Blockchain & Cryptocurrencies

Francesco Farinola

April 2022

Contents

1 Cryptocurrencies

1.1 Bitcoin

2009 - announced by Satoshi Nakamoto, a pseudonum for person or group of person. In May 201 the first Bitcoin purchase - 10.000 BTC for a couple of pizzas. 2011-2013 advent of Silk road and Pirate Roberts. End of 2013 price skyrocket.

Bitcoin is a P2P network of nodes, each node keeps a log of all Bitcoin transactions - ledger known as the **blockchain**.

Transactions are broadcast to nodes in the P2P network. Nodes validate transactions and valid transactions are added to the blockchain and broadcast. Accepted only when it appears on the blockchain.

The ultimate goal is that *all transactions are known and agreed upon by the network* - global consensun of events in ledger.

Characteristics:

- Anonimity and privacy
- Openness - just need internet to access it
- small fees
- decentralization
- very volatile

Possibilities for remittances, bank the unbanked and micro-payments.

Its value is based on the faith that you have in the value of what you can procure with BTC.

1.1.1 Wallets

Bitcoin is a protocol accessed using a client application that speaks the protocol. Just like a web browser, many implementation:

- **Desktop**: offer several features, autonomy and control options. Very convenient and is available as your device. Run on general-use OS so has security disadvantages as they are often insecure and poorly configured.
- **Mobile**: runs on smartphones OS - similar to desktop.
- **Web wallet**: through a web browser and store the user's wallet on a server owned bu third party. Convenient as nothing to install and works on multiple devices. But security worries as hackers can break in, get private keys and transfer bitcoins.
- **Hardware**: devices that operate a secure self-contained bitcoin wallet on special-purpose hardware. Via USB or via NFC on a mobile device. The most secure and suitable for large amounts.

- **Paper**: can be printed for long-term storage, a low-tech means of storing. Offline storage also referred as cold storage. With a public (to receive) and private key(to send - never share).

1.2 Other cryptos

Bitcoin hard forks:

- **Bitcoin Cash BCH**: rejects Segwit with adjustable block size up to 8MB (around 54 tps)
- **Bitcoin Gold BTG**: uses EQUINASH as MHPoW, difficult to adjust every block
- **Bitcoin Diamond BTD**: max block size 8MB, with max supply of 210M and amount encryped.

Others:

- **Ethereum ETH**: infinite max supply, introduces smart contracts, used to develop DAPPs, has a crypto-fuel called Ether. Mining: Etash PoW, block time:10-19s.
- **Litecoin LTC**: max supply 84M, mining Scrypt MHPoW (Memory Hard)and has higher tps than bitcoin.
- **IOTA (MIOTA)**: Introduces the Tangle DAG where a new transaction approves two random. t is premined so no fees,has quantum proof and needs a coordinator with tps scalable.
- **DASH**: max supply of 18.9M, tps scalable. Network of 3000 Masternodes, has InstantSend and PrivateSend for instant and private payments, one of the first successful DAO and introduces Decentralized Blockchain Governance.
- **Z-Cash ZEC**: Introduces Decentralized Anonymous Payment, uses zk-SPARK (zero-knowledge cryptography)
- **Ripple XRP**: designed to connect different payment systems together and integrate with the financial system, fees almost inexistent 0.00001 XRP.

1.3 Stablecoins

Are price stable cryptos, pegget to another asset.

Because cryptos are unnecessary currency risk as we can't pay salaries due to volatility, it is difficult to transact without them, and there are users that don't want to speculate but just want to escape the banking system.

FairCoin is a fork of Bitcoin 0.12. Promote south-south exchanges - developing countries' producers that buy and sell their product. Widely used in

ethical purchasing groups (Italian GAS). Is offering fixed exchange rates to merchants. However, as FairCoin is also listed on some exchanges, some people are also investing into the growth of its value. This makes it necessary, to adapt the official exchange rate from time to time by the decision of a general assembly. Rate: 1:20 (20 FairCoins per euro then 1:1 Dec 2017, 1.2:1 Jan 2018) - KEEP OUT SPECULATORS and attract long-term investors that support the project.

Type of stablecoins:

- **Fiat-Asset Collateralized Currency**: backed by real-world currency USD-UST - 1-to-1 ration against USD. When a user wants to liquidate his stablecoins, he destroy the stablecoins and wire his the USD. 100% stable, can be redeemed at any time, less vulnerable to hacks since no collateral held in the blockchain.

  Centralized - a custodian must store the fiat, need audits ensure transparency, highly regulated, expensive and slow liquidation to fiat.

- **Crypto-collateralized currency**: backed on reserve of another cryptocurrency. Entire system on blockchain → decentralized. As volatile as the crypto backing it. These are over-collateralized - for every coin of stable there is more than one coin of crypto in reserve. The more volatile, the bigger the ratio should be.

  PROS: decentralized, liquidation is cheap and quick, transparent - easy to inspect the collateralization ratio.

  CONS: less stable, can be auto-liquidated during price crash - → holders lose their collateral. Tied to the health of anothr crypto, inefficient use of capital, highest complexity to ensure stability.

- **Non-collateralized currency:** aims to maintain stability without relying on collateral reserve. Value of money determined through supply and demand. Code the logic of central banks into smart contracts which use **oracles**: monitor price of the coin on exchanges, create new coins when price goes up, buy back and destroy them when price goes down.

  PROS: no collateral requires, most decentralized and independent, not tied to any fiat or crypto, aim to stability.

  CONS: most vulnerable to crypto decline or crash, no ability to liquidate. Some complexity, Difficult to analyze health, requires continual growth

Libra: facebook's crypto, stable, collateralized with a set of stable and liquid assets, helps protect against speculative swings, non-anonymous (require authentication). Delayed - launch separate Calibra wallet which supports multiple currencies, one of which is Libra.

2 Smart Contracts

SC are computer program code stored in the blockchain. Act as agreements.

Terms and conditions are accessible to all parties. No way to dispute once the contract is established. All participants run the same code. Each node can verify the logic of SC. Privacy may be an issue.

A smart contract can be writte in a Turing complete language (Ethereum), can do anything that a normal computer can do - but all nodes need to run the code in parallel.

Can be used to build currencies, financial derivatives, voting systems, decentralized organization, data feeds, title registries.

Transaction protocol that executes the terms of a contract

2.0.1 Decentralized application DAPP

Contract plus graphical interface for contract execution. (SC on blockchain and user interface on decentralized file system).

Examples:

- Vueve project: social media that pays when creating videos built with blockchain
- 4G capital: micro-credit company for Africa's fintech
- FirstBlood: global decentralized eSports paltforms for gamers to play in matches using a ERC20 token - players gain rewards for activity
- BitShares an open-source financial management platform based on Graphene where BTS tokens are used as collateral for decentralized financial services (decentralized exchanges, banking, derivative creation)
- Ethlance : records freelances' work in Ethereum with feedback

2.1 Initial Coin Offering ICO

Unregulated means by which someone can raise money from supporters by issuing tokens - **crowdsale**. Buy virtual tokens paying through crypo-money and potentially earn a return on their investments.

Similar to **IPO Initial Public Offering** for stocks of companies to the public. Company has to disclose financial and business information and investors purchase shares of a company (VISA, Facebook and Alibaba). ICOs are similar but **without regulations**, but will be regulated by SEC the Security and Exchange Commision (CONSOB italian).

2.1.1 Airdrops

A marketing strategy for distribution of tokens usually for free to random people to gain attention and new followers in the hope of having a **Network effect** $\rightarrow$ Application utility: the more the users, the more investors, the higher the value, the more developers and repeat $\rightarrow$ Financial utility: the earlier the people join, the more they earn for time and money.

Hubs: SEC for USA, Finma for Switzerland, MAS for Singapore.

2.1.2 Tezos case

Tezos raised in july 2017, 232 million dollar through its ICO. The token sale was uncappped and lauched just before the huge growth of bitcoin. The launch delayed multiple times due to continuous conflict but then resolved.

This taught us that market sentiment is an important factor in short term. And don't follow tweets or Reddit threads saying the coin will moon.

2.2 Tokens

A token is just a **smart contract** running on top of a blockchain and can be obtained by buying them in exchange for fiat currencies or by performing specific services on the network (mining). Appreciates and depreciates based on the demand. There are currency, utility or security token. These tokens have different properties, based on their **Ethereum Request for Comments (ERC)**

2.2.1 ERC-20

Defines a common list of rules than an Ethereum token has to implement. DAPP developers are encouraged to follow the standards to ensure that their tokens can undergo interactions with various wallets, exchanges and smart contracts without any issues. Those cannot be distinguished → **fungible tokens** (similar to money)

It is basically a set of 6 functions that can be recognized and identified by other smart contracts:

```
// https://github.com/ethereum/EIPs/issues/20
pragma solidity ^0.4.0;
contract ERC20 {
    string public constant NAME = "Token Name";
    string public constant SYMBOL = "SYM";
    uint8 public constant DECIMALS = 18; // most common number of
        decimal places

    function totalSupply() constant returns (uint totalSupply);
    function balanceOf(address _owner) constant returns (uint balance);
    function transfer(address _to, uint _value) returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
        returns (bool success);
    function approve(address _spender, uint _value) returns (bool
        success);
    function allowance(address _owner, address _spender) constant
        returns (uint remaining);

    event Transfer(address indexed _from, address indexed _to, uint
        _value);
    event Approval(address indexed _owner, address indexed _spender,
        uint _value);
```

```
}
```

- Get the total token supply
- Get account balance of another address
- Send an amount of tokens to an address
- Send an amount from an address to another
- Allow a spender to withdraw from my account multiple times up to an amount
- Returns the amount a spender is still allowed to withdraw

The other two are triggers to transfer and approve.

OpenZeppelin is a framework of reusable smart contracts for Ethereum and other EVM blockchains focused on security with a modular approach and open source.

2.2.2 ERC-721

For handling deeds where each token is unique and non-interchangeable → non-fungible tokens (for digital and physical goods - NFTs...) (CryptoKitties)

```
contract ERC721 {
    // ERC20 compatible functions
    function name() constant returns (string name);
    function symbol() constant returns (string symbol);
    function totalSupply() constant returns (uint totalSupply);
    function balanceOf(address _owner) constant returns (uint balance);
    // Functions that define ownership
    function ownerOf(uint _tokenId) constant returns (address _owner);
    function approve(address _to, uint _tokenId);
    function takeOwnership(uint _tokenId);
    function transfer(address _to, uint _tokenId) returns (bool success);
    function tokenOfOwnerByIndex(address _owner, uint _index) constant
        returns (uint
    tokenId);
    // Token metadata
    function tokenMetadata(uint _tokenId) constant returns (string
        infoUrl);
    // Events
    event Transfer(address indexed _from, address indexed _to, uint
        _tokenId);
    event Approval(address indexed _owner, address indexed _spender,
        uint _tokenId);
}
```

tokenOfOwnerByIndex: An owner can own more than one token at one time. The contract keeps a record of the IDs of each token that each user owns. Because of this, each individual token owned by a user can be retrieved by its index in the list of tokens owned by the user.

tokenMetadata: What makes non-fungible items non-fungible is their unique set of attributes. But, storing data on the blockchain that tell the defining characteristics of each token is extremely expensive. To combat this, we can store references, like an IPFS hash or HTTPS link, to each token's attributes on the chain (so that a program outside of the chain can find more information about the token). These references are data about data, or metadata. This function lets discover a token's metadata, or the link to the its data.

2.2.3 ERC-1190

Non-fungible token for the payment of royalties. Enable the creator of a unique digital object, like a work of art or an object in a game, to benefit automatically every time the rights to that object are used. Transactions are sale of the ownership license, sale of the creative license, a rental of the digital asset to a third party for a fixed period of time.

Creator can get Revenue or Rental Revenue from the change in ownership value: the gamer can use the object as long as they want. If the in-game asset gains notoriety, its value can rise. Gamer decides to sell the ownership license keeps most of the revenue from the sale but a fixed portion passes automatically to the holder of the creative license (in this case, the creator).

The creator can also sell the creative license but the new holder would get all the future royalties.

2.3 STO: Security Token Offering

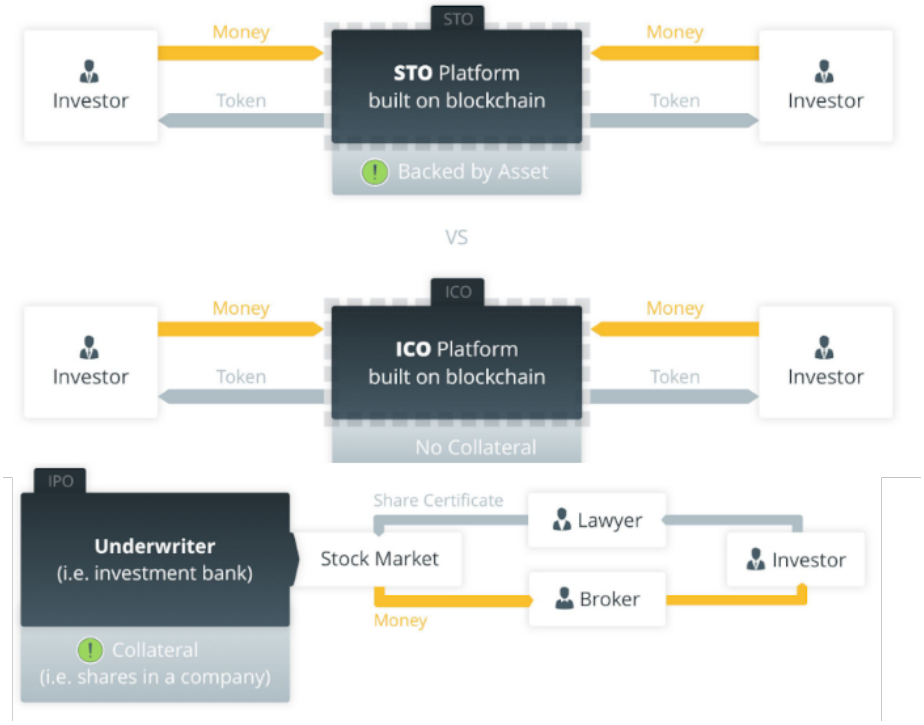
Security: an investment product that is backed by a real-world asset. STO is an investment contract into an underlying investment asset $\rightarrow$ information on the ownership of the investment product, recorded on a blockchain and issued as a token, comply with regulatory governance, is an hybrid of ICOs and IPOs. Need to pay lawyers and advisors, but typically won't have to pay large fees to investment banks or brokerages.

IPOs issue share certificates on traditional markets, used in private companies that want to go public.

2.4 DAO: Decentralized Autonomous Organization

Is a traditional organization but decentralized, a DO on blockchain, a set of people that interact according to a protocol.

A smart contract may maintain the record of each individual's holdings, a voting system to select the positions of the board of directors and the employees, smart property management...



Governance is a set of rules, norms and actions that define, regulate and account interactions between different users/actors and regulates the process of decision-making. There is no centralized legal entity or employment contracts.

Key points:

- Tokens: A DAO needs some internal property (token) that is valuable in some way, and it has the ability to use that property for rewarding certain activities
- Autonomous: Once deployed the entity is independent of its creators. DAOs are open source. Financial transaction record and program rules are maintained on a blockchain
- Consensus: To withdraw or move funds, a majority of stakeholders must agree on the decision
- Contractor: A DAO cannot build a product, write code or develop hardware. It needs a contractor to accomplish its goals. Contractors get appointed via voting of token holders
- Proposals for making decisions. To avoid people overloading the network with proposals, a DAO could require a monetary deposit to prevent people from spamming the network

- Voting: After submitting a proposal, voting takes place

TheDAO project failed in 2016, exploited and stole 1/3 of funds caused a hard fork in Ethereum. Was a business model for organizing both commercial and non-profit enterprises. Participants receive DAO tokens, then vote for which projects to fund.

2.4.1 DASH

P2P cryptocurrency forked out of Bitcoin. Aims to offer faster and more private transactions to users and has a decentralized blockchain governance system which runs autonomously, doesn't have a CEO and is made up of volunteers from all over the world.

Tipically the reward for mining is given all to miners, but here the reward is divided as: 45% to **miners** for Proof of Work, 45% to **masternodes** for Proof of Service which decide the projects to fund by voting and 10% to **treasury**.

Treasury: allows to fund everything that is required on the network (Money-as-a-service provider) and allows the DAO to be independent - No need for grants, donations to foundation, corporate sponsorship. With this treasury, the community can hire developers, auditors, marketers and translators.

Masternodes is a node that proves an ownership of at least 1000 Dash. With this kind of wealth of stake, masternodes are incentivized to make good decisions from which they will profit.

Pre-proposals are made on a forum, they receive feedback and eventually get published on blockchain for 5 Dash. Masternodes review and rank, once consensus is reached, funds are allocated and developers interact with masternodes who want to track progress. Funds can be revoked.

Evolution: InstandSend, PrivateSend, Bank-like payments, Simplifications for daily use.

3 Ethereum

Extends the blockchain concepts from Bitcoin: runs computer code equivalently on many computers around the world; what Bitcoin does for distributed data storage, Ethereum does for distributed data storage + computations (smart contracts).

Similarity with Bitcoin: manages a blockchain with blocks of data and smart contracts, the blockchain is public and permitionless, has a Proof of Work mining (even Proof of Stake is proposed), has a native cryptocurrency: ether (ETH).

Two types of accounts:

- **Externally owned accounts:** Controlled by people/private keys similar to Bitcoin
- **Contract accounts:** Controlled by smart contract code. Allow developers to use Ethereum as a general purpose framework to create decentralized applications (DApps)

Smart Contracts on Ethereum are seen as **autonomous agents living on the Ethereum blockchain**, can send and receive transactions, are activated when they receive a transaction, and can be deactivated, have a fee per CPU step.

To run: create a transaction sending a payment of ETH to the contract, and possibly supplying some other input information. Each miner runs the smart contract and if no one is behaving badly, each computer on the Ethereum network will produce the same output. The winning miner will publish the block to the rest of the network.

The code executed by the EVM is a **low-level, stack-based bytecode language** (gonna be replaced by eWASM). 3 storages used:

- **Stack**: temporary variables
- **Memory**: an infinitely expandable byte array for temporary variables inside functions
- **Long-term storage**: a key/value store for state variables which a permanent on blockchain.

Two kind of transactions:

- **External transactions**: Originated by an externally owned account, a bridge between the external world to the internal state of Ethereum
- **Internal transactions (messages)**: Possibly generated by a smart contract after it received another transaction/message

Anti-denial-of-service model: Network could be disrupted by malicious transactions (infinite loops). So we charge a fee (gas) for each computational step.

Every transaction has two (additional) parameters: **STARTGAS**: maximum number of computational steps the tx execution is allowed to take (If exceeded the transaction is cancelled without refunding the fees). **GASPRICE**: representing the fee (in Ether fractions) the sender pays per computational step.

Fees are rewards for miners that validate transactions and run the code. Users must have enough Ether in their account balance to cover the maximum price. The sender is refunded for any unused gas at the end of the transaction, exchanged at the original rate. If a called function goes out of gas, the parent functions does not revert, but the childs are aborted too.

Messages do not have gas limit. GasLimit is determined by the external creator of the original transaction, by some external address.

3.1 Improving scalability

Main limitation: every node executes every transaction. The network is only as fast as the individual nodes rather than the sum of its parts.

3.1.1 Layer 1 solutions

Sharding:

- Grouping subsets of nodes (called shards)
- Each shard processes transactions specific to that shard
- Shards are run in parallel → increase on throughput
- Consensus within a shard is reached through a Proof of Stake (Casper) consensus of randomly selected nodes that are applied to a shard for specific consensus round

3.1.2 Layer 2 solutions

- ***State Channels***: Performing transactions and other state updates "off-chain". Put results "on-chain" with the possibility to verify that off-chain state changes were correct - A State is locked with smart contracts, off-chain transactions are done, result written back on-chain.

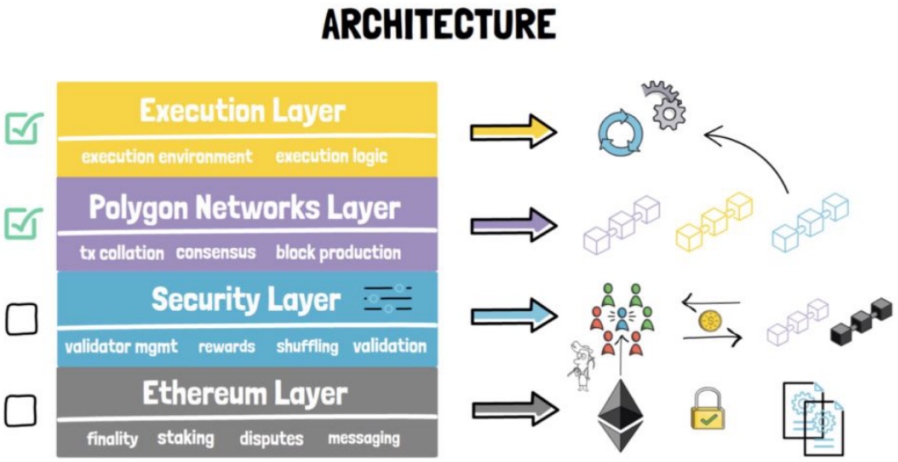
  Benefits: **Speed** (blockchain not involved), **Throughput** (number of off-chain transactions is not limited by the blockchain's throughput), **Privacy** (off-chain transactions do not show up in the public ledger), **Cost** (only the final settlement transaction).

  Drawbacks: **Trustworthiness** (Off-chain micropayment transactions might not be as trustworthy since are not stored on immutable data storage), **Liquidity** (To establish a payment channel, money from one or both sides of the channel participants needs to be locked up in a smart contract for the lifetime of the payment channel), **Modifiability** (A new wallet or extension to the existing wallet is needed to support the micropayment protocol)

  MicroRaiden for frequent, fast and free micro-payments of ERC20 tokens. Do not require any fees. Will want to charge fees on a low percentage basis for providing their own channels to the network. This creates competitive market.

- ***Plasma***: Creation of "child" blockchains attached to the "main" ethereum blockchain. Fewer nodes in the child blockchains. Perform complex operations at the child-chain level with minimal interaction with the ethereum main-chain

  - **Plasma MVP:** Minimal Viable Plasma is a design for an extremely simple UTXO-based plasma chain. The basic Plasma MVP specification enables high-throughput payment transactions, but does not support more complicated constructions like scripts or smart contracts.
  - **Plasma Cash:** is a plasma design primarily built for storing and transferring non-fungible tokens



- **Plasma Debit:** is like Plasma Cash, except every token is a payment channel between the user and the chain operator. It's sort of like a big Lightning hub, but the channels can be transferred just like a Plasma Cash token!
- **Plasma Prime** is a fancy new design that makes use of RSA accumulators (section below) to solve the problem of large history proofs in Plasma Cash
- **Loom Network** is a multichain interop platform live in production since early 2018. Optimized for scaling high-performance dapps that require a fast and smooth user experience, the network allows dapps to offer a UX comparable to traditional applications and onboard new users without the friction of needing to download crypto wallet software.

  Loom also has integrations to Bitcoin, Ethereum, Binance Chain, and Tron (with EOS and Cosmos coming soon). This allows developers to integrate assets from all major chains, as well as build a dapp only once and offer it to users on all platforms simultaneously
- **Polygon** is a protocol and a framework for building and connecting Ethereum-compatible blockchain networks. Aggregating scalable solutions on Ethereum supporting a multi-chain Ethereum ecosystem. 10x faster and 10x cheaper than Ethereum.

  Ethereum as the base layer for the Polygon chain, high security but low flexibility. *Security layer:* Specialized, non-mandatory layer managing a set of validators that can periodically check the validity of any Polygon chain for a fee - as a meta-blockchain parallel to Ethereum. *Polygon network layer:* A constellation of sovereign blockchain networks. Each of the networks serves its respective community, maintaining functions like transaction collation, local con-

sensus, block production. *Execution layer:* This layer interprets and executes transactions that are agreed upon and included in Polygon networks' blockchains

- ***TrueBit***: Conduct heavy or complex computation off-chain. Verification game: incentive for other parties (challengers) to check the solvers' work

4 Solidity

After you deploy a contract, it's immutable - permanently on the blockchain. No way to patch bugs after deployment - you would have to tell users to start using a different smart contract address that has the fix.

High level smart contract languages are **deterministic**: all nodes must produce the same results, no random() primitives. We don't want the smart-contract to be able to read or delete the hard-drives of the nodes it runs on. **Oracles** exist that provide contracts with data about the outside world in a trustable way.

You can use a browser-based IDE to program your first contracts (without deploying them on a network) with **remix-solidity**: a web based environment to write, compile and debug smart contracts.

Cryptozombies.io: an introduction course on Solidity to create a multi-player game on the blockchain.

OpenZeppelin.org: A tested framework of reusable smart contracts for Ethereum and other EVM and eWASM blockchains

4.1 Language

Is an high level language similar to JavScript. It is like object oriented language and supports inheritance with new types as well. Special variables to get information about blockchain like **msg.sender**, **msg.value** and **now**.

Contracts are similar to classes and can contain state variables, functions, modifiers, events, structs and enums.

Data types: bool, int8...int256 (with negative), uint8,..., uint256, address (Ethereum address 20 bytes value - 0x0cE446255506E92DF41614C46F1d6df9Cc969183), several members like balance, string and enum.

Mapping: KeyType => ValueType is like a hash table where we store a value using a key.

reputation[0x111] = 5 and with var val = reputation[0x111], var will have value 5.

Pragma is the first like of solidity contracts, specifies which solidity compiler version to be used.

Struct:

```
struct Student{
    uint age;
    string name; }
```

```
Student vitalik = Student (23, "Vitalik")
```

Fixed size byte arrays includes byte arrays from length 1 to 32, bytes1,...,bytes32

```
uint[4] fixedArray;
string[7] stringArray;
```

Dynamically-sized byte arrays

```
Student [] public students;
students.push(vitalik)
```

Data location in EVM:

- Storage: Persistent and expensive
- Memory: Persists for function call, cheap
- Call stack: Cheapest, local variables, limited call depth : 1024

Memory: keyword used to store data for the execution of a contract. It holds functions argument data and is wiped after execution.

Storage: can be seen as the default solidity data storage. It holds data persistently and consumes more gas.

Special variable about blockchain: give information about the state of the blockchain/transactions

- block.coinbase (address payable) Current block miner's address
- block.difficulty (uint) Current block difficulty
- block.number (uint): Current block number
- blockhash(uint blockNumber) returns (bytes32) Gives hash of the given block and will only work for the 256 most recent block due to the reason of scalability.
- block.timestamp: Current block timestamp as seconds since unix epoch
- gasleft() returns (uint256): Remaining gas
- msg.sender (address payable) Sender of the message (current call)
- msg.value (uint) Number of wei sent with the message
- msg.sig (bytes4) First four bytes of the calldata (i.e. function identifier)
- msg.data (bytes calldata) Complete calldata
- now (uint) Current block timestamp (alias for block.timestamp)
- tx.gasprice (uint) Gas price of the transaction

- block.gaslimit (uint) Current block gaslimit
- tx.origin (address payable) Sender of the transaction (full call chain)

Functions access modifiers:

- **public**: can be inherited and can be accessed by external elements. All can access a public element. Solidity copied array arguments to memory (expensive)
- **external**: can't be inherited but it can be accessed by external elements. Current contract instance can't access external element, it can be accessed externally only. Can be called from other contracts and transactions. Sometimes more efficient when they receive large arrays of data. Arguments read directly from calldata (cheaper - saves copying step)
- **private**: accessible only from this contract. Doesn't get inherited and can't be accessed by external elements.
- **internal**: can be inherited but can't be accessed by external elements. Only the base contract and derived contract can access internal element.

Exception Handling

`assert(condition)` to check if something completely unexpected happens and make sure that the contract will not fall into an invalid state - underflow, overflow, divide by 0. Throws an error and stops execution if some condition is not true. Revert all the changes made and **consumes ALL the gas of the transaction**.

`require(condition)` a guard against transactions that you would not want to execute although the function would allow it if require() was omitted - to validate user inputs (require(input¡20)). Revert all the changes and **consume the gas used up to the point of failure**.

View Specifier: do not write to the storage.

Pure Specifier: neither read nor write to the storage.

Actions that modify the blockchain are contract creation, changes to the state of a contract - require to be submitted as a new transaction picked up by miners.

View and pure are used for data retrieval from the blockchain with no charge for the gas used to execute the call.

View to guarantee security: only wat to modify someone else's balance would be to steal his private key.

```
mapping (address => balance) balances;
function setMyBalance(uint _myBalance) public{
    balances[msg.sender] = _myBalance;}
function whatIsMyBalance() public view returns (uint){
    return balances[msg.sender];}
```

4.2 Proof of Existence

Mapping of hases, to record if they have been notarized. **Notarize** calls **proofFor** to calculate the hash and store the proof calling **storeProof**. The other 3 functions are pure and view so they don't write or read on the blockchain - no gas.

```
// Proof of Existence contract, version 2
contract ProofOfExistence2 {
    mapping (bytes32 => bool) private proofs;
    // store a proof of existence in the contract state
    function storeProof(bytes32 proof) {
       proofs[proof] = true;}
    // calculate and store the proof for a document
    function notarize(string document) {
       var proof = proofFor(document);
       storeProof(proof);}
    // helper function to get a document's sha256
    function proofFor(string document) pure returns (bytes32) {
       return sha256(document);}
    // check if a document has been notarized
    function checkDocument(string document) view returns (bool) {
       var proof = proofFor(document);
       return hasProof(proof);}
    // returns true if proof is stored
    function hasProof(bytes32 proof) view returns(bool) {
       return proofs[proof];}}
```

4.3 DNS service

The contract is a database inside the Ethereum network that can be added to, but not removed from. Anyone can register a name with some value.

```
pragma solidity ^0.4.0;
contract SimpleDNS {
    struct Record {
       address owner;
       string ipaddr;}
    mapping (string => Record) records;
    function addDomain(string _domain, string _ipaddr) {
       if (records[_domain].owner != address(0x0)
    && records[_domain].owner != msg.sender) {
           return;}
       records[_domain] = Record(msg.sender, _ipaddr);}
    function getDomain(string _domain) view returns(string) {
       return records[_domain].ipaddr;}
    function transfer(string _domain, address _to) {
       if (records[_domain].owner != msg.sender) {
           throw;}
```

```
        records[_domain].owner = _to;} }
```

4.4 Payable

A special type of function that can receive Ether requires a certain payment to the contract in order to execute a function. Possible because in Ethereum the money, the data and the contract code live in the same system. Simply mark the function as **payable**: `function buySomething() external payable{require ( msg.value == 0.001 ether); transferThing(msg.sender) }`

After you send Ether to a contract, it gets stored in the contract's Ethereum account.

We can use **function modifiers** in the body to ensure that sender has enough in his account. Like this we avoid to mix precondition logic into state-transition logic and reuse the same code in different contexts.

```
modifier only_with_at_least(uint x){
    require (balances[msg.sender]>=x); _;>}
function transfer(uint _amount, address _dest) public
                        only_with_at_least(_amount) {
    balances[msg.sender] -= _amount;
    balances[_dest] += _amount;}
```

4.5 Inheritance, Interfaces

Solidity supports contract inheritance between smart contracts, also multiple inheritance. **Internal** variables are available.

If you nee your contract talk to another that you don't own, you can import it or use an abstract contract/interface.

To use a contract interface, treat it like a contract skeleton - only declare the functions you need to import and do not define the bodies but just the declaration followed by a semicolon.

To use the functions in the interface you have to know the address of the called contract, to instantiate a local proxy to the contract, call the function through the proxy.

```
contract A{
    function f1(bool arg1, uint arg2) public returns (uint);}
contract YourContract{
    function doYourThing(address AddressOfA) public returns (uint){
    A myA = A (AddressOfA);
    return myA.f1(true, 3);}}
contract B is A{
    function f1(bool arg1, uint arg2) public return (uint){return
        "ciao";}}
```

4.6 Events and Transfers between addresses

A way to communicate that something happened on the blockchain. The app in the front-end, can be 'listening' for certain events and take action when they happen. The event is registered in a log that can be queried. Useful for interfaces.

```
contract Event{
    event IntegersAdded(uint x, uint y, uint result);
function add(uint _x, uint _y) public returns(uint){
    uint result = _x + _y;
    emit IntegersAdded(_x,_y,result);
    return result;}}
```

Balance of an address queried using the property balance. Ether can be sent to an address using the transfer function. If it is a contract address, its code, its **fallback function** is executed together with the transfer call.

Fallback function: a contract may have precisely one unnamed function, which cannot have arguments, nor return anything. Fallback functions are executed if a contract is called and no other function matches the specified function identifier, if no data is supplied, whenever a contract receives plain Ether, without any data.

4.7 Security issues

- **Reentrancy**: A function is repeatedly called before first invocation is finished. Similar to the DAO attack.
- **Cross Function Race conditions**: Functions in the same contract share state. If multiple contract functions are called, what happens?
- **Solutions**: Finish internal work before calling external functions - External functions at the end of you function, Use proven design patterns -Pull over push payments, Mutual exclusion
- **Send can fail:** When sending money, your code should always be prepared for the send function to fail
- **Loops can trigger gas limit:** Be careful when looping over state variables, which can grow in size and make gas consumption hit the limits
- **Timestamp dependency:** Do not use timestamps in critical parts of the code, because miners can manipulate them.

4.8 Editing and Deployment

Environments:

- Remix: an in-browser Solidity editor to edit and test contract code - simulator local and private

- Ganache : local private development blockchain
- Metamask: an Ethereum waller implemented as plugin for public and pivate blockchains
- Geth : public/private blockchains
- Public test network: Ropsten network
- Etherscan: Ethereum blockchain explorer
- Web3.js, Web.py: frontend calls to deployed contracts

Options:

- **Ethereum mainnet:** highest security, real value, immutable, public, expensive, slow, highest risk
- **Public Test network**: public, free ether, not final, still slow
- **Private network**: fast, free, private, isolated, no real value.

4.8.1 Development workflow

1. **Create account:** Geth via the JavaScript console, Metamask via browser extension, Ganache initializes a blockchain with 10 accounts with 100 ether each, Remix with 5 pre-funded accounts with 100 ether each.
2. **Fund accounts**: on private with ganache or remix, on mainnet with purchase of Ether from exchange, on testnest from a faucet
3. **Develop:** with Remix, a browser based Solidity smart contract development IDE or via text editors like Atom, Sublime or VSCode
4. **Compile: Remix** contains a Solidity smar contract compiler or with **SoIC** a command-line compiler or **The Truffle** a framework with a compiler.
5. **Sign & Deploy:** with **Mist** a web3-enabled browser with wallet and contract deployment, or Remix which connects to an Ethereum node and deploy contracts or with Truffle, sophisticated contract migration management.
6. **Interact & Test**: Remix can help test transacting with deployed contracts, Metamask allows to interact with deployed applications via a browser interface, Truffle includes an automated testing framework and Web3.js/Web.py for frontend calls to deployed contracts.

Suggestion: Truffle + Ganache + Metamask

5 Tools and Data structures for Distributed Systems

To create a new identity: create a random key pair (sk, pk) where pk is the public key - the name of the identity, your pseudonym and referred as the address - and pk is the private key which lets you to "speak for" the identity.

None knows who you are since the key "looks random" - these decentralized identities are called addresses in Bitcoin.

5.1 Hash functions

are deterministic transformation which take any string as input and outputs a fixes size string with 256 bits, efficiently computable. These are generally used in hash table and dictionaries for fast look-up.

$$H : \{0,1\}^* \to \{0,1\}^\lambda \tag{1}$$

In crypto these are used to verify the integrity of messages and files, for digital signatures, password verification and Proof-of-work.

Cryptographic hash functions are computationally efficient, and offer security properties as they are collision-free, hiding and puzzle-friendly.

Collision exist but if output has 256 bits then we have 2^256 possible output values - you need to guess to find a collision. It is possible but low chance to collide.

Like the Birthday paradox, with 23 people ina room the possibility that a pair share a common birthday is higher than 50%.

In Data Integrity, we just attach to the message we want to send, the digest. So we hash the document and remmber its hash, when sending the message we attach the digest calculated and if the two are identical, integrity is verified.

Hiding When given H(x) it is unfeasible to find x. Not always success, it can fail.

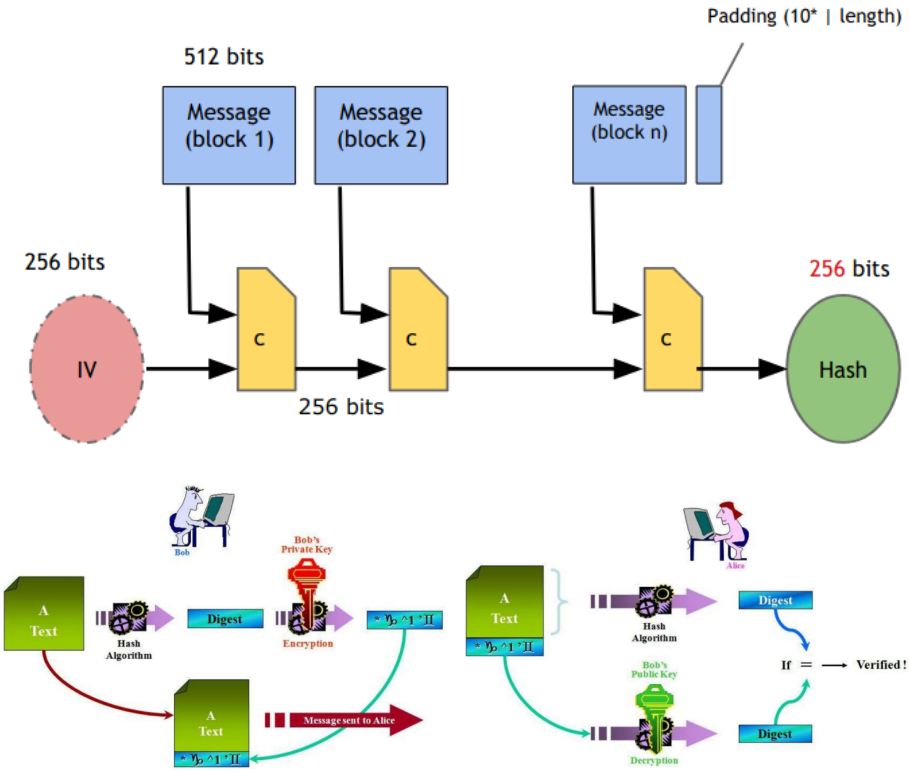
If we have personal data like a phone number there are 1 trillion of possible values and with SHA256 it would take less than 13 minuts to find the Hash, with GPU even faster x60.

It only works when we have a large set of possible values of x, there are no particular x that are more likely and there are no particular x that are more interesting than others.

To devide crypto-puzzles: given a message x we concatenate a random string r and hash it to get a random string - it is unfeasible to find x. It is more easy with a random value from a random set Y. The smaller Y, the more difficult the puzzle.

Existing Hash Functions MD5 (128 bits), SHA-1 (160 bits) and SHA-256/512. MD5 and SHA-1 are attackable but collision not verified. Suggested to use SHA2 or SHA-3/Keccak.

SHA-256 hash function:



Were IV is an initialization vector (number used for every call to the function), c is the compression function and if c is collision free then SHA-256 is also collision free.

5.2 Hash pointers

Pointer to where some information is stored, together with a cryptographic hash of the info. These are ways not also to retrieve info but also to verify that it hasn't changed. It is comprised of two parts:

- Pointer: to where some information is stored, used to get the information
- Cryptographic hash of that information: used to verify that some information has not changed

Given n items, to prove membership of an item in a has pointer structure costs O(n)

5.3 Merkle Trees

Allow each piece of the data to be accompanied by a short proof (logarithmic in size of data) - the piece is a part of the whole.

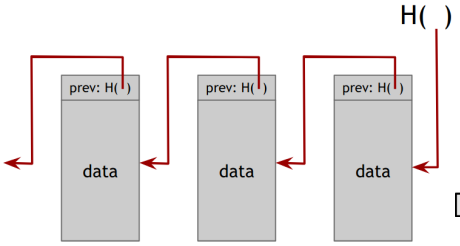


Figure 1: Hash pointers

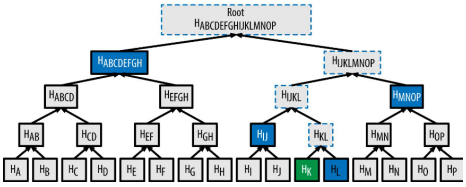


Figure 2: Merkle trees

Given n items to prove membership in merkle trees it costs O(log(n))

Merkle Trees **applications**: BitTorrent uses Merkle trees to ensure that the files you download from peers haven't been tampered. IPFS has linked authenticated data structures. Cryptos (bitcoin and ethereum), Distributed version control (git and mercurial), Copy-On-Write Filesystems (btrfs/ZFS), Distributed NoSQL Databases (Cassandra, Riak, Dynamo).

In BitTorrent a peer receives a piece and a Merkle proof for it, calculates the hash of that piece, request the root hash from the trusted site, using this information the client recalculates the root hash of the tree, and compares it to the root hash it received from the trusted source.

5.4 Trie

"Try" from reTRIEval - a tree representing a collection of strings with one node per common prefix, a natural way to represent a map where keys are strings. The smallest tree such that each edge is labeled with a character c, a node has at most one outgoing edge labeled c, each key is spelled out along some path starting at the root.

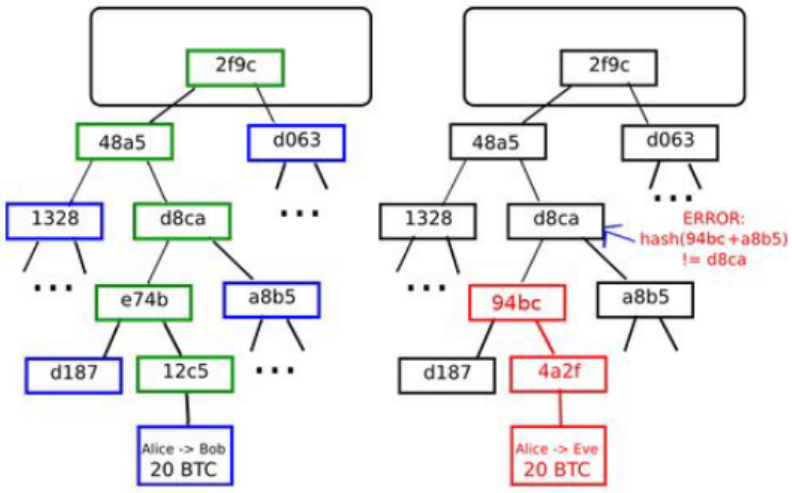


Figure 3: On left side it suffices to present only a small number of nodes in a Merkle tree to give a proof of the validity of the branch. On right, any attempt to change any part of the Merkle tree will eventually lead to an inconsistency somewhere up the chain.

Let Σ be a fixed alphabet, each node x corresponds to some string given by the path traced from the root to the node. It sores a flag (green or red) indicating whether the string is spelled out to this point is in the set and an array of $|\Sigma|$ pointers, one for each character.

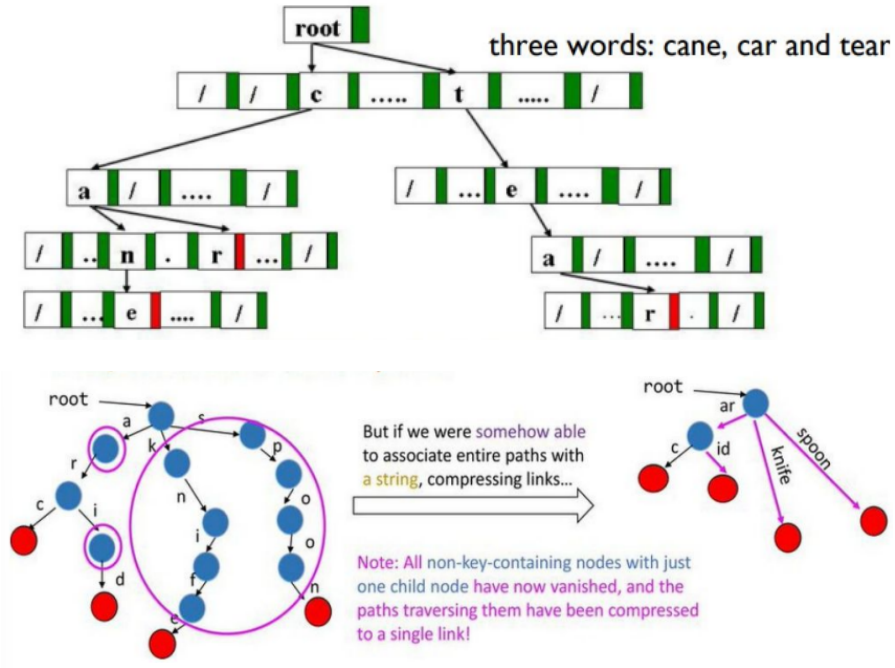
The string w lookup cost follow at most $|w|$ pointers to get to the node for w if it exists. At each step look up (LU) of a pointer, depends on the data structure implementing the pointers:

- array of child pointers of size Σ: waste of space, but LU O(l)
- a hash table of child pointers: less waste of space, LU O(l)
- list of child pointers: compact, LU is O(Σ) in the worst-case
- a binary search tree of child pointers: compact and LU is O(lg) is the worst-case and O(l) at minimum

Total time is O($|w|$) - lookup cost is independent of the number of strings in the trie.

To insert a new string we proceed as before like a normal lookup and set the flag of final node.

Removal of string: mark the node as no longer containing the word - if node has no children we remove that node and repeat the process at the node one level higher.



Extremely space-inefficient: with N nodes and $|\Sigma|$ alphabet we need space $O(N * |\Sigma|)$ due to the pointers in each node.

Solution - Patricia Trie: Practical Algorithm to Retrieve Information Coded in Alphanumeric which makes a compressed version where we replace a chain of one-child nodes not including terminal nodes with an edge labeled with a string of more than one character.

5.5 Merkle Patricia Tries

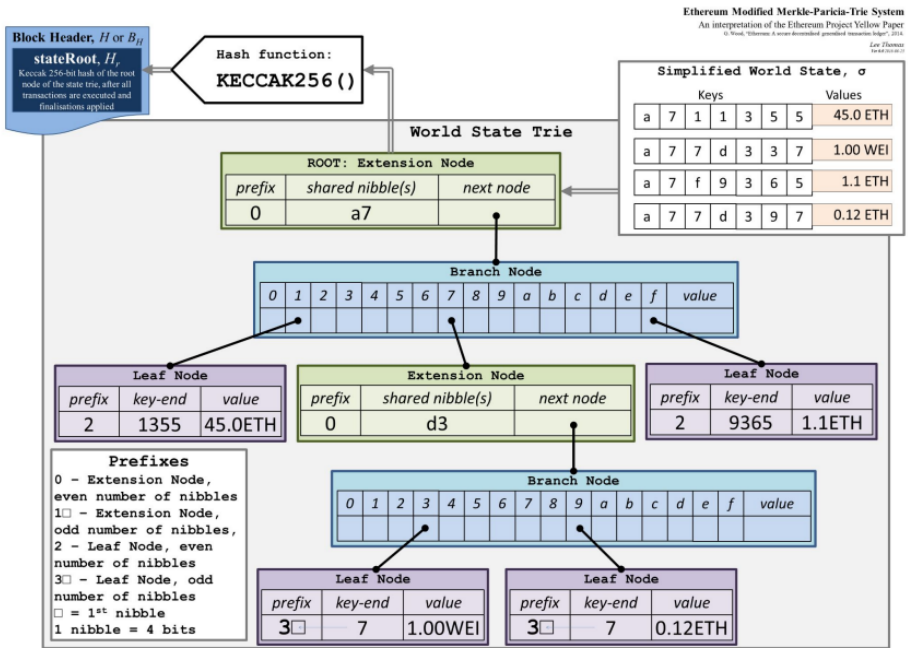
are used by Ethereum to represent the state as a combination of key (address) / value (balance) pairs. A combination of Merkle tree to maintain data integrity and Patricia Tree to enable faster search of data:

- Patricia trie with keys
- Merkle tree of the values stores in the leaves of the Patricia trie.

Only the Merkle tree root is meaningful and stored in the blockchain - root node is a cryptographic fingerprint of the entire data structure.

5.6 Bloom Filters

Consider the set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements chosen from a very large universe U. We want to know if k is an element of S.



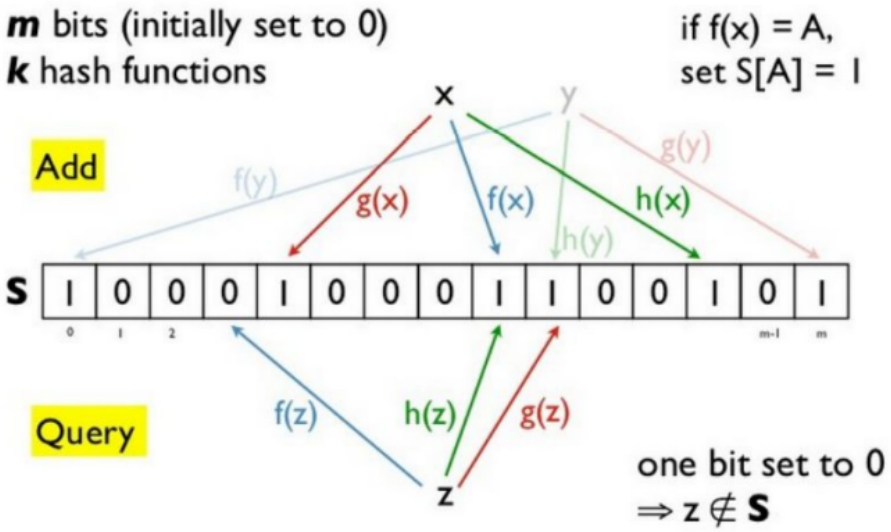
S may be a set of keywords describing the files shared by a peer, selected from the universe of all the keywords (Gnutella 0.6); the set of pieces of file owned by a peer (BitTorrent); a set of Bitcoin addresses: lightweight mobile nodes build Bloom filters with the interesting addresses; send them to the full nodes: bandwidth saving (Bitcoin).

Problem: choose a representation of the elements in S such that: the result of the query is computed efficiently and the space for the representation of the elements is reduced. The results may be approximated to save space and there is a possibility of returning false positives. - We need a trade off.

Given:

- $S = \{s_1, s_2, \ldots, s_n\}$ of n elements
- a vector B of m ($n << m$, generally, nk < m) bits with $b \in \{0, 1\}$
- k has independent functions $h_1, ..., h_k$ which return a value uniformly distributed in the range $[1..m]$
- Construction procedure for a Bloom Filter $B[1..m]$:
  $\forall x \in S, B[h_j(x)] = 1, \forall j = 1, 2, .., k$
- a bit in B may be target for more than 1 element.

When lookup, we apply the k hash function to y and if a bit=0, the element does not belong to the set.



False positives: To test for membership, you simply hash the string with the same hash functions, then see if those values are set in the bit vector. If they aren't, you know that the element isn't in the set. If they are, you only know that it might be, because another element or some combination of other elements could have set the same bits.

The basic assumption is that hash functions are random and independent.

Probability of false positives depends on:

- m/n: number of bits exploited for each element of the set
- k: number of hash functions

If m/n is fixed, it seems two conflicting factors for defining k: decreasing k increases the number of 0 and hence the probability to have a false positive should decrease, but ... increasing k increases the number of elements to be checked and the precision of the method. Hence the probability of false positive should decrease.

Fixed the ratio m/n, the probability of false negatives first decreases, then increases, when considering increasing values of k.

Let us now suppose that k is fixed, the probability of false positives exponentially decreases when m increases (m number of bits in the filter). For low values of m/n (few bits for each element), the probability is higher for large values of k.

The **optimal values** are obtained when the probability that a bit is equal to 0 after the application of the k functions to the n elements is equal to 1/2.

An "optimal" Bloom Filter is a "random bitstring" where half of the bits chosen uniformly at random, is 0.

Operations:

Union: given two Bloom filters with same number of bits and number of hash function the union is the bitwise OR of B1 U B2.

Delete: t it is not possible to set to 0 all the elements indexed by the output of the hash functions, because of the conflicts.

Counting Bloom Filters: each entry of the Bloom Filter is a counter, instead of a single bit: exploited to implement the removal of elements from the Bloom filter - at insertion time, increment the counter - at deletion time, decrement the counter.

Intersection: Bloom Filter from S1 intersection S2 is given by the bitwise AND of B1 and B2.

5.6.1 Applications

Google BigTable and Apache Cassandra use Bloom filters to avoid costly disk lookups considerably and to increases the performance of a database query.

The Google Chrome web browser uses a Bloom filter to identify malicious URLs.

Any URL is first checked against a local Bloom filter and only upon a hit a full check of the URL is performed.

Bitcoin uses Bloom filters to verify payments without running a full network node.

Gnutella 0.6 exploits a simplified version of Bloom Filter.

6 Distributed Hash Tables DHT

DHTs is a hashmap whose entries are distributed across a bunch of computers + routing protocol to find peers with a given ¡key, value¿ pair.

Each node is identified by a number or node ID. Data (usually file hashes) is mapped to the same ID space. Node ID provides a direct map to file hashes: A node stores data whose id are similar to the node id.

Interface

- `put(GUID, data)`: Publish an object with GUID. The data is stored in all the nodes responsible for a replica
- `get(GUID)`: The data associated with the object GUID are retrieved
- `remove(GUID)`: Remove all the replicas of the object whose GUID is GUID

6.0.1 DHT Structure

Step 1:

Definition of the keyspace. Ex: Logic keyspace as a ring. Assumption: keyspace $0, \ldots, 2^m - 1$ larger than the amount of data items (es m=160), need for a total ordering and distance notion (in modulus). Each node is assigned a

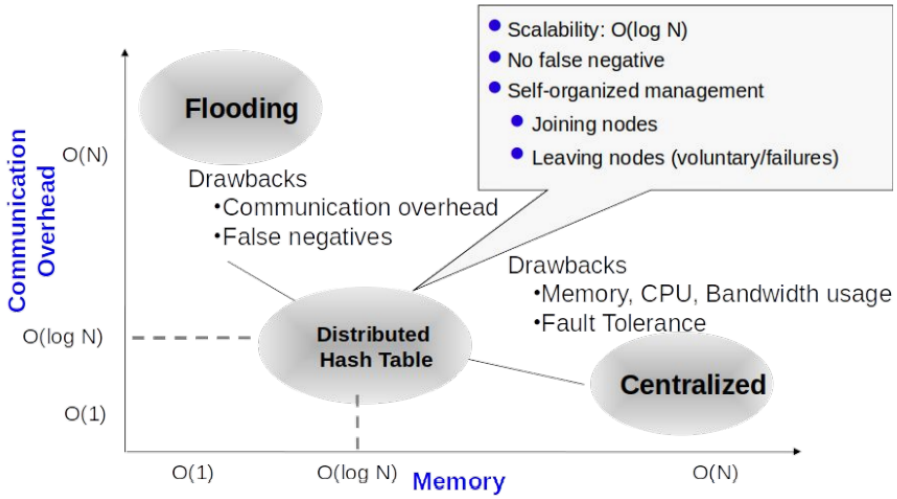
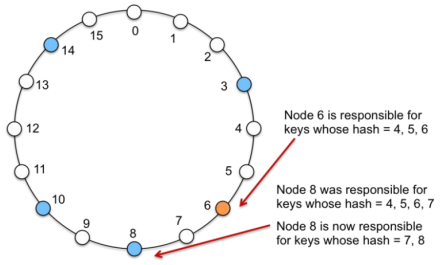


Figure 4: A tradeoff between Flooding (ask to people repeatedly if not available) and Centralized (which involves use of server memory, CPU, bandwidth and can happen fault tolerance - if server is unavailable you can't retrieve)



single key (its ID) - Mapping node - key through a hash function. The overlay network is not correlated to the real (Internet) topology.

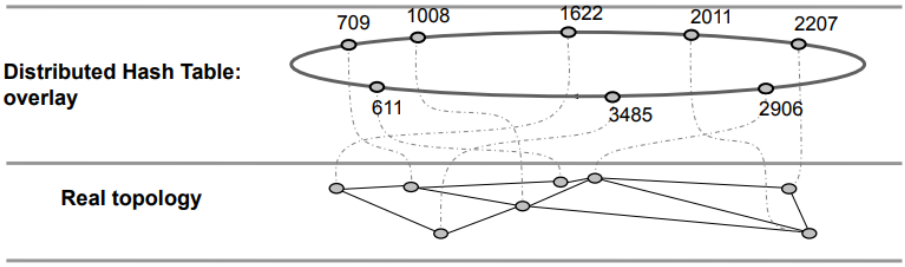
Step 2:

Each node is responsible for a set of data items stored in the DHT. This data set is composed of items with keys in a contiguous interval in the keyspace. Typical DHT implementation: map key to node whose id is "close" to the key (need distance function).

Data items mapped in the same keyspace of the nodes, through the hash function:

- NodeID = hash(node's IP address)
- KeyID = hash(key): Key is a file name - Hash can be performed on the filename or on its entire content
- m-bit identifier

Approach	Memory	Communiction Overhead	Complex Queries	False Negative	Robustness
Central Server	O(N)	O(1)	✔	✔	
Pure P2P (flooding)	O(1)	$O(N^2)$	✔		✔
DHT	O(log N)	O(log N)		✔	✔



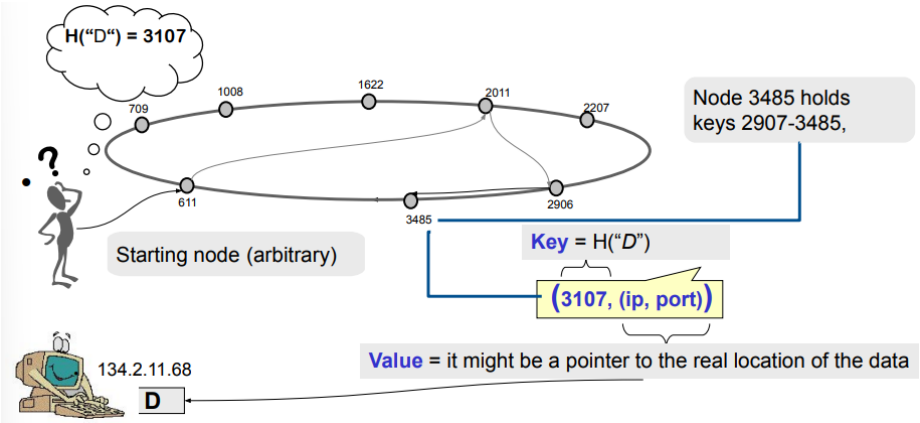
- Cryptographic hash function (e.g., SHA-1)

Sometimes a certain redundancy is introduced (overlapping).

Problem: uniform distribution among DHT nodes - **unbalanced load** can be caused by a node with a bigger portion of the key space, uniform partitioning of keyspace among nodes but an interval contains more data, or certain items are more requested than others. This leads to lower robustness and scalability.

Given a data item D, fin the node that holds key=H(D). Each node forwards query to closer node. Maintain routing tables adaptively and each node knows some other nodes.

Solution - Key based routing



6.0.2 Applications

DHTs offer a generic distributed service to store and index information.

The data item can be file, IP address, any kind of information.

Examples of applications: DNS implementation with key hostname and value IP address. P2P storage systems, Distributed search engine, Distributed caching, Distributed file system.

Popular DHTs: Chord (MIT), Pastry, CAN, P-Grid, Kademlia (eMula, BitTorrent), Symphony etc.

Used by Dynamo (Amazon), IPFS, BitTorrent, Cassandra, Oracle (caching system), Coral CDN, OpenDHT.

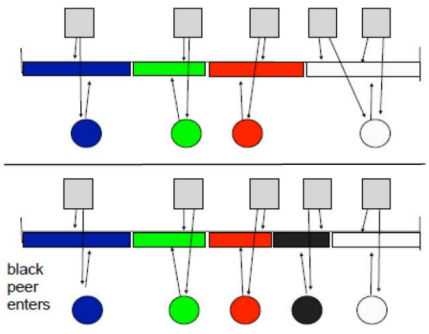


Figure 5: Nodes joining the DHT

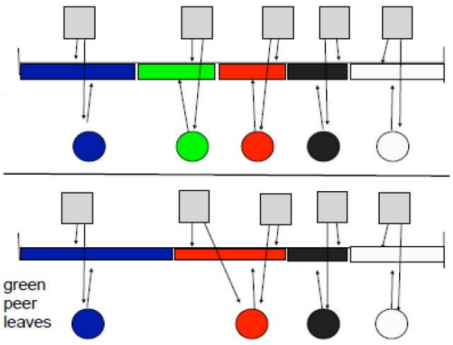


Figure 6: Nodes leaveing the DHT

6.0.3 Chord

Topology

Hash-table storage with `put (key, value)` to insert data items in Chord and `value = get(key)` to lookup data.

Identifiers generated through SHA-1 (Secure Hash Standard).

Each node has a ID: id = sha-1 (IP address, port).

Each data item has a key: key = sha-1 (data).

Keys and IDs are mapped in the same keyspace.

Probability of 2 nodes or 2 keys have the same identifier must be negligible so m-bit ID space (2^m IDs, usually m=160),. Think of them organized in a logical ring according to their IDs.

We have a uniform distribution of keys to nodes, where KeyID k is assigned to first node whose NodeID $\geq$ k (clockwise) - denoted `successor(k)`.

Basic Lookup

```
get(k):
    if (i have k):
        return "me"
    else:
        P next node
        return P.lookup(k)
```

Each node remembers only the next node - O(N) not so good.

To get O(logN) we need fingers. Each node has **fingers** to nodes 1/2 way around the ID space from it, 1/4 the way - or in reverse order fingers at distance $2^0, 2^1, ..., 2^i$ and finger[i] at p contains $successor(p + 2^{i-1})$

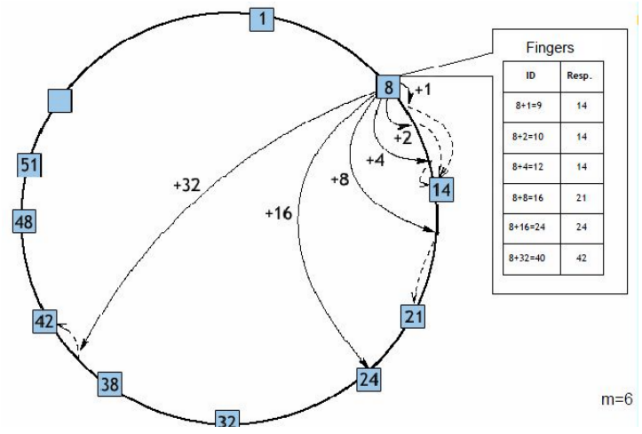


Figure 7: The i-th finger of n, points $1/2^{n-i}$ way around the ring

At each node we have a finger table, first finger is successor, predecessor. For m=160 and 1 million nodes the log n is 20.

7 Blockchain details

P2P is Hard. The main challenge is determining ownership. Without strong ownership we are prone to forgery (a user can mint arbitrary currency), double spending (how do you validate) and theft (impossible to separate true and false claims about ownership).

For **theft**, each entity is defined by a public/private keypair and knowledge of private keys gives ownership of neucoins in associated wallet and coins are transferred from one public key to another → transaction is signed by the private key.

For **forgery**, we add some reference to these coins like serial numbers and this reference can be a pointer to a previous transaction.

For **double spending**, we publish the history of all transactions on a public ledger.

To validate we broadcast each transaction to all miners which with a voting mechanism validate them - works well if users are all honest but this is not the case in practice.

7.1 Sybil attack

Another issue of Bitcoin is that everyone is free to participate and because of pseudo-anonymity, nodes do not have persistent long term identities. Also a node can autonomously generate as many addresses as he wants and so, a node in a P2P net can claim multiple identities.

Imagine a transaction is valid when you hear from N nodes. An hacker can introduce many fake nodes and so one node = one vote doesn't work as it is vulnerable.

7.2 Distributed Consensus

The key technical challenge of decentralized e-cash. It covers how transactions are initially received, ordered in the ledger, replicated to the cluster and committed permanently to the ledger.

This is hard as nodes may crash, be malicious and network is imperfect as not all pairs of nodes are connected, there may be faults in network and latency.

Typical approaches in distributed systems exploit **explicit consensus**: nodes have identitites.

It is reliable and has authenticated point-to-point communications. Identity is hard in P2P systems based on internet consensus and prone to sybil attack.

Implicit consensus: the key idea is that at each round, a random node n is picked. n unilaterally proposes, without contacting other nodes, the next block B. Can be viewed as a random leader election on every block. n chooses a block B of transactions from a **mempool** and broadcasts b.

All nodes implicitly accept/reject this block: accept if all transactions are valid and update their blockchains.

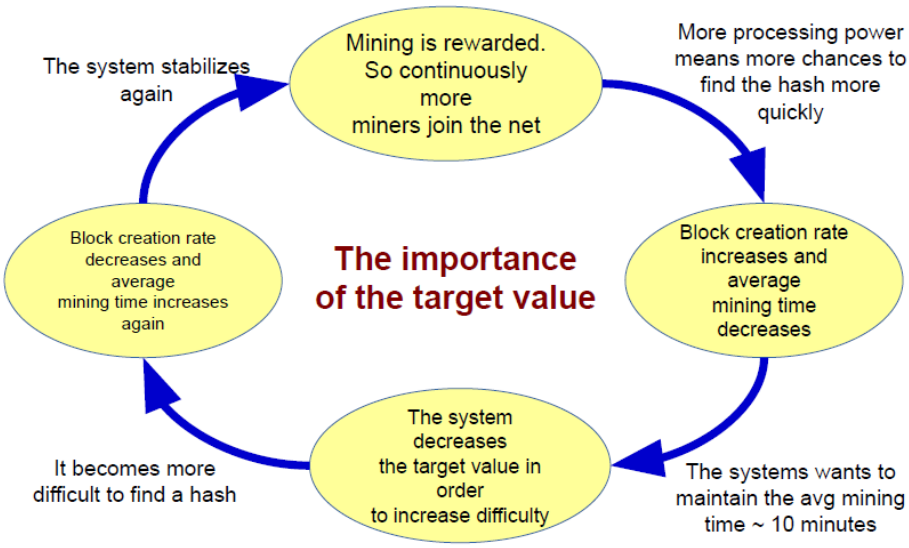
To select a random node we use **computational work**: to add a transaction, you must present a **proof-of-work** generated by solving a crypto-puzzle - for sybil prevention. In bitcoin the majority of sybils is defined as the the majority of computational power - like this a malicious user cannot represent 100x CPU power. Proof-of-Work is like leader election - the miner is the one which decides how to evolve the system

Mining through crypto-puzzles: miners try to solve a puzzle, the first one to solve creates a block and get some reward - the solution of puzzle i defines puzzle i+1.

A puzzle should be cheap to verify, with adjustable difficulty, and the chances of winning is proportional to computational power (hash power). We should guess a combination lock which is difficult to find and easy to verify.

SHA-256 is an hash puzzle where to create a block we need to find a nonce so that $H(nonce|prevhash|tx|\ldots|tx)$ is very small - based only on luck if secure.

- **Difficult to compute**: a relative measure of how difficult it is to find a new block. The difficulty is adjusted periodically as a function of how much hashing power has been deployed by the network of miners.
- **Parametrizable cost**: nodes automatically recalculate the target every 2016 blocks (2 weeks) as the average time between two blocks should be 10 minutes - P(Alice wins next block) = fraction of global hash power she controls.



- **Trivial to verify**: nonce must be published as part of the block so that other miners can verify that $H(nonce|prev\_hash|tx|\ldots|tx) < target$

7.3 Blocks

Block is a group of transactions like a page in a ledger: refer to previous blocks.

The Block **ID** is the has of all other fields, **Prev** is the ID of previous block and **Nonce** is a number chosen to make the ID small enough. Changing the nonce changes the output of the hash function unpredictably.

You cannot change a previous agreed block, you cannot swap blocks since it would change the hash value.

Structure of a block:

- **Magic number** (first 4 bytes) not specific of bitcoin, introduced in unix. Used to identify the type of file/data structure - indicate message origin network.
- **Block size** (4)
- **Version** int of 4 bytes indicate the block format version (currently 2)
- **Previous Block Hash** - SHA235 a uint256 of 32 bytes - comprises from Version to Nonce
- **Merkle Root** Top hash of the merkle tree built from all transactions - a uint256 of 32 bytes
- **Timestamp** uint of 4 bytes in UNIX format of block creation time
- **Difficulty target** - target T for the proof of work problem in compact format - a uint of 4 bytes
- **Nonce** - allowing variations for solving the proof of work problem
- **Transaction counter** a variable of 1-9 bytes
- **Transaction list** a variable of upto 1MB.

Transactions in the same block are considered to have happened at the same time. The ones not yet in a block are called unconfirmed and can be collected by any node to create a block.

If two nodes find different block simultaneously, nodes always accept the longest chain (the most work). When this is broadcast, all nodes will switch to the longer chain.

Fork event in blockchain: two blocks are found simultaneously, two blocks propagate splitting the network, a new block extends one fork reconverging the network on a new longest chain.

7.4 Incentives for mining

Bitcoin solves the incentive problem in two ways: new blocks mint new coins called coinbase transaction and transactions may include a transaction fee.

Coinbase transaction: miner rewarded by allowing it to mint new coins: Node who wins "mines" a fixed amount of coins as a prize and this is why it's called mining.

Current reward is 6.25 BTC. Rule: half the mining reward alter all 210.000 blocks (roughly every 4 years). Reward will become 0 in year 2140; 21 million total bitcoins. At this point, only transaction fees will incentivize miners.

7.5 Transactions in Bitcoin

No records of account balances are kept. Ownership of funds is verified through links to previous transactions. To understand how much money any given person has, you need to check all previous transactions.

Two types:

- **Regular** used to transfer existing Bitcoins among different users
- **Coinbase transactions** where new bitcoins are introduced into the system - included in every block as the very first transaction and are meant as a reward for proof of work.

Each transaction is composed of two lists, a list of transaction outputs (holds the recipient address and an amount) and a list of transaction inputs (a tuple consisting of a reference to a previously created output and an hash of the transaction that created the output).

Condition for validity: sum of input funds $\geq$ sum of output funds - the transaction must not spend more than the available inputs.

Transaction fee = sum of input funds - sum of output funds: goes to the miner as a fee and optional.

Through references input linkage, ownership of Bitcoin is passed along in a kind of chain. Validity of each transaction is dependent on previous transactions.

You need to know every transaction - when you install bitcoin wallet it downloads all transactions ever made (over 24h). Owning Bitcoins: there are transaction that point to your name and haven't been spent (UTXO Unspent transaction Output). To add a transaction we need to transmit the transaction, propagate it in the net and let the receiver be aware of the transaction.

7.6 Transaction propagation

Each peer broadcasts (flooding) each transaction to its neighbors - which broadcast it to their neighbors and so on until the entire network knows of the new transaction.

Leave: there is not an explicit way to leave the network. If a node hasn't been head from in a while, other nodes start to forget it - generally 3 hours.

Should i relay a proposed transaction? Sanity checks (not mandatory): the default is that a script matches a whitelist (avoid unusual scripts); haven't seen before (avoid infinite loops); doesn't conflict with others i've relayed (avoid double-spends).

How big is the network? Difficult, at the moment around 15k nodes and 400GB in size which are permanently connected and fully validated.

Block propagation: relay a new block when you hear it if block meets the hash target, block has valid transactions (run all scripts), block builds on the current longest chain (sanity check can be avoided - avoid forks)

Messages for exchanging data:

- **INV** an announcement message, send the hash of blocks of transactions owned by a node (does not contain wholenodes or transactions) - hash and type of the adversarial block - every time a node receives a transaction or a block it announce the transaction to each neighbour
- **GETDATA** request for a block/transaction sent from peers who don't have
- **BLOCK, TRANSACTION** contain the actual block/transaction

GETBLOCK: Allows a peer P which has been disconnected or started for the first time to get block it needs to build the updated blockchain; *Ask a neighbour for its local vision of the blockchain*; The neighbour replies with the hash of a set of blocks at various heights on their local chain. P can find the first common hash and ask for following blocks through the message GETDATA. Iterative process, after having downloaded all the blocks another GETBLOCK is sent.

8 Consensus

Problems with Proof of Work is that it is very expensive and entitites with large hash-rate computing power can control the blockchain.

Mining process

```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
    header = makeBlockHeader(transactions, coinbase_nonce);
    for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
        if (SHA256(SHA256(makeBlock(header, header_nonce)))
        < TARGET)
            break; //block found!
        }
        coinbase_nonce++;}
```

Professional mining centers need cheap electricity, good network and cool climate - e.g. BitFury - a mining center in republic of georgia (ASIC)

A solution for the bitcoin lottery are **mining pools** where group of nodes work together and there's split proceeds when any node finds the next block.

Distributed consensus covers how transactions are initially received, ordered in the ledger, replicated to the cluster and committed permanently to the ledger.

8.1 Proof of Stake PoS

Creator of the next block chosen in a deterministic way, via various combinations of random selection and wealth or age (i.e. the stake)

Stake: the more coins a miner possess, the higher chances he has to find blocks on the blockchain: from proof of work to proof of owning.

Here there is no block reward, miners take transaction fees only and are called **forgers** or **validators**. Also called **virtual mining** - the rich gets richer but it is reasonable that the validator with a large stake is more inclined to contribute to the security of the cryptocurrency and would not endanger the system.

PROs:

Environmentally friendly: PoS is a lot cleaner for the environment than PoW: PoS needs only to own coins → virtual mining and PoW requires hardware and electricity to function.

Fewer cartels: PoS does not require costly hardware: Consensus reached based on the amount of coins each miner has

Investment needed for mining through PoS does not devaluate over time

Impossible 51% attacks: The attacker should have 51% of current currency, which is extremely expensive.

8.2 Delegated Proof of Stake DPoS

Introduces a **voting element** - used in BitShares. The community selects a number of **witnesses** or block producers and users approve the witnesses via a voting system which sign each block.

Only a small number of trusted witnesses verify each block and more transactions can be included in each block to increase the speed of transactions. This can compete on a scale compared to Visa and Mastercard.

The network pays the top witnesses in the system with the greatest reward. Competition remains high for those top spots.

Alternatives: liquid PoS in Tezos or Randomized delegated PoS.

8.3 Proof of Cooperation

Used in FairCoin. Block generation is performed by so-called **Cooperatively Validated Nodes (CVNs)**. CVNs are appointed in a democratic process and can be added or removed from the network dynamically.

To find the CVN that should create the next block a deterministic alg is executed, based on examining the blockchain history in a **Round Robin like**.

8.4 Practical Byzantine Fault Tolerance

PBFT is a solution to the Byzantine Failure model used in Tendermint, IBM's openchain, Corda, Hyperledger, Stellar, Ripple.

PBFT provides transaction finality without the need for confirmations like in PoW. If a proposed block is agreed upon by the nodes in a PBFT system, then that block is final. All honest nodes are agreeing on the state of the system. PBFT is not computationally intensive - reduction in energy usage w.r.t. PoW.

Works well with small consensun group size only. Susceptible to Sybil attacks - a single node can manipualte large number of identities - better to use in combination with other consensun mechanism like PoW.

System model

Asynchronous distributed system Delay (messages are delivered eventually), duplicate or deliver messages out of order.

Byzantine failure model: Faulty replicas may behave arbitrarily

Preventing spoofing and relays and corrupting messages with **public-key signature**: one cannot impersonate other and **collision-resistant hash**: one cannot tamper other's messages.

Algorithm:

The set of replica is R; $|R| = 3f + 1$ (f is of faulty replicas tolerated). Each replica is identified by an integer in $\{0, ..., 3f\}$ and is deterministic and starts at the same initial state. Replicas move through a sequence of configurations called **views**.

A view is a configuration of replicas where $replicap = vmod|R|$ is the primary of view v, all other replicas are backups and **View changes are carried out when the primary appears to have failed**

1. Client sends request to the primary. It is assumed that the client waits for one request to complete before sending the next one
2. Primary validates the request and initiates the 3-phase protocol to ensure consensus among all (non-faulty) replicas:
   - **Pre-prepare**: Acknowledge a unique sequence number for the request
   - **Prepare**: The replicas agree on this sequence number
   - **Commit**: Establish total order across views

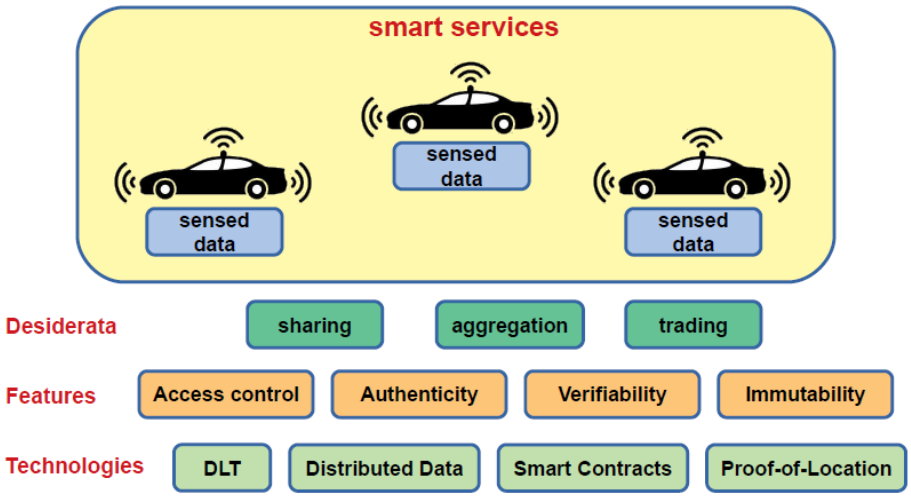
   Pre-prepare and prepare establish a total order of executions of requests.
3. The replicas execute the request and send result directly to the client
4. The client accepts the result after receiving f+1 identical replies

9 IOTA

Typical blockchain limitations are scalability, storage, bandwidth, fees, no data privacy. Not all blockchains are chains of blocks → **Distributed Ledger Technology DLT**.

IOTA is a ledger of things, a permissionless distributed ledger for a new economy.

IoT is here and growing, applied in smart homes, smart buildings (automatic turn off lights and air conditioning after last employee leaves, heating, ventilation and video surveillance), connected rail operations (passenger security, route optimization with collision avoidance, fuel saving and critical sensing with proactive maintenance), connected cars (sensors like before and urban connectivity for reduced congestion and safety), automatic parking, smart city (La spezia uses Kiunsys with 1010 parking spot sensors which communicate free parking spots) - hundreds clouds, thousands of fogs (cars, pc, smartphones) and millions of things (elettrodomestici, smart watch, glasses etc)



IoT requirements: low resource consumption, widespread interoperability, billions of nano-transactions, data integrity.

An attempt from IOTA for a solution is to develop a new Distributed Ledger Architecture designed for the Internet of Things and novel Machine to Machine interactions.

9.1 The Tangle

Instead of blocks that hold multiple transactions, each transaction is another node in the graph. In order to make a new transaction you have to choose two transactions to verify. There are no transaction fees. Removes the dichotomy

between transactions makers and authenticators. Anyone adding a transaction contributes also through mining.

Bundles all transactions in a Directed Acyclic Graph DAG, completely self-regulating, with consensus no longer decoupled and scalable.

No hard limit on the transaction rate - no block size limit involved (a main bottleneck of the blockchain is the block)

Weight: transactions with a larger weight are more important - current implementation sets the weight of any transaction to 1 for simplicity. **Cumulative weight**: own weight is the weight of transaction plus the sum of weights of all transactions that directly or indirectly approve this one.

Depth: Length of the longest reverse-oriented path to some tip. **Height:** Length of the longest oriented path to the genesis. **Score:** Sum of own weights of all transactions approved by this transaction plus the own weight of the transaction itself.

Unforkable

They claim that the network becomes stronger when the number of transaction increases - high tx throughput: if more txs are created, the confirmation rates are getting better.

A tangle can branch off and back into the net creating an offline tangle cluster.

It is quantum computing protected - they may array aroound 2030-2050 and will be able to crack current data encryption methods. IOTA uses Winternitz One-Time Signature scheme: Each time you send a tx from an address, a part of the private key is revealed - Easier for attackers to know you private key via brute force → you can receive as as many txs you want, but once you make a tx from this address, you should NOT reuse this address again.

IOTA has its own hash function called Curl based on SHA3/Keccak was found vulnerable and patched to **Kerl**

Coordinators are full nodes run by the IOTA foundation which create zero value txs called **milestones** which full nodes reference to (checkpointers) - to protect the net at its infancy stage to sustain possible large-scale attacks.

9.2 Transactions

Transaction is an object containing several fields such as address, signature, value and tag. Two types: **transfer of value** have to sign inputs or **message** to send a transaction to an address with no value.

Bundle is a set of transaction involved in a transfer - these are atomic transfer (either all of none transaction in the bundle are accepted).

Transferring IOTAs from one address to another requires several different transactions: An output transaction that increments the recipient's balance by the desired amount; (usually) 1-3 inputs that authorise the spending of the IOTAs from the sender's address(es); (if necessary) a change transaction that sends any remaining amount to a new address owned by the sender.

Making a transaction is a 3 step process:

- **Signing:** Your node (computer / mobile) creates a transaction and sign it with your private key
- **Tip Selection:** Your node chooses two other unconfirmed transactions (tips) using the **Random Walk Monte Carlo (RWMC)** algorithm
- **Proof of Work:** Your node checks if the two transactions are not conflicting. Next, the node performs PoW by solving a cryptographic puzzle. Simpler than Bitcoin and similar to Hashcash not so difficult and can be performed on mobile devices. Prevents spam and Sybil attacks.

Lazy tips are transaction which approve very old ones and so does not help the network and are unlinkely to be approved by anyone. RWMC to avoid lazy tips - select tips at random but with a bias: the probability based on the cumulative weight. RWMC used to select tips and verify txs.

In practice you select some txs deep into the Tangle but not too deep, since you need to select tips in a reasonable time and from these do a random walk towards tips with higher cumulative weight. The two that reach a tip first will indicate the two tips to approve.

Approver's job is to make sure Alice really has 10 IOTAs and does not double spend. Say Charlie runs the tip selection algorithm, and it turns out he needs to approve Alice's transaction. Charlie must verify that Alice really had the IOTAs she spent, if he approves a bad transaction, his own transaction will never be approved! Charlie has to list all the transactions approved directly and indirectly by Alice's transaction, all the way back to the genesis.

Confirmation level of a transaction: execute RWMC N times, let be M the number of times you land on a tip that has a path to your transaction $\rightarrow$ probability of your transaction being accepted is M/N. e.g RWMC 100 times and 60 tips have a path to my transaction - my transaction is 60% to be confirmed.

To avoid double spend, eventually, both conflicting transactions are in the path of validation of one transaction (transaction "5") $\rightarrow$ it sees the conflict and not attach to the elected tips. It reselects tips until it finds valid ones.

9.3 Masked Authenticating Messaging MAM

Means the message is **encrypted** (Masked), is confirmed to be coming **from the device** (Authenticated), a **continuous message stream** is created on the Tangle and will carry on until the device stop publishing the data (Messaging).

Is a **module build on top of IOTA** that makes it possible to send messages fully encrypted from authenticated parties.

Makes it possible for sensors to encrypt entire message streams and securely store those in the Tangle, each on a separate address, and only authorised parties will be able to read and reconstruct the entire message stream.

IOTA uses the **gossip** protocol to propagate messages through the network through a **channel id**.

In a MAM stream or message chain, every message holds a reference to the next message. The message stream only flows one direction and a subscriber with a channel ID has no access to the upstream messages.

The message is encrypted and also contains a signature to prove that the published created the message.

Address is where masked message is being stored and is obtained from a generated parameter called **root**.

There are **access modes**:

- **Public** (everyone): root is used as the address of the (zero value) transaction that the message is published to (and also to encrypt the data) - Anyone that knows a message root will be able to access all messages published from that one on - root is address and is encryption and decryption key.
- **Private** (only me): root hash is used as the messages address and root to encrypt the message. Knowing the address it is not possible to derive the root - cannot decrypt
- **Restricted** (specify viewers): root is combined with an authorization key (side key) and the hash is used as the messages address - side key to decrypt data - The publisher could stop using the side key without changing their channel, so access could be revoked to subscribers - hash(root) is address and sideKey is encryption and decryption key

nextRoot: To post masked message of one generation, one must generate TWO merkle trees. First tree is for the current generation and the other is for next generation.

There are APIs for several languages: C, Go, Java, Javascript, Python, Rust.

Send a Hello World transaction Create the code to send a transaction to a IOTA node:

- Require the library and connect to a node
- Define a depth and a minimum weight magnitude
- Define an address to which you want to send a message
- Define a seed
- Create your "hello world" message
- Define a transaction that sends the message to the address
- Send the transaction

then execute the file and search the transaction in the Tangle.

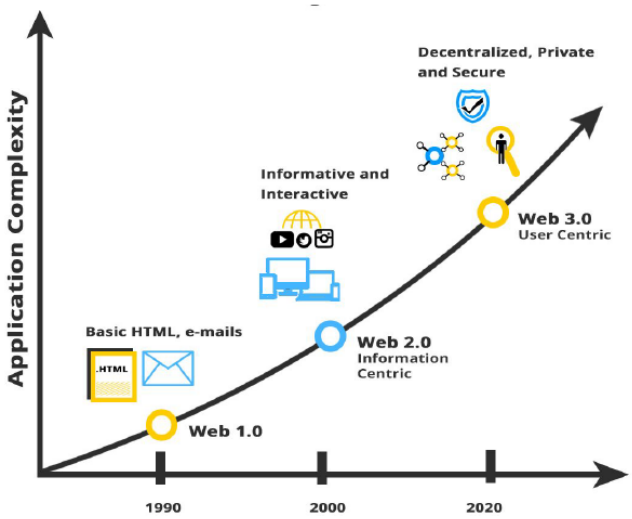
```
// pip install pyota
// Import the packages
from iota import Iota
from iota import ProposedTransaction
from iota import Address
from iota import Tag
from iota import TryteString
// Connect to a node
api = Iota('https://nodes.devnet.iota.org:443', testnet = True)
// The testnet argument sets the minimum weight magnitude to 9
// minimum weight magnitude is the difficulty of PoW
// Define an address to which you want to send a message
address =
'ZLGVEQ9JUZZWCZXLWVNTHBDX9G9KZTJP9VEERIIFHY9SIQKYBVAHIMLHXPQVE9
IXFDDXNHQINXJDRPFDXNYVAPLZAW'
// This address does not have to belong to anyone or have
// IOTA tokens.
// It must only contain a maximum of 81 trytes
// I know, we didnt mentioned trytes in these slides)
// Define a message that you want to send to the address
// and convert it to trytes
message = TryteString.from_unicode('Hello world')
// Define a zero-value transaction that sends the message
// to the address
tx = ProposedTransaction(
address = Address(address),
message = message,
value = 0
)
// Proposed transaction objects are those that you can edit
// because they are not yet attached to the Tangle
// Pass your ProposedTransaction object to the send_transfer()
// method to do tip selection, remote proof of work,
// and to send the bundle to the node
// Note: bundle = message
result = api.send_transfer(transfers = [tx])
print(result['bundle'].tail_transaction.hash)
// You can read your transaction, using a utility such as the
// Tangle explorer
```

Through a proper selection of full nodes it is possible to achieve reliable ledger updates (low errors). However, the measured latencies are relevant.

10 Interplanetary File System IPFS

A new hypermedia distribution protocol - to upgrade the Web.

Web has problems of information collected by big servers, censorship (Turkey) and poor bandwidth utilization.



We need **Content based addressing** instead of location based addressing. We want to create a file system that can handle a lot of data, be accessible everywhere and very fast.

Protocol that allows to connect all computing devices in a peer-to-peer network with the same distributed file system. Goal → same interface of the Web - web developers should be able to layer whatever app on top of IPFS.**Brave the first browser enabling a decentralized web**

Characteristics: peer-to-peer, distribute and versioning of large data, no privileged nodes, nodes store IPFS objects in a local storage.

Nodes do not need to connect to the original source of the content, being connected to anyone who has the content is enough.

Access requests originate from a node - retrieved content is cached and idle cached content is garbage collected. Nodes cannot be forced to fetch anything or to pin anything.

In IPFS, each node of the network keeps a cache of the files it has downloaded and shared. Helps to share if other people needs it a Bittorrent-like swarm but, a single swarm for all IPFS, not a swarm for each content

Want to resolve paths the same way unix does: /ipns/example.com/foo/bar/paz.png → /ipfs/QmW98pJrc6FZ6/foo/bar/paz.png

If node goes offline there is no guarantees of replication - Possible solutions: **incentivize** nodes to store files and make them available, **proactively distribute** files guarantee a number of copies in the network.

Filecoin is built over IPFS and incetivize file sharing - a decentralized market for storage, if you have free space on your disk you can make money storing files of other users.

You **can't delete content from the network**. If you add data to the network, and another node chooses to rehost it, there is no way to cause them to delete it from their blockstore.

IPFS makes **no claims about anonymity**. We can discover the IP address of a peer hosting any given block

Application: Distributed uncensorable wikipedia (turkey has censored it), **D-Tube distributed Youtube** (Crypto-decentralized video platform, built on top of the STEEM Blockchain and the IPFS), **Orbit P2P** chat, **Zeronet** (Open, free and uncensorable websites, using Bitcoin cryptography and BitTorrent network), **ipfs-image-dapp** (Simple prototype app: you can store data to IPFS and place the immutable, permanent IPFS links into a blockchain transaction)

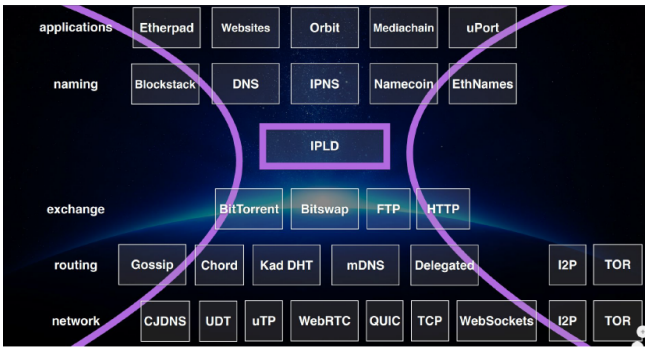
10.1 How to use IPFS

Through APIs of Go, Javascript, HTTP or command line interface CLI (ongoing Python also).

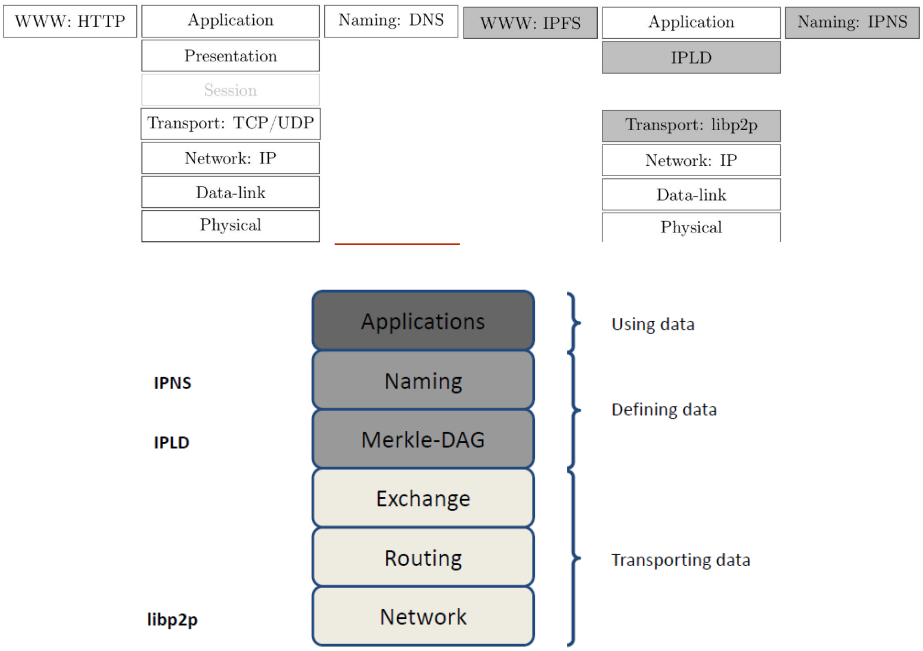
With CLI:

- `ipfs init` for initialization - a node is a computer system running the IPFS daemon. A node is a running process that can find, publish and replicate objects.
- `ipfs daemon` to take your node online
- `ipfs swarm peers` to see your peers
- `ipfs cat link > file.png`
  `open file.png` to get objects from the network
- `hash=echo "..." curl "link"` to add a content to IPFS and see the result through curl
- web console to http://localhost:5001/webui

10.2 The Stack



Routing: DHT with a set of improvements for: security: S/Kademlia and performance: sloppy Hierarchical DHT. Block exchangers: BitTorrent. Version control system: Git. Self-Certified FileSystems: SFS



10.3 Defining the data

IPLD: InterPlanetary Linked Data is the data model of the content-addressable web. IPLD is a single namespace for all hash-inspired protocols. Through IPLD, links can be traversed across protocols, allowing you explore data regardless of the underlying protocol.

10.3.1 MerkleDAG:

a directed acyclic graph whose nodes are data objects (unstructured binary data of size less than 256 kB) and links are hash pointers whose structure has name, hash and size. Any data structures are represented as DAGs.

Files are dag nodes and are split into many blocks of size ¡256kB. Directories are also dag nodes.

```
$ ipfs object get QmarHSr9aSNaPSR6G9KFPbuLV9aEqJfTk1y9B8pdwqK4Rq //CID
    Content Identifier
{"Links": [{
       "Name": "AnotherName",
       "Hash": "QmVtYjNij3KeyGmcgg7yVXWskLaBtov3UYL9pgcGK3MCWu",
       "Size": 18},{
       "Name": "SomeName",
       "Hash": "QmbUSy8HCn8J4TMDRRdxCbK2uCCtkQyZtY6XYv3y7kLgDC",
       "Size": 58}],
   "Data": "Hello World!"}
```

Large files are chunked, hashed, and organized into an IPLD (Merkle DAG object).

Versioning is supported with commits that update the content and reference to the parent object.

Content Identifier CID allow to identify and address content/peers, contain the hash value of the referred object + multiple metadata fields. Obtained as concatenation of multiformat objects: objects containing self-describing data and allows an easy extension of protocols/data

A CID is formed by the concatenation of 4 multiformats: **multibase, cid-version, multicoded, multihash** each with different variables.

Multiformat Protocol: Collection of standards/protocols suitable to support future software modifications. Some scenarios where this is useful: current hash function may be broken in future and same for network protocols.

Enhance data format values with self-descriptions in the value itself, as small as possible, promote extensibility, human readable. Current:

- **Multibase**: self-describing base encodings - has a prefix b (base32), z (base58) and f (base16) followed by the base encoded data
- **Multicodec**: uniquely identifies the encoding method used to encode the data - identifies method to serialize the content itself, not related to the hash of the content, but to the content itself like plain JSON, Protobuf or Bencode
- **Multihash** starts with the type of hash function, the digest length and the the digest value - the goal is to write applications that future-proof their use of hashes, allow multiple hash functions to coexist. The hash function code is a multicoded.
- **multiaddr**: self-describing network addresses

There is a site https://cid.ipfs.io/ which is a CID inspector and gives us all the fields' values.

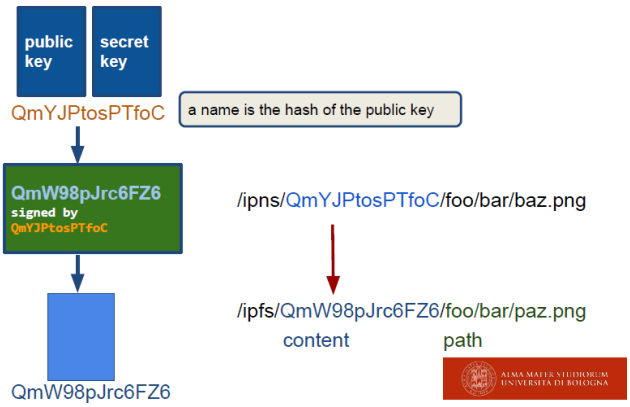
10.3.2 Naming

Problem: hash links to content in IPFS are not human readable and immutable (if we change the content, its hash changes, so the link should be changed)

IPNS: Interplanetary Naming System: a system allowing to always point to the the latest version of a content, human-readable and easy to remember like a DNS.

A name is the hash of the public key. nodeID is stable: this link does not change to find a content you retrieve it through this nodeId (via DHT). Mutable data: whenever you want to update the content, you create and sign a novel pointer. This new pointer is added to the DHT (data structure used for the lookup)

Imagine we made something like `ipfs add -r mywebsite`




```
const ipns = require('ipns');
const crypto = require('libp2p-crypto'); //for generating RSA keypair
function generateRsaKeypair(){
    //generating an 2048 bit RSA keypair
    crypto.keys.generateKeyPair('RSA', 2048, async(err, keypair) => {
    if(err){console.log('error ', err);}
    else{
        console.log("\nGenerated new RSA Keypair\n");
        createIpnsRecord(keypair);}});}
/*
Creating an IPNS record with a lifetime
ipns.create(privateKey, value, sequenceNumber, lifetime, [callback])
privateKey (PrivKey RSA Instance): key to be used for cryptographic
    operations.
value (string): ipfs path of the object to be published.
sequenceNumber (Number): number representing the current version of the
    record.
lifetime (string): lifetime of the record (in milliseconds).
callback (function): operation result.
*/
function createIpnsRecord(keypair){
    let sequenceNumber = 0;
    let lifetime = 1000000; //1000000 milliseconds
    //hash to my website
    let value = 'QmYVd8qstdXtTd1quwv4nJen6XprykxQRLo67Jy7WyiLMB';
    var recordData;
    ipns.create(keypair, value, sequenceNumber, lifetime, (err,
        entryData) => {
        if(!err){
            //Created new IPNS record
            console.log("\nGenerated new IPNS record\n");
            console.log(entryData);
            validateIpnsRecord(entryData, keypair);}});}
```

```
/* Creating an IPNS record with a fixed expiration datetime.
ipns.createWithExpiration(rsa, value, sequenceNumber, expiration,
    [callback])
privateKey (PrivKey RSA Instance): key to be used for cryptographic
    operations.
value (string): ipfs path of the object to be published.
sequenceNumber (Number): number representing the current version of the
    record.
expiration (Date): Date object.
callback (function): operation result.
*/
function createIpnsRecordWithExpiration(keypair){
    ipns.createWithExpiration(keypair, value, sequenceNumber, expiration,
    (err, entryData)=>{
    if(!err){
        validateIpnsRecord(entryData);}});}

/* Validate an IPNS record previously stored in a protocol buffer.
ipns.validate(publicKey, ipnsEntry, [callback])
publicKey (PubKey RSA Instance): key to be used for cryptographic
    operations.
ipnsEntry (Object): ipns entry record (obtained using the create
    function).
callback (function): operation result.
*/
function validateIpnsRecord(entryData, keypair){
    ipns.validate(keypair.public, entryData, (err)=>{
    //if no err then the validation was successful
    if(!err){
        console.log('\nIPNS Record Validation Successful\n');}});}

generateRsaKeypair();
```

10.4 Transporting the data

Libp2p a collection of peer-to-peer protocols for finding peers and connecting to them, for finding content and transferring it. For exchange, routing and network.

10.4.1 Exchange

BitSwap a protocol for data (blocks) exchange. Files are split into blocks, the smallest unit of transferable data - implemented by libp2p library.

Inspired to the principles of BitTorrent but not 100% BitTorrent. use a tit-for-tat like-strategy and get rare pieces first. The difference is BitTorrent uses a separate swarm of peers for each file and IPFS a single swarm for all data.

The single swarm may be monitored by the IPFS CLI interface, returns the multi-addresses of the peers of the single IPFS swarm.

The **basic strategy** is based on the exchange of previous blocks between two peers. when two peers exchange blocks they keep track of the amount of data they share (builds credit) and receive (build debt).

BitSwap Ledger: keep tracks of the accounting history between two peers. node Y decides to send blocks required by X on the basis of the blocks exchanged in the past: if X has credit Y will send the requested block, if X has a debt then Y may share or not depending on deterministic function where the chance of sharing becomes smaller when debt is bigger.

Share ratio is a measure of trust, between the peers: protects previously successful trade relations, even if one of the nodes is temporarily unable to provide values and choke relations that have greatly deteriorated until they improve. If the ratio exceeds 2, the probability quickly drops off and chocking free riders until a good balance is re-established

Do not penalize too much seeders that want to serve blocks without anything in change, and as in Bittorrent, favours clustering of peers with similar bandwidth.

The function implicitly incentives nodes to cache rare pieces, even they are not interested in them.

Bitswap structure:

- ledger: one for each connected peer
- peers: currently open connections
- need_list: multihash of the blocks needed by the current node
- have_list: multihash of the blocks that the current node can transfer to the others
- want_list: multihash of the blocks wanted by the current node, includes blocks needed by peer's peers

Bitswap protocol

Open message: open a connection with a node, the peers exchange their ledgers, each peer compares the received ledger with its own ledger - if they match, the connection is accepted otherwise it is refused and no connection is accepted for a certain period of time.

Send message: `send(want_list)` the sender advertises its want list and is executed when a connection is opened, after a randomized timeout and when the want list is modified. the receiver of the message checks if any requested block is in his have list and decide whether to transfer (share ratio).

Send Block `send_block(B)` transfers a block B requested from a partner, use the protocol specified with the multiaddr of the partner, the partner verifies the integrity of the data if ok block is removed from the recipient's need and want list. The ledgers are updated to keep consistency in the amount of bytes exchanged.

10.4.2 Routing

Use the hash of the file as a key to return the location of the file. Once the location is determined, the transfer takes place peer-to-peer as a decentralized transfer. Very similar to Bittorrent + Mainline DHT. Routing is an ongoing work not fully functional. Ideas from Kademlia.

There is no root, a peer only knows a small subset of others.

In Kademlia the identifier space is modelled by a tree and leaves are **identifiers** while internal nodes are identifier prefixes. The routing table of each node includes some references to nodes of this tree. Idea: the routing table of each node maintains at least one link to each area with a prefix that is the shortest string not prefix to the nodeID.

Keys are stored in the k nodes closest to the key Distance computed through XOR.

Closer nodes are located in sub-trees "close" to that of the key. Routing table includes some nodes in each of these sub trees. A query is sent to one of the known nodes in a sub tree. The contacted node provides the id of a node closer to the target. Each time we move to a sub-tree closer to the destination, we reduce the distance to the destination.

S/Kademlia a secure key-based routing protocol based on Kademlia. High resilience against common attacks. A set of security improvements with respect to Kademlia - limitation of free node identifier generation: use of PoW through cryptopuzzle and public key cryptography. Extension of Kademlia routing table by a sibling list - parallel look-ups over multiple disjoint paths.

Generating Id in S/Kademlia: generation of NodeId in Kademlia is critical: generate a particular ID: Eclipse attack and generate a huge amount of Ids: Sybil attack.

Possible solutions: use a central authority (can co-sign peers' certificates, control/limit sybils but needs a trusted party) or use Pow implemented through crypto-puzzles: static (generate key K so that c1 first bits of (double hash) H(H(K)) = 0) or dynamic (generate X so that c2 first bits of H(K x X) = 0 and increase c2 over time to keep NodeId generatione expensive)

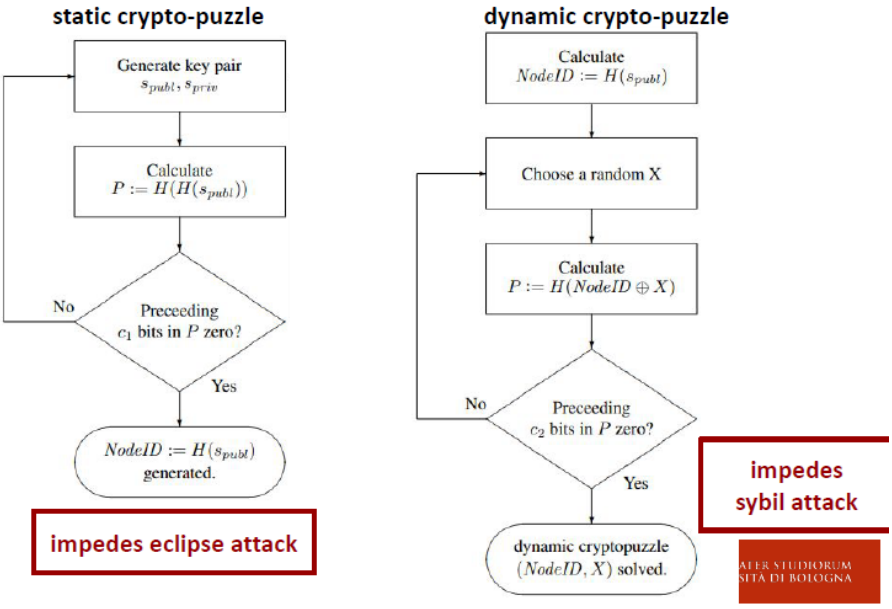
Inspired also on Coral Sloppy DHT: a peer to peer distribution network.

Implement a client caching scheme, DHT can be useful for indexing content providers, use the original URL as key and value is a list of nodes that cache that web page.

Sloppy DHT: similar to the Mainline Bittorrent Mainline DHT, but solves some issues of classical DHTs, mainly hotspots and poor data locality.

Hotspots: A single node responsible for a popular URL, stores a huge list of node references, quickly becomes overloaded with frequent updates also. Storing the content directly in the DHT is not a practical solution: data is stored at nodes where it is not needed, users not interested to store and burn bandwidth for unknown, content which they do not share.

Data locality: Only some DHT enables to route request through nodes with low latency, the last hops are essentially random, because few contacts are available in the routing table. A query may travel all the world before learning



that a neighbour caches the required content. An important issue for peer-to-peer CDN: DHT node may have considerably worse network connectivity than web servers.

Distributed Sloppy Hash Table: basic idea is each node stores only a maximum number of values for a particular key, values associated to a key are spread on multiple nodes.

When the number of values exceeds the maximum, spread the values across multiple nodes. Different from classical DHT replciation: replicate the key and all the associated values, no splitting of the value set.

Content insertion: store the data locally and insert a pair (content key, reference to the node) in the DHT, approach the responsible node through Kademlia routing, if closest node is "full" backtrack one hop on the lookup path and store on the first free node. Nodes are grouped into clusters based on locality and level-0 cluster covers the full net. Nodes at level i are subset of nodes at level i-1.

10.4.3 Network

With LibP2P. Each peer controls a pair (public, private key) - the key pair allow peers to establish secure communication channels with each other. PeerID: cryptographic hash of a peer's public key like contentId and may be also encoded in structures called multiaddr.

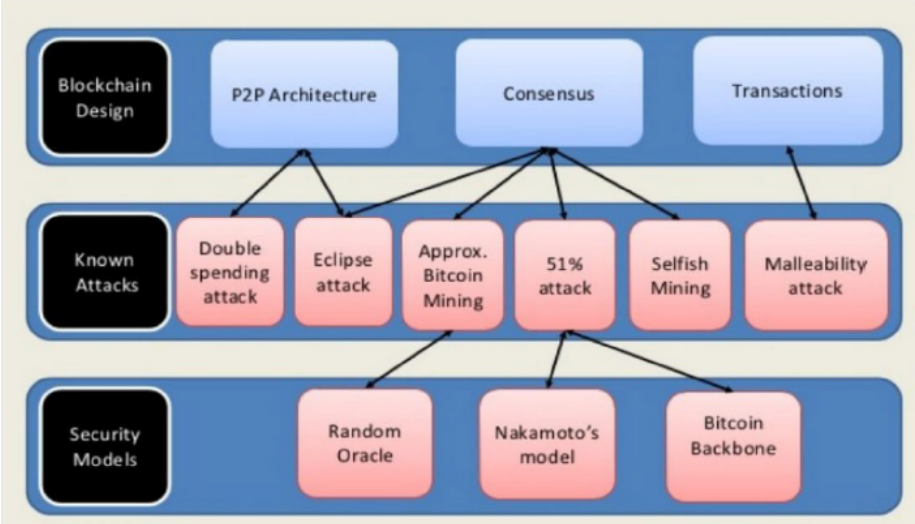
Multiaddr: support addresses for any network protocol, self-describing, human-readable and efficient machine-readable representations. A recursive encoding where nested protocols are admitted.

/ip4/127.0.0.1/udp/1234 encodes two protocols: tells to use the 127.0.0.1 loopback address of the IPv4 protocol and tells to send UDP packets to port 1234

/ipfs/QmYyQSo.....hx5N uniquely identifies an IPFS node and use the IPFS registered protocol and the multihash of the IPFS node's public key.

/ip4/7.7.7.7/tcp/4242/ipfs/QmYyQSo....hx5N suppose that node's public IP is 7.7.7.7 and TCP connection port is 4242 combine the "location multi-addr" and the "identity multiaddr" and produce a new multiaddr containing both key pieces of information

11 Blockchain attacks



Can miners steal bitcoins of transactions belonging to the block they have created? This would require to create and insert a valid transaction that spend those coins. To do this, the miner would forge the bitcoin owners' signatures. Impossible if a secure digital signature scheme is used and cryptography makes this not happen.

Double spending: A is a miner and validates a block B that includes both transactions, Other nodes they check the validity of B and reject it, the attack does not succeed, despite A has solved the proof-of-work.

A proposes next block, and deliberately forks the chain to perform double spending. Ignores the block that contains the payment to B, proposes a block including a pointer to the same previous transaction outputs, the transaction transfers to herself of the same bitcoins spent with B.

Honest nodes will accept the longest valid branch. The transaction to A' will be orphaned. If it succeeds, B must be careful to not deliver the service to A until he is sure that the transaction will be included in the longest chain -

wait before delivering the service. So the system is only vulnerable to a double spend attack near the end of the chain, which is why it's recommended to wait for several blocks before considering received money final.

Double spend probability **decreases exponentially with the number of confirmations** - most common 6 confirmation.

Recap: Protection against invalid transactions is cryptographic but enforced by consensus, Protection against double-spending is purely by consensus. You're never 100% sure a transaction is in consensus branch.

Can we penalize the node that created the node A'? No

What if I control 51% of the networks hashing power? All bets are off, attacker can control all future blocks, monopolize creation of new coins, double spend, deny arbitrary transaction.

A 51% attacked cannot steal coins, can suppress transaction from the blockchain but not from the P2P network. Cannot change the block reward. He can destroy confidence in Bitcoin.

Default strategic decisions: include any transaction in a block above the minimum tx fee, mine the longest valid chain, between colliding blocks choose the first block heard, announce new blocks immediately after finding them.

11.0.1 Forking attacks

Certainly possible with $\alpha > 0.5$, may be possible with less, is detectable, might be reversed and might crash exchange.

Bribing attack (game theory): changes the equilibrium of a mechanism by paying the players, conditioning on their strategy choices. Instead of acquiring mining power up to $\alpha > 0.5$, you rent it: creating a mining pool with highest shares, insert large tips in the blockchain.

11.1 Block-withholding attack - Selfish Mining

here you don't announce blocks and try to get ahead. Other miners waste their effort and you get a bunch of time to mine by yourself with higher expected profits - never seen in practice, only theoretical.

11.2 Blacklisting

via Punitive Forking

Objective: Censor the Bitcoin addresses owned by certain people, say Alice, and prevent them from spending any of their Bitcoin. Strategies:

1. Refuse to mine any chain with a transaction from Alice - doesn't work unless $\alpha\tilde{1}$ and can only cause delays and inconveniences.

2. Announce to the world that you will refuse to mine any chain with a transaction from Alice - with $\alpha < 0.5$ you'll be behind, but works if $\alpha > 0.5$

Third startegy via Feather forking: Announce that you will attempt to fork if you see a block from Alice, but you will give up after a k confirmations. With k=1 you will give up after 1 confirmation - chance is α^2.

But other miners are now aware that their block has a α^2 chance of being orphaned. Unless Alice pays $\alpha^2 * BlockReward$ in fees for his transaction, other miners will mine on the malicious chain.

Feather Forking is good for freezing individual bitcoin owners and enforcing a minimum transaction fee - when block rewards will be over.

11.3 Pool Cannibalization

Cannibalizing Pools: Distribute your 1% equally among all other pools, withhold valid blocks. Submit valid lower-difficulty shares but do not submit shares that match full Bitcoin blocks. Rewards will still be received. This reduces the other pools' income by 1% but attacker would still continue to gain 1% of what the pools did earn. Undetectable unless statistically significant. More profitable to cannibalize pools than mine honestly.

The miner's dilemma: If pool A chooses to attack pool B, pool A gains revenue, pool B loses revenue - Pool B can in turn attack pool A and gain more revenue.

Attacking is a dominant strategy in each iteration. Both attacking each other, is a Nash Equilibrium. Both will earn less than they would have if neither of them attacked. If the pools agree not to attack, both benefit in the long run. This is an unstable situation since on a practical level you can attack another pool anonymously.

To defend from all this you can dummy block signature, fork-punishment rule, uniform tie breaking, unforgeable timestamps, weighted fork resolving policy.

11.4 Eclipse attack

An adversary controlling a sufficient number of IP addresses can monopolize all connections to and from a victim bitcoin node.

Gossip protocols: fixed probability, probabilistic broadcast, dandelion dandelion++.

Dandelion consists of two phases: **Stem phase** = the message is sent to just one neighbor that is selected at random. **Fluff phase** = the message is broadcast, all the neighbors receive it.

Dandelion ++ : Fail-safe mechanism against Denial of Service attacks = if a node receives a message during the stem phase and it does not get it back in the fluff period, then such a node will start the fluff phase by itself, by broadcasting the message - used by Zcoin, Monero.

12 Anonymity

Anonymity = pseudonymity + unlinkability

Pseudonymity: Bitcoin addresses are public key hashes rather than real identities.

Unlinkability: different interactions of the same user with the system should not be linkable to each other.

If anyone is ever able to link your Bitcoin address to your real world identity, then all of your transactions — past, present, and future — will have been linked back to your identity.

Zooko's triangle: only two of these three can be achieved: Human meaningful, decentralized and secure.

Many Bitcoin services require real identity to transact - online wallet services, exchange.

Linked profiles can be deanonymized through side channels. Shoudl be hard to link - different addresses of the same user, different transactions made by the same user, the sender of a payment to its recipient.

All transactions are public and traceable: no anonymity leads to no privacy. Concern: anonymous cryptocurrencies can be used for money laundering or other illegal activities.

New regulations on financial services to ensure customer privacy and secure online and mobile payments.

General Data Protection Regulation (GDPR): Forces European companies (and banks) to rethink how to store and manage personally identifiable information.

Revised Payment Service Directive (PSD2): Directive to regulate payment services and payment service providers

EU Fifth Anti-Money Laundering Directive (AMLD5): regulation for financial institutions. There are obliged entities: Entities required to monitor user activities and report to authorities potential fraudulent activities. Some of these are wallet providers.

Obligations are: registration, know-your-customer, risk assessment and monitoring ans suspicious transaction reporting.

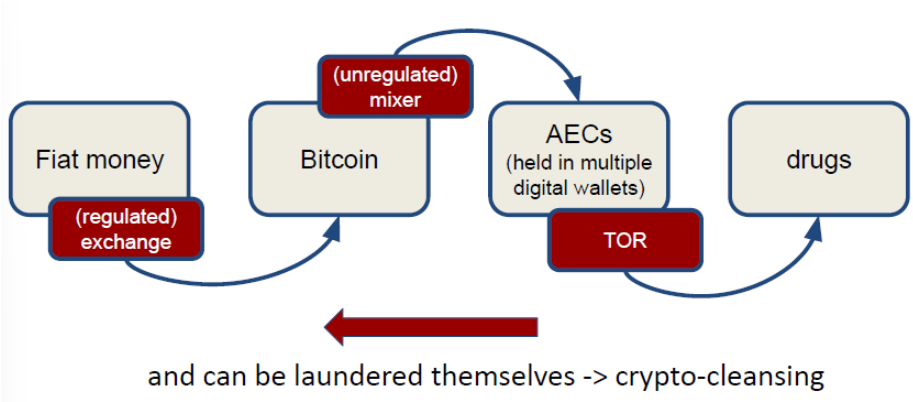
Anonymity Enhanced Currencies (AECs): Monero, Dash and ZCash.

12.1 Best practices

Create a new pseudonym for every transaction and let your wallet manage all these addresses.

Shared spending is evidence of joint control: Anyone can infer that the two inputs are most likely under the control of the same user. Or send to two outputs one of which sends back to myself.

Anyone who knows your public key can look on the blockchain and see the dates and amounts of transactions coming in and out of your account. From this data, they might be able to put together a pattern of transactions from



which they could deduce your activities. ML to decode spending patterns on dark-market sites.

De-anonymize Bitcoin:

- Tagging by transaction: If you can cluster the address with others → you can tag that entire cluster with the service provider's identity
- Via service providers: if law enforcement wants to identify a user, they can ask service providers that interact with some individuals
- Carelessness: people often post their Bitcoin addresses in public forums
- Network-layer deanonymization: In the P2P system, "the first node to inform you of a transaction is probably the source of it." If you have the IP address, you can try to unmask the identity of an individual

12.2 Mixing

Also Tumbling services for Bitcoin. CoinShuffle, MixCoin, CoinSWap and TumbleBit.

If you want anonymity use an intermediary. Users send coins to an intermediary and get back coins that were deposited by other users.

The coins that you withdraw won't be the same as the coins you deposited. Online wallets provide a measure of unlinkability. Limits to using online wallets for mixing: Most online wallets don't actually promise to mix users' funds; instead, they do it because it simplifies the engineering, Even if they do mix funds, they will maintain records internally that link your deposit to your withdrawal, Reputable and regulated services will also require and record your identity.

Dedicated mixing services: promise to not keep record and do not require identity. They perform a swap between an intermediary address but you have to trust them (most of them are reported to be stealing bitcoins)

Uniform transactions: All mixes should agree on a standard chunk size value, This increases unlikability and the anonymity set, Makes easier to use a series of mixes without having to split or merge transactions.

12.3 Decentralized mixing

Get rid of mixing services and replace them with a P2P protocol, users can mix their coins. No bootstrapping problem: Users don't have to wait for reputable centralized mixes to come into existence.

Theft impossible: The protocol ensures that when you put in bitcoins to be mixed, you'll get bitcoins back of equal value with better anonymity.

Coinjoin: Different users jointly create a single Bitcoin transaction that combines all of their inputs - The signatures corresponding to each input are separate from and independent of each other.

Each user supplies an input and output address, Together they form a transaction with these addresses, Order of input and output addresses is randomized so as An outsider will be unable to determine the mapping between inputs and outputs. Participants check that their output address receives the same amount of Bitcoin that they are inputting. Once they have confirmed this, they sign the transaction.

Algorithm: Find peers who want to mix, exchange input/output addresses, construct transaction, send the transaction around, collect signature - before signing each peer checks if his output is present and broadcast.

Problems: how to find peers? Peer know your input-output mapping (worse problem). Denial of service: a peer can participate and refuse to sign in the second phase or a peer can try to take the input that it provided to its peers and spend it in some other transactions.

For finding peers we can use an untrusted server (watering-hole) which allow users to connect and group together.

For peer anonymity we can use Tor: so we exchange input, disconnect and reconnect tor and exchange outputs or use some special-purpose anonymous routing protocol.

For denial of servise we can impose a cost to participate in the protocol: proof of work, proof of burn (Technique to provably destroy a small quantity of bitcoins that you own) or server kicks out malicious participants. Otherwise use cryptographic ways to identify a non-compliant participant and kick them out of the group.

Merge avoidance for high level flows: Instead of a single payment transaction, Receiver provides multiple output addresses, Sender avoids combining different inputs.

Generally this mitigates the problem of high-level flows: an adversary might not be able to discern a flow if it is broken up into many smaller flows that aren't linked to each other. It also defeats address clustering techniques that rely on coins being spent jointly in a single transaction.

12.3.1 Anti Money Laundering

With social net analysis + machine learning like clustering. We use some heristics to cluster addresses. Based on random walks on the bitcoin transaction graph to compute features based on the distance to illicit transactions