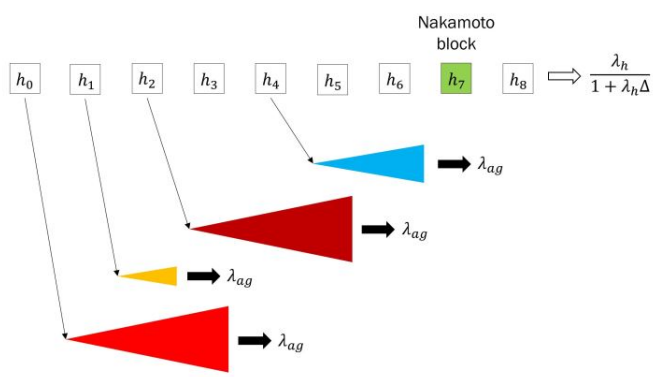

Figure 6.7: Race between the adversarial trees and the fctitious honest chain. While there may be multiple adversarial trees simultaneously racing with the honest chain, the growth rate of each tree is bounded by the growth rate of the adversarial chain in the private attack. An honest block is a Nakamoto block when all the previous adversarial trees never catch up with the honest chain past that block. To simplify notations, $\lambda_{ag} = \beta\lambda$ and $\lambda h = (1\beta)\lambda$.

6.10 Importance of synchronous network

The safety analysis of the longest chain protocol in this lecture highlights the importance of keeping the adversarial hash power within certain bounds relative to the overall mining rate and worst-case network delay. As long as the adversarial hash power is limited, the protocol remains safe, ensuring that the blockchain maintains its integrity and security.

However, the analysis also emphasizes the significance of having a finite worst-case network delay. In a purely asynchronous network, where there are no guarantees on message delivery times, the adversary can exploit the lack of synchronized communication to break consensus. By mining alternate chains and selectively sharing them with different subsets of honest nodes, the adversary can create confusion and disagreement among the nodes, leading to a breakdown of consensus and safety.

To address this issue, an intermediate network scenario, known as the partially synchronous network model, is considered. In this setting, the network delays are finite but unknown. There is a global stabilization time (GST), after which all messages are guaranteed to be delivered to all nodes within bounded time. In this partially synchronous setting, safety can potentially be violated because the confirmation depth k (used in the k-deep confirmation rule) could be smaller than the GST, allowing for vulnerabilities.

However, it is crucial to note that after the GST, consensus is restored, and the blockchain system becomes safe once again. This highlights the trade-off between the guarantees of safety and the network conditions in such scenarios.

In subsequent lectures, the focus will shift to studying consensus protocols that can ensure safety even under partially synchronous network settings, addressing the challenges posed by the unknown and varying network delays. These protocols aim to provide robustness and security even in more realistic network conditions, ensuring the integrity of the blockchain system under various scenarios.

Chapter 7

Liveness of Bitcoin

The last lecture focused on the safety of the longest chain protocol, which is a rule that determines which block is the valid one in case of a fork. The safety property means that once a block is confirmed by being k-deep in the chain, then the probability of it being deconfirmed by another fork is very small, as long as k is large enough. This ensures that the blockchain does not have conflicting histories, and that users can trust the transactions that are recorded in it.

We are interested in exploring the scenarios where the ledger does not receive any new blocks, or where some transactions from honest users are excluded from the blocks.

7.1 What is Liveness

Liveness is a term that describes the ability of a blockchain protocol to ensure that the transactions that are submitted by honest users are eventually included in the ledger, and that the blocks that contain these transactions are part of the longest chain. Liveness is an important security property because it guarantees that the network will not stop producing new blocks and confirming transactions, and that users can use the system without delays or failures. Liveness also prevents the network from being stuck in a fork, where different nodes have different views of the ledger.

Both liveness and safety are essential for a blockchain protocol to function properly and securely. A good consensus protocol should achieve both liveness and safety under reasonable assumptions about the network and the participants. However, there is often a trade-off between liveness and safety, as increasing one may decrease the other. For example, requiring more confirmations for a block to be considered final may increase safety, but it may also increase latency and reduce liveness. Therefore, finding the optimal balance between liveness and safety is a challenging task for blockchain protocol designers.

Liveness is definitely compromised if there is a deadlock. A deadlock occurs when the protocol gets stuck and the miners and parties cannot perform their tasks and mine new blocks.
Longest chain protocol cannot deadlock, which means that it will always produce a unique longest chain, even if there are multiple forks. This ensures that the network can always reach a consensus on the state of the ledger, and that users can always find the valid block.
The mining operation is very democratic, which means that any honest miner with a small amount of hash power can eventually succeed in mining a block and adding it to the chain. This encourages participation and decentralization in the network, and prevents monopoly or censorship by powerful miners.

There are two adversarial events that can threaten liveness:

1. All blocks on the longest chain are mined by an adversary
2. An adversary mines a block that is empty or censors specific transactions.

We need to examine Chain Quality, which is based on Chain Growth, to determine the likelihood of an honest miner's block being left out of the longest chain or under what circumstances this occurs.

7.2 Chain Growth

Definition 7.2.1: Chain Growth

Chain growth is defined as the rate of growth of the longest chain, denoted by CG.

If there is no fork in the chain, and only the longest chain exists, then its growth rate depends on the difficulty level, and it adds one block every 10 minutes on average. But it slows down for two reasons :

1. Not all miners may be behaving safely.
2. It could be that they're working in parallel

The chain keeps growing and new blocks are continuously added to the longest chain because of positive chain growth. An honest miner has a chance to mine a block, even if it is small, because the mining operation is random.

Chain growth depends on the behavior of the adversary, who controls a fraction of the hash power in the network, denoted by β. If the adversary stays silent, which means that he does not mine or broadcast any blocks, then chain growth is equal to the rate of block production by the honest nodes :

$$CG = \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta}$$

One can see that the adversary really cannot do anything better than this. Delivering an honest block earlier than Δ time or publishing any adversarial block will just increase the chain growth. Therefore, we have the bound:

$$CG \geq \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta} \tag{7.1}$$

If the adversary acts honest, which means that he mines and broadcasts blocks according to the protocol rules, then chain growth is equal to :

$$CG = \frac{(1-\beta)\lambda}{1+(1-\beta)\lambda\Delta} + \beta\lambda$$

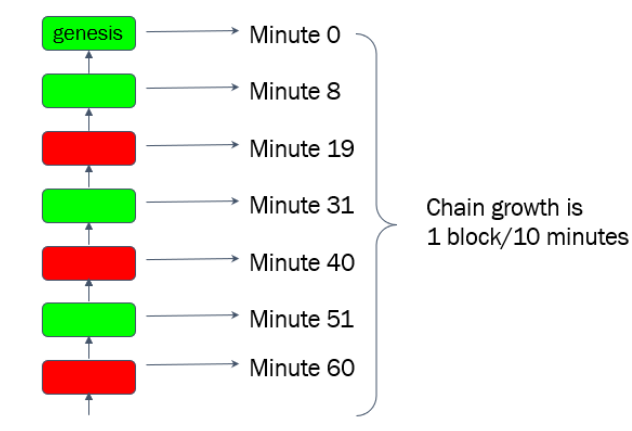

Figure 7.1: Chain Growth

7.3 Chain Quality

Definition 7.3.1: Chain Quality

We define the chain quality of a longest chain C as the fraction of honest blocks in C, denoted as CQ

$$CQ = \frac{\text{\# of honest blocks in the longest chain}}{\text{\# of all blocks in the longest chain}}$$

Positive chain quality ensures that a positive fraction of blocks mined by honest nodes enter the longest chain. Having enough blocks mined by honest nodes on the longest chain (i.e., high CQ) ensures that transactions enter the blockchain at a fast clip – thus CQ helps quantify the level of liveness.

Furthermore, CQ quantifies fairness: since block rewards are provided to blocks in the longest chain, having blocks mined by honest nodes in the longest chain ensures that honest miners are rewarded.

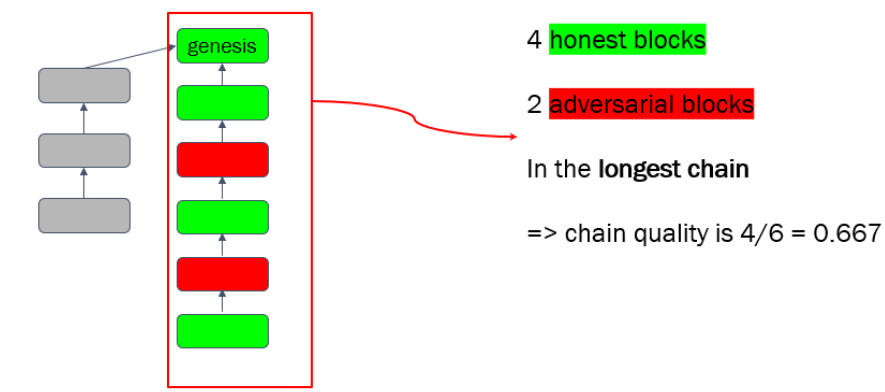

Figure 7.2: Chain Quality

Positive chain growth and positive chain quality together combine to ensure that any honest transaction will eventually be added to a block on the longest chain and to the ledger, which gives us liveness.

If the chain growth is CG and the adversary can mine blocks at a rate of at most $\beta\lambda$, by the definition of the chain quality, we have a lower bound:

$$CQ \geq \frac{CG \times T - \beta\lambda T}{CG \times T} = \frac{CG - \beta\lambda}{CG} \tag{7.2}$$

Plugging 7.1 in 7.2 :

$$CQ > 0 \Leftrightarrow \frac{(1-\beta)\lambda}{1 + (1-\beta)\lambda\Delta} > \beta\lambda$$

This is exactly the same threshold as the safety guarantee as we have seen before. Therefore, we can conclude the following security guarantee on the longest chain protocol.

Theorem 7.3.1

The longest chain protocol is safe and live, exactly under the following condition:

$$\frac{1-\beta}{1 + (1-\beta)\Delta} > \beta$$

7.4 Selfish Mining

The ideal chain quality is when the number of honest blocks on the longest chain is proportional to the honest hash power, which is $CQ = 1 - \beta$. However, this does not happen in the longest chain protocol because of **selfish mining**, which is a strategy where an adversary tries to create a private fork and withhold it from the network until he can overtake the public chain.

Consider the following adversarial strategy :

- The adversary always mines on the block at the tip of the longest chain, whether the chain is private or public. Upon successful mining, the adversary maintains the block in private to release it at an appropriate time.
- When an honest miner publishes a block the adversary will release a previously mined block at the same level (if it has one). We assume that the adversary can break ties in its favor, so honest miners will mine on the adversarial block.

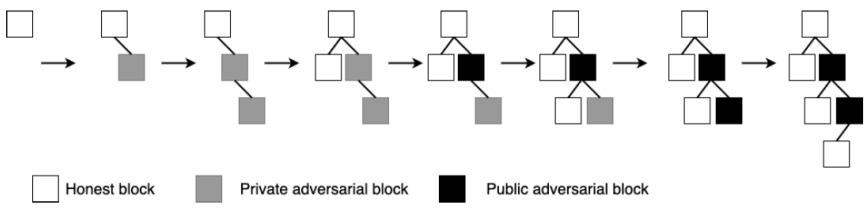

Figure 7.3: Selfish mining example

Selfish mining reduces the chain quality and liveness of the protocol, and gives an unfair advantage to the adversary.

7.5 Fruitchains

The longest chain protocol would have the best possible chain quality, $CQ = 1 - \beta$, if the adversary followed the protocol rules. But what if the adversary tries to cheat and disrupt the protocol? Can we design a better protocol that can achieve the optimal chain quality no matter what the adversary does? And can we do it with only a minor change to the longest chain protocol?
Fruitchains is a protocol that does exactly that. It achieves the optimal chain quality, and it does it fast.

The longest chain protocol links the security and the rewards of a block. The more blocks are built on top of a block, the safer and more profitable it is. Fruitchains is a new protocol that separates these two aspects by using two different types of blocks:

- **transaction block**, which has all the transactions (or data) that a block in the longest chain protocol would have.
- **proposer block**, which has only the header information that a block in the longest chain protocol would have.

These two types of blocks need to be linked and coupled together in a secure way. This is done by putting the hash of each transaction block inside a proposer block (see Figure **??**). The two types of blocks are also connected during the mining process by using "2 for 1 mining", also known as **cryptographic sortition**.

7.5.1 Cryptographic Sortition

In the mining process, the two types of blocks – transaction and proposer blocks – are mined together. A miner first creates a **superblock**, which is similar to a normal block in the longest chain protocol. The superblock has

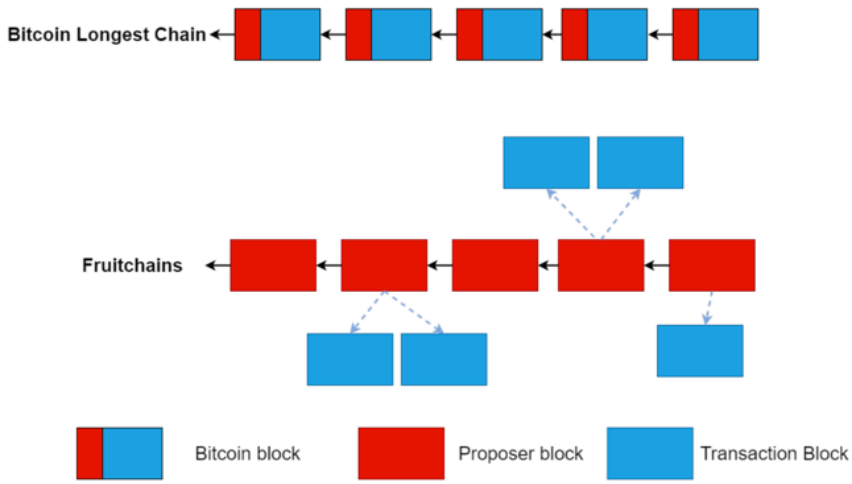

Figure 7.4: Comparison between blocks in the longest chain protocol and the transaction and proposer blocks in Fruitchains.

two parts: a transaction block and a proposer block. But that's where the similarity ends.
We say a superblock is successfully mined if

$$Hash(\text{nonce}, \text{superblock}) < T_{tx} + T_{prop}$$

The mining process determines whether a superblock is a transaction block or a proposer block based on the hash output:

- If the hash output is smaller than T_{prop}, the superblock is a proposer block.
- If the hash output is between T_{prop} and $T_{tx} + T_{prop}$, the superblock is a transaction block.

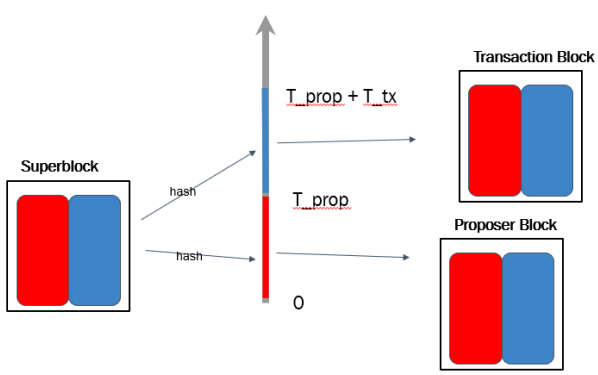

Figure 7.5: Superblock

The hash functions produce random outputs, so the chance of a superblock being a proposer or a transaction block depends on T_{prop} or T_{tx}, respectively. This means that two different kinds of blocks are mined together, but only one of them is chosen as the result of the mining process. This is called "two-for-one mining" or "cryptographic sortition".

7.5.2 Fruitchain Protocol

The ledger is a list of transactions that are sorted by some rules. The rules are:

- First, sort the proposer blocks by using the longest chain rule, which means that the valid chain is the one with the most blocks.
- Second, sort the transaction blocks that each proposer block refers to. For example, you can sort them by their hash value, which is a unique code for each block.
- Third, sort the transactions in each transaction block. For example, you can sort them by using the lexicographic ordering of the transactions in the Merkle tree representation, which is a way of organizing the transactions in a tree shape.

Safety of fruitchain protocol

Assuming λ_{tx} and λ_{prop} as the mining rate of the transaction blocks and proposer blocks respectively; these mining rates are directly proportional to the difficulty targets T_{tx} and T_{prop}, respectively. The safety of this protocol follows directly from the safety of the longest chain protocol: as long as λ_{prop} is small enough, i.e.,

$$\frac{1-\beta}{1+(1-\beta)\lambda_{prop}\Delta} > \beta$$

the k-deep proposer block in the longest chain will be permanent with high probability (for large enough k). This stabilizes the ledger guaranteeing safety.

Liveness of fruitchain protocol

We want to show that the optimal CQ of 1β is achieved no matter what the adversary does to interfere with the transaction and proposer block mining. We start by looking at the selfish mining attack: suppose an adversary tries to hide some proposer blocks that were mined by an honest party (and that contain some honest transaction blocks) by mining on a secret branch. Then, by the liveness property of the longest chain protocol (which guarantees positive CQ and CG), sooner or later an honest party will mine a new proposer block that includes those displaced transaction blocks. And if k is large enough, this proposer block will be securely added to the ledger.

We know that the honest and adversarial parties mine transaction blocks according to Poisson processes with rates $(1\beta)\lambda_{tx}$ and $\beta\lambda_{tx}$ respectively. And by the liveness property of the longest chain protocol, every honest transaction block will eventually be part of the proposer chain.Therefore, by considering an average on the entire longest proposer chain, we have

$$CQ \geq \frac{(1-\beta)\lambda_{tx}}{(1-\beta)\lambda_{tx}+\beta\lambda_{tx}} = 1-\beta$$

Note that here we define chain quality as the fraction of honest transaction blocks instead of honest proposer blocks, and further determining mining rewards by transaction block yields fair rewards.

7.5.3 Short time scale optimal CQ

The protocol is not secure in a short time span, because a private block attacker could secretly mine many transaction blocks and release them all at once, making a large portion of the proposer chain consist of adversarial transaction blocks.

This issue is resolved by making sure that a transaction block is attached to a proposer block that is close enough to the proposer block that includes it. Each transaction block has two parent blocks:

- A confirmed parent, which is a proposer block that has been stabilized/confirmed (i.e., k-deep block) and that the transaction block hangs from.
- A proposer parent, which should be the tip of the longest proposer chain.

Note that a proposer block also has a confirmed parent, because the two-for-one mining process links the transaction block mining and the proposer block mining.

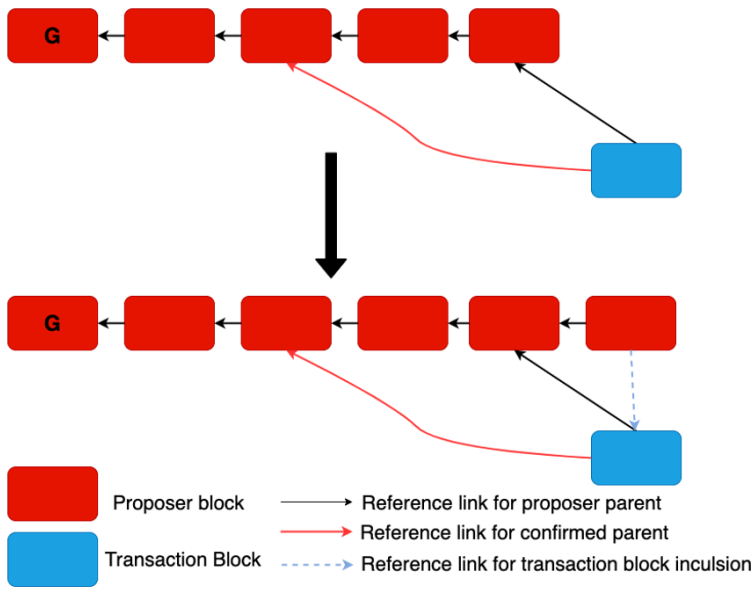

Figure 7.6: Fruitchain protocol

A transaction block B_{tx} is recent with respect to a proposer chain C if its confirmed parent is not too far from the tip of C. The distance is measured by the recency parameter R , which is a fixed number. We achieve our goal by only allowing proposer blocks to include recent transaction blocks. By choosing a large enough R, we make sure that any transaction block mined by an honest party will be deep enough in the proposer chain. So, the optimal CQ property still holds. Moreover, since only fresh transaction blocks can be included, the optimal CQ is achieved even for short parts of the longest chain.

Chapter 8

Scaling Latency

8.1 Bitcoin Latency

In Bitcoin, when a transaction has been included in a block, it will be confirmed only after the block is buried k-deep in the longest chain. Furthermore, to ensure confirmation with high probability, k must be large. The average latency of confirmation is then the product of k and average inter-block arrival time; again just like the throughput, the latency in Bitcoin is also limited by security. How bad is this trade off? in summary we have:

- **Confirmation in Bitcoin:** In Bitcoin, a transaction is confirmed when it is included in a block, which is then buried k-deep in the longest chain.
- **Latency of Confirmation:** The average latency of confirmation is the product of k and the average inter-block arrival time.
- **Trade-off with Security:** To ensure high probability of confirmation, k must be large. However, this trade-off is limited by security concerns.
- **Analysis with Zero Network Delay:** The analysis begins with the assumption of zero network delay ($\delta = 0$).
- **Error Probability (ϵ):** The largest probability of deconfirmation is achieved by the private attack. The error probability (ϵ) decays exponentially with the depth of the confirmation rule (k).
- **Exponential Decay of Error Probability:** The error probability (ϵ) is given by $\epsilon = e^{-ck}$, where $c = -log_e(4\beta(1-\beta))$.
- **Parameters:** Here, represents the fraction of the hash power of the adversary.

This analysis demonstrates that the error probability decreases exponentially with an increase in the confirmation depth (k) and is influenced by the fraction of hash power controlled by the adversary (β). It illustrates the security implications and limitations of setting the confirmation depth for transactions in the Bitcoin blockchain. In summary, the latency of Bitcoin's confirmation process is influenced by the error probability ( ) and the fraction of hash power controlled by the adversary ( ). The formula for the latency (τ_{LC}) is given as:

$$\tau_{LC} = \frac{k}{(1-\beta)\lambda} = \frac{1}{c\lambda(1-\beta)ln(\frac{1}{\epsilon})} = O(\frac{1}{\lambda ln(\frac{1}{\epsilon})})$$

where the constant hidden in the $O(\mathring{u})$ only depends on β. The latency increases as the error probability decreases, emphasizing the trade-off between security and confirmation time. For example, with $\beta = 0.3$ and $\epsilon = 10^{-3}$, the latency (τ_{LC}) works out to be approximately $\frac{66}{\lambda}$, which translates to around 11 hours when the average inter-block time ($\frac{1}{\lambda}$) in Bitcoin is 10 minutes.
The plot in Figure 1 illustrates this trade-off between latency and security, showing that even for modest security requirements, the latency can be significant in Bitcoin due to the potential for private attacks, which are the worst-case attacks for zero network delay ($\delta = 0$).

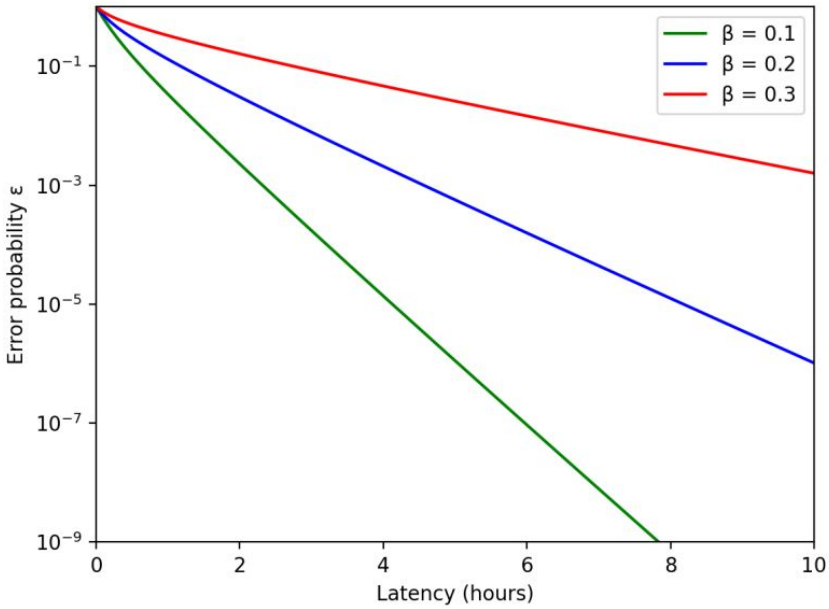

Figure 8.1: Bitcoin's latency–security trade-off for different β assuming $\delta = 0$. We set λ to be 6 blocks per hour, as in Bitcoin.

In the general case when $\delta > 0$, the private attack is no longer the worst-case attack in terms of success probability. However, it still provides a lower bound on the error probability, which consequently sets a lower bound on the latency for a given error probability. Both the error probability and latency tend to increase as δ increases due to increased forking caused by propagation delays in the network.

From a practical perspective, what is more interesting are the upper bounds on the error probability and latency, as they provide practical guidance for users of Bitcoin to determine when a transaction can be considered confirmed with a desired level of confidence. Recent research has derived such upper bounds, which are fairly close to the lower bounds implied by the private attack.

For example, consider a scenario where the adversary controls 10% of the total mining power and the block propagation delays are within $\delta = 10$ seconds. With these settings, a Bitcoin block can be confirmed with less than 10^{-3} error probability after approximately 5 hours and 20 minutes. For an even higher level of confidence, a block can be confirmed with less than 10^{-10} error probability after approximately 12 hours and 15 minutes.

Figure 2 illustrates the security-latency trade-off as implied by both the lower and upper bounds. The plot shows how the error probability decreases with increasing latency and how the upper and lower bounds are relatively close, providing a reasonable estimate of the confirmation time for a desired level of security.

8.2 Prism: Fast Confrmation

Lecture 8 introduces Prism 1.0, a blockchain protocol with two types of blocks: transaction blocks and proposer blocks. These blocks separate the security and payload aspects of the longest chain protocol. The proposer blocks serve two roles: they act as leaders for the "round" and provide confidence to ancestor blocks through the k-deep confirmation rule. Additionally, they vote for all ancestor blocks via parent link relationships. The full Prism protocol further decouples voting from proposer blocks by introducing separate voter blocks. The proposer blocktree anchors the blockchain, and each proposer block contains reference links to transaction blocks and a reference to its parent proposer block. Honest nodes mine proposer blocks following the longest chain rule in the proposer tree. The level of a proposer block represents its distance from the genesis proposer block, while the height of the proposer tree is the maximum level containing any proposer blocks. To determine the ordering of

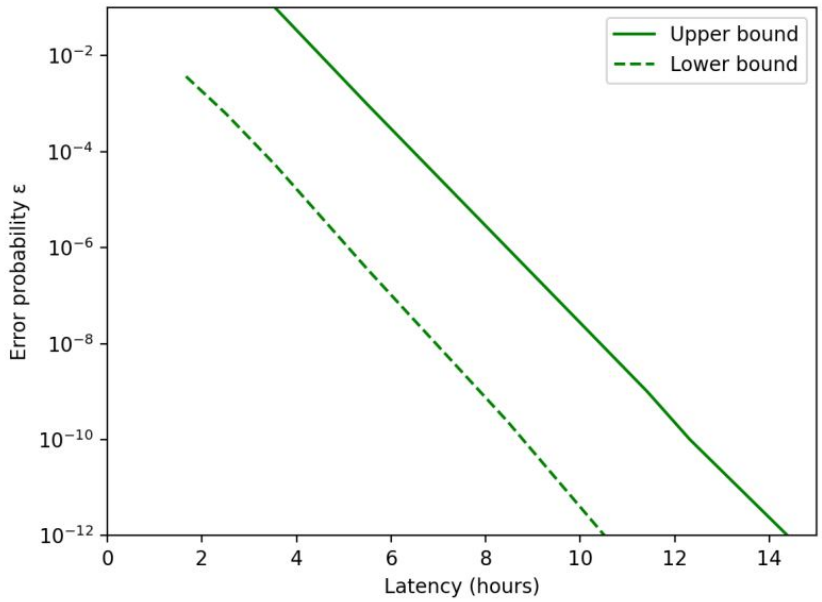

Figure 8.2: Bitcoin's latency–security trade-off for $\beta = 0.1$ when $\delta > 0$. We set λ to be 6 blocks per hour as in Bitcoin and $\delta = 10$ seconds. The lower bound on latency is derived from the private attack, while the upper bound is borrowed from this work.

proposer blocks and transactions, one leader proposer block is elected from each level, forming the leader sequence up to the height of the proposer tree, determined by the voter chains.

- Prism 1.0 introduces two types of blocks: transaction blocks and proposer blocks.
- These blocks decouple the security and payload aspects of the longest chain protocol.
- Proposer blocks have two roles: proposing as leaders for the round and adding confidence to ancestor blocks through the k-deep confirmation rule.
- Proposer blocks also vote for all ancestor blocks through parent link relationships.
- The full Prism protocol further separates voting from proposer blocks by introducing separate voter blocks.
- The proposer blocktree anchors the blockchain and contains reference links to transaction blocks and a parent proposer block reference.
- Honest nodes follow the longest chain rule to mine proposer blocks in the proposer tree.
- The level of a proposer block represents its distance from the genesis proposer block, and the height of the proposer tree is the maximum level with proposer blocks.
- One leader proposer block is elected from each level to determine the ordering of proposer blocks and transactions.
- The leader sequence, determined by voter chains, comprises leader proposer blocks up to the height of the proposer tree, independent of the chain structure of proposer blocks to avoid deadlocks.

In Prism, there are m voter chains, where $m1$ is a fixed parameter set by the system designer. The value of m determines the level of parallelism in the voting process, and a larger m leads to shorter confirmation latencies. The choice of m is typically limited by network bandwidth and memory management considerations. For example,

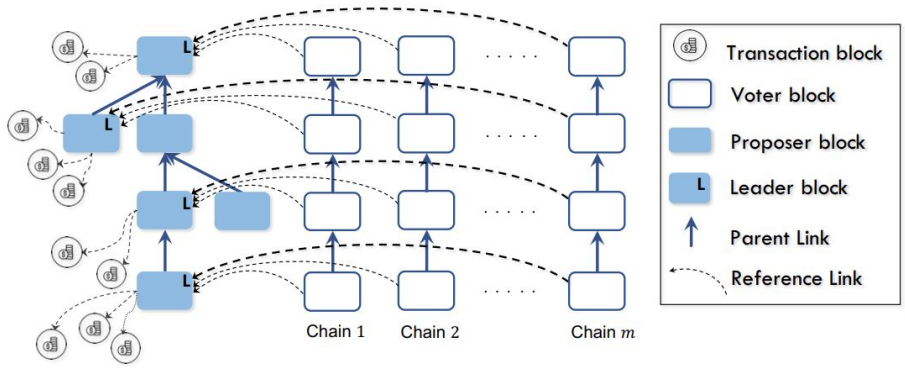

Figure 8.3: Factorizing the blocks into three types of blocks: proposer blocks, transaction blocks and voter blocks.

a full-stack implementation of Prism may use $m = 1000$.
New voter blocks are continuously mined on each voter chain, following the longest chain rule. Each voter block serves as a vote for a proposer block by containing a reference link to that proposer block. The voting process has specific requirements:

1. A vote is valid only if the voter block is part of the longest chain in its respective voter tree.
2. Each voter chain votes for exactly one proposer block at each level.
3. Each voter block votes for all proposer levels that have not been voted by its parent.

The leader block at each level is determined by the proposer block that receives the highest number of votes among all proposer blocks at that level. In case of ties, the leader is chosen based on the hash of the proposer blocks. The elected leader blocks establish a unique ordering of the transaction blocks, forming the final ledger of the Prism blockchain.
In Prism, cryptographic sortition is employed to prevent the adversary from concentrating its mining power on specific types of blocks or voter chains. To form a "superblock," a miner combines three components: a transaction block, a proposer block, and a voter block from the i-th voter tree, where $1im$. A superblock is considered successfully mined if its hash value, computed using a nonce, is less than the threshold value $T_{tx} + T_{prob} + mT_v$. Once a superblock is successfully mined, it is identified as one of the following based on its hash output. (See Figure 4 for an illustration):

- A proposer block if the hash output is less than T_{prop}.
- A transaction block if the hash output falls within the range $[T_{prob}, T_{tx} + T_{prob})$.
- A voter block on the i-th voter tree if the hash output falls within the range $[T_{tx}+T_{prob}+(i1)T_v, T_{tx}+T_{prob}+iT_v)$, where $1im$.

As we said above, Prism is a blockchain protocol that decouples the security and payload aspects of the longest chain protocol. It introduces two types of blocks: transaction blocks and proposer blocks, which act as leaders and voters in the security aspect. Additionally, there are m voter chains to parallelize the voting process, and cryptographic sortition prevents the adversary from concentrating mining power on specific blocks or voter chains. The protocol ensures a unique ordering of transaction blocks and enables ledger sanitization after the ledger has stabilized.

More details are listed below:

1. Prism employs cryptographic sortition to prevent the adversary from focusing its mining power on specific blocks or voter chains.

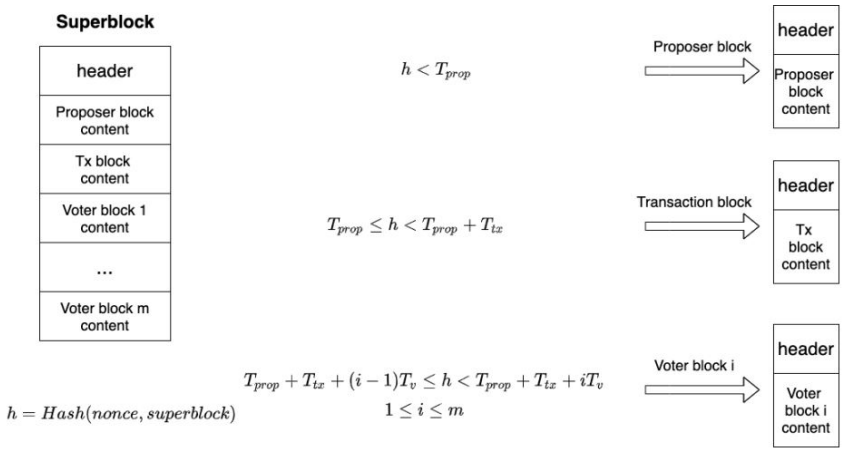

Figure 8.4: Superblock containing a transaction block, a proposer block and m voter blocks is "$(m+2)$-for-one" mined; the resulting cryptographic sortition identifes the type of the block as the winner of the mining process.

2. Superblocks are formed by combining a transaction block, a proposer block, and a voter block from the i-th voter tree $(1im)$.
3. The successful mining of a superblock is determined by the hash output being less than specific threshold values T_{prob}, T_{tx}, and T_v.
4. The probability of a superblock being identified as a proposer block, a transaction block, or a voter block on the i-th voter tree is proportional to T_{prob}, T_{tx}, and T_v, respectively.
5. The transaction target difficulty (T_{tx}) is set easy, allowing for parallel mining of transaction blocks, while the proposer and voter chain target difficulties (T_{prop} and T_v) are set high for security.
6. The Prism protocol decouples the process of mining blocks from validating transactions, as the miner cannot validate a block before mining due to the non-chain structure of proposer blocks.
7. After a confrmed leader sequence of proposer blocks is established, the ordering of transaction blocks becomes uniquely determined, enabling ledger sanitization and validating individual transactions.

Overall, Prism combines parallelism, security, and decoupling to achieve efficient and secure transaction processing in a blockchain protocol.

8.3 Fast confrmation rule in Prism

In summary, Prism's parallel voting scheme allows for faster confirmation latency. By leveraging the law of large numbers, votes can be considered even if they are not fully stabilized, providing security while minimizing latency. However, these techniques need to be carefully analyzed to ensure the protocol's robustness against attacks. More details are listed below:

1. **Parallel Voting Scheme in Prism:** The parallel chain structure of the voting scheme in Prism allows for low confirmation latency. Votes accrue in parallel, unlike Bitcoin's sequential voting approach, leading to faster confirmation times for a fixed level of security.
2. **Simple Case of Confrmation Latency:** Let's consider a simple case in Prism where a single honest proposer block (B_p) at level one is mined at time 0. In Prism, B_p can be confirmed if no other proposer block at level one receives more votes than it. However, this naive solution is not operational since the confirmation depends on the future.

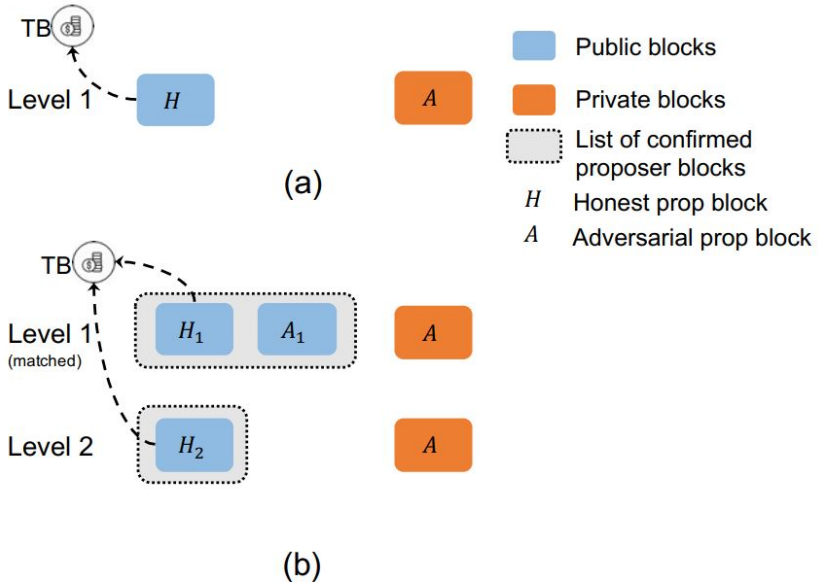

Figure 8.5: (a) Transaction block is referred to by an isolated honest proposer block. (b) Transaction block is referred to by a non-isolated proposer block but on the next level there is an isolated proposer block. Note that since H_2 is honest, it refers to all unconfrmed transaction blocks, i.e., TB.

3. **Principled Confirmation Rule:** A principled confirmation rule that balances security and fast latency is needed. Waiting until each vote stabilizes by being k-deep in their chains (similar to Bitcoin) results in high latency. However, it's not necessary to wait for each vote to stabilize fully.

4. **Leveraging the Law of Large Numbers:** Even if each vote is ephemeral (with a small value of k, e.g., $k = 2$ or 3), the averaging of all votes ensures overall security. This phenomenon is analogous to the law of large numbers or boosting in machine learning, where combining many low-precision classifiers creates a high-precision classifier.

5. **Analysis of a Simple Attack:** Consider a scenario where an honest proposer block H refers to a transaction block TB and is currently isolated at its level, meaning no other public proposer block exists at the same level. Block H starts collecting votes, each from the longest chain of its respective voter tree (See Figure 5(a)). The adversary attempts a private attack by mining a private proposer block A at the same level and forking off private alternate chains on each voter tree. It then sends its vote to block A. The adversary continues to mine on each voter alternate chain to overtake the public longest chain and shift the vote from H to A. The attack is successful if the adversary gets more votes on A than on H.

8.3.1 Fast confrmation rule

The use of multiple voter chains in Prism reduces confirmation latency while maintaining security. By leveraging Nakamoto's calculations and employing averaging principles, the protocol achieves faster confirmation times without compromising on robustness against attacks. In summary we can say:

1. **Attack Analogy to Nakamoto's Private Attack:** The attack in Prism, with m chains, can be seen as the multi-chain analog of Nakamoto's private attack on Bitcoin. Instead of a single race between the honest chain and the private chain, we have m races in Prism.

2. **Using Nakamoto's Calculations for Determining Waiting Depth:** Nakamoto's calculations on the success probability of an attack on a single chain can help determine how deep we should wait for votes to confirm the proposer block H in Prism. For tolerable adversary power ($\beta = 0.3$), the reversal probability in a single chain is 0.45 when a block is 2-deep.

3. **Latency Reduction with Multiple Voter Chains:** With $m = 1000$ voter chains and each vote being 2-deep, the expected number of chains that can be reversed by the adversary is 450. The probability that the adversary can reverse more than half the votes (500) is about 10^{-3}. Thus, to achieve $\epsilon = 10^{-3}$, we only need to wait for 1000 votes each 2-deep, resulting in much shorter latency than waiting for 24 block depths for each vote to be reversed with a probability of 10^{-3}.

4. **Conceptual Similarity to Averaging Classifiers:** The reduction in latency in Prism is conceptually similar to averaging many unreliable classifiers to form a strong aggregate classifier. With more voter chains, individual votes need less certainty of permanence, leading to reduced confirmation time without sacrificing security.

5. **Ensuring Security:** Despite the reduced waiting depth for votes, each voter chain in Prism is still operating slowly enough to tolerate adversarial hash power (). This ensures security against attacks even with shorter confirmation times.

8.3.2 Tentative Confrmation rule

By employing the tentative confirmation rule based on the function h(t), Prism achieves a confirmation time that is observable and accurate even in the absence of direct time information in the blockchain. This allows for efficient and secure confirmation of proposer blocks in the protocol.

1. **Tentative Confirmation Rule:** To confirm proposer block B_p at time t, we look for a time t where the function $h(t) = \frac{1}{2}$. The function $h(t)$ calculates the fraction of stabilized votes (or effective vote from one voter chain) at time t.

2. **Closed Form Expression for $h(t)$:** The closed form expression for $h(t)$ can be derived in the appendix and depends only on the mining rate λ and the tolerable adversary hash power fraction β.

3. **Numerical Calculation:** For instance, when $\beta = 0.3$, we can numerically calculate $t\frac{3}{\lambda}$, resulting in a confirmation time that is a small factor of the average inter-block arrival time.

4. **Time Estimation:** Time is not directly observable in a proof-of-work blockchain due to potential inaccuracies or adversarial manipulation of timestamps in blocks. However, it can be estimated from the growth of the voter chains in Prism.

5. **Accurate Estimation:** The large number of voter chains in Prism ensures accurate estimation of time even for short periods due to the law of large numbers. The total mining rate of all voter chains is $m\lambda$, and on average $m\lambda$ voter blocks are mined within t units of time.

6. **Block-Based Confirmation Rule:** Using the conversion from time to blocks, a block-based confirmation rule becomes observable and practical.

8.3.3 Final confrmation rule

Exactly, in Prism, the tentative confirmation rule states that proposer block B_p will be confirmed when $mt^*\lambda$ voter blocks are mined and there is no competing proposer block at the same level. As mentioned earlier, t is a function of the mining rate λ and the tolerated adversary hash power fraction β. For example, when $\beta = 0.3$, it is numerically calculated that $t\frac{3}{\lambda}$.
Therefore, under these conditions, Prism confirms the proposer block B_p once $3m$ voter blocks are mined, ensuring a secure and efficient confirmation process. The large number of voter chains (m) and the ability to estimate time accurately through voter chain growth contribute to the reduction in confirmation latency while maintaining security against adversarial attacks.

8.4 Non-isolated Proposer Block

Consider now the case when the transaction block TB is referred to by an honest proposer block H_1 which is not isolated at its level, i.e. H_1 is matched by an adversarial public proposer block A_1 (the competing proposer block could also be honest). This matching could persist for L levels until reaching a level when there is an isolated

honest proposer block. See Figure 5(b) for the special case of $L = 1$. Let us separately consider the life cycle of an honest transaction vs. a double-spent one.

8.4.1 Honest transaction

For confrmation of an honest transaction, a naive approach would be to wait until we can definitively confirm either H_1 or A_1. However, this approach may be slow due to adversarial attacks attempting to balance votes between H_1 and A_1. A key insight is that for honest transactions, we don't need to know which of H_1 and A_1 will be confirmed—only that one of them will be confirmed. This weaker form of list confirmation works because if A_1 eventually gets confirmed, a later honest proposer block can still refer to TB.
To confirm an honest transaction at level i, we need two events:

1. **List confirmation of all levels up to i:** Once we have list-confirmed a set of proposer blocks at level i referring to TB (e.g., either H_1 or A_1 will be the leader), we know that no other block can be the leader at that level. However, list confirmation alone is not enough if the transaction is not present in all ledgers. In that case, we also need to wait for an isolated honest proposer level where the proposer block will include TB in the ledger.

2. **Isolated honest proposer level:** Once this isolated honest proposer level is confirmed, and all the preceding levels are list-confirmed, we can be sure that TB will appear in the final ledger.

The confirmation latency is thus the maximum of two parts:

1. **List confirmation:** We quickly confirm that the adversary cannot produce a private block A with more votes than the votes of public blocks H_1 and A_1. The logic is similar to the case of an isolated honest proposer block discussed above, viewing the situation as a race between honest nodes voting for the public blocks H_1 or A_1 and the adversary voting for A. Adversarial actions (e.g., presenting H_1 to half the honest nodes and A_1 to the other half) can cause the number of votes to be evenly split between H_1 and A_1, which can slow down list confirmation, albeit not significantly.

2. **Isolated honest proposer level:** For example, in Figure 5(b), if we wait until level 2, we see an isolated public proposer block H_2, which can be fast confirmed. At this point, we know that the final leader sequence at levels 1 and 2 is either H_1, H_2 or A_1, H_2, both of which contain our honest transaction since H_2 refers to all previous unconfirmed transaction blocks.

8.4.2 Double-spent transaction

For confirmation of double-spent transactions, we require stronger conditions than those mentioned earlier. Instead of list confirmation, we need unique block confirmation, which confirms which block at a proposer level will be the ultimate leader. This is achieved once list confirmation occurs, and one of the list-confirmed blocks can be reliably declared the winner. If one of the public proposer blocks H_1 or A_1 gathers many more votes than the other block, then we can fast confirm a unique leader, even for double-spent transactions.
However, certain adversarial attacks, like balancing the votes on H_1 and A_1, can cause the number of votes to be evenly split between H_1 and A_1. In such cases, we cannot fast confirm a leader block. In this scenario, we must wait until every vote on H1 and A1 stabilizes, at which point either H_1 or A_1 is confirmed, and only one of the double-spent transactions is accepted. A content-dependent tie-breaking rule can be used to break ties after votes are stabilized, ensuring that the final decision on the accepted transaction is deterministic.

8.5 Scaling latency via Prism

The latency of Prism, denoted as τ_{Prism}, under the "normal path" (i.e., when there is a single proposer block for a level) is given by:

$$\tau_{Prism} = O(\frac{1}{\lambda}),$$

where λ is the mining rate. This latency is independent of the error probability ϵ when the parameter m (number of voter chains) is chosen to be sufficiently large. The constant hidden in the $O(\mathring{u})$ notation depends only on the

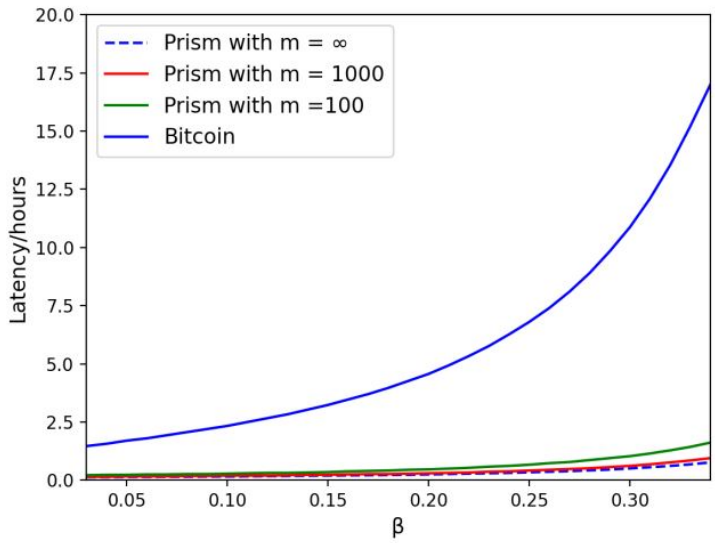

Figure 8.6: Comparison between Bitcoin and Prism latency under various values of β. We set λ to be 6 blocks per hour and the error probability $\epsilon = 10^3$. This also justifes the choice of m in the implementation of Prism.

fraction of tolerable adversary hash power β.

Compared to Bitcoin, Prism achieves faster confirmation because it only needs to wait until the effective vote $h(t)$ is greater than 0.5. The use of many voter chains in Prism allows the application of the law of large numbers horizontally (in space). This means that the majority of the voter chains have voted for a proposer block irreversibly, and hence we can confirm the block with high probability.

In contrast, in Bitcoin, there is only one chain, so we have to wait until $h(t)$ is $1 - \epsilon$ to guarantee a small deconfirmation probability ϵ. In Bitcoin, the law of large numbers is applied vertically (in time), as discussed in Lecture 6.

The comparison between Bitcoin and Prism latency is illustrated in Figure 6. In summary, by using many voter chains, Prism eliminates the $ln(\frac{1}{\lambda})$ term present in Bitcoin's latency. This decouples latency from security and achieves fast confirmation.