

Blockchains, an opiated overview

Diego Olivier Fernandez Pons
TZ Ventures - Tezos Combinator

Simão Melo de Sousa
DIUBI, C4, Lab Release, Nova-Lincs



Introduction

a blockchain is a **distributed ledger technology** that runs upon a **peer to peer network**, runs **cryptographic and distributed** algorithms and protocols in order to ensure

- **decentralization** (and by this, enforces openness)
- **immutability** (and by this, allows total auditability)

it's a highly complex software infrastructure

blockchains can be qualified by being :

- **public** vs. **private** : depends on who is authorized to make transactions
- **permissioned** vs. **permissionless** : depends on who is authorized to process transactions

most popular blockchains you may know are **public** and **permissionless**
(our focus here)

usually corporate blockchains *are not*

What problems do blockchains solve ?

the original purpose of blockchain technology was to solve the problem of **electronic cash**

- **1989** DigiCash (David Chaum)
- **1996** e-Gold (Jackson and Downey)
- **2009** Bitcoin (Satoshi Nakamoto)

from that point on, the purpose of blockchain technology has shifted to solve the problem of **enforceable electronic contracts**

- **2015** Ethereum (Vitalik Buterin)
- **2018** Tezos (Arthur Breitman)

Many blockchains try to solve a variant of these problems :

- **Remittance** (Stellar)
- **Financial transfers** (Ripple)
- **Fast payments** (Litecoin, OmiseGo)
- **Gambling applications** (EOS)

Electronic cash

electronic money has existed since the 60s with the internationalization of the banking system

- **1872** Wire transfer by telegraph
- **1958** Bank Americard (later Visa)
- **1966** Interbank Card Association (later Mastercard)
- **1977** SWIFT (Society for worldwide interbank financial telecommunication)
- **1978** Electronic Funds Transfer Act (USA)

creating electronic money requires achieving the same properties as paper money (e.g. **legal tenderness** and **non duplicability**)

the first issue to address is to ensure money cannot be duplicated

electronic payment systems (online banking, VISA, etc) achieve non-duplication by

- giving you an **account** that represents how much money you own in their system
- keeping track of payments (in and out) in a **database**
- allowing merchants to **check** in the database whether there is enough money for a transaction
- adding an **audit** system and an **insurance** to cope with any issues that may arise

however, while these systems achieve **non-duplication**, they **fail** to create a form of electronic money whose properties (**non duplication**) is **independent** of external trusted parties (auditors, banks). that means the failure of one entity (e.g. bank) results in the loss of the funds.

the critical property that supports non-duplication in electronic money is the **ability for everyone to check there is enough money for a transaction**

proponents of blockchain understood this doesn't need to be done in a privately owned database

it can be done in a **public distributed database**

Cryptocurrencies - distributed ledger

a **distributed ledger** is a network of computers that all keep track of the amount of money in each account all transactions that involve a bank account (in and out)

anyone can query any node in the system to check if there is enough money for a transaction

each node can be owned by different private entity (bank, company, etc.)

each node has a synchronized copy of the whole ledger: **independence**, that is, if a node fails (e.g. bankrupt) there is no money loss

thus, from distributed to **decentralized**

we need to ensure the distributed system doesn't allow **double spending**, just like paper money and electronic money

avoiding double spending is more difficult in distributed ledgers because of the **latency** of the peer-to-peer network (delays that happens in data communication over a network)

(more on this, later)

The rise of digital contracts

There are many reasons paper contracts may not be enforced

- one party is in charge of enforcement and refuses to do so
- too small amount to complain to a judge
- lack of evidence
- ambiguous terms

typical scenario: Alice wants a 15€ Pizza from Bob. Alice pays 15€ to Bob, Bob send the pizza to Alice

no matter what means of payments is used (cash, electronic money, in advance, upon reception), in all cases one of the parties is in charge of the execution of the contract and **can refuse to do so**, with little recourse for the other party

there is a need for **contracts that can always be enforced**

while creating an electronic currency, blockchain proponents also created a **programmable currency** that allows writing electronic contracts, also named **smart-contracts**

- **1997** Nick Szabo introduces the idea of contract enforced not by law, but by computers

code is law (Lawrence Lessig, Jan 2000, Harvard Magazine)

- **2009** Bitcoin script allows writing simple contracts like multisig, escrows, time locks, oracles or lotteries
- **2015** Ethereum popularises smart-contracts with its Solidity language that is a variant of JavaScript
- **2017** Tezos introduces Michelson a low-level stack based language for smart-contracts, amenable to formal verification
- **2018** several emerging blockchains advocate the use of WebAssembly as the low-level programming language for smart-contracts

blockchains enforce contracts the same way they execute transactions

- contracts are **programs** that define the **conditions that need to be met** for a transfer of funds to happen
- contracts are denominated in a **crypto-currency**
- a transaction is an **unconditional contract** that says "transfer X from A to B"

Enforceable electronic contracts - limitations

not all paper contracts can be transformed into smart-contracts

- some conditions cannot be translated into computer programs
- we are still missing infrastructure to translate all contracts

for instance, in an e-commerce smart-contract, blockchains need connections to **external services like package delivery services** to know whether a package was delivered or not

Enforceable electronic contracts - limitations

recall that a particular smart-contract to be executed in a particular blockchain infra-structure is a **program** expressed in a **domain specific programming language** that targets a specific **execution environment**

many things can go wrong in a smart-contract

- humans are not good at details
- computers need every process to be described to them down to the smallest details

it is surprisingly (?) difficult to get **right** smart-contracts, **right**

- correctly enforcing adequate business rules is hard
- many smart-contracts have mistakes
- many smart-contracts frameworks are poorly designed

... which may result in losses of funds

“smart-contracts as programs” paradigm

It never hurts to insist

a bug is an open door for a robbery in plain sight

and **bugs can be everywhere**, and there are lot of places to hide

- in the software stack of the underlying blockchain architecture (VM, node, etc.)
- in the execution (and its economical) model
- in the smart-contracts programming languages design or (missing...) semantics
- in the compiler
- in the (confused, unprepared, overconfident, etc.) head of the programmer
- etc.

one would expect blockchain designers to have taken advantage of the valuable lessons from the history of software engineering and the best recipes from programming language research, but ...

Ethereum Wiki

major-issues-resulting-in-lost-or-stuck-funds

TABLE OF CONTENTS

- Major issues resulting in lost or stuck funds
 - The DAO
 - Ethereum Classic replay attacks
 - Parity multisig library contract issue 1
 - Parity multisig library contract issue 2
 - Cases covered in EIP-155
 - "The" profits enforcement (Banking42)
 - Ethereum.JS Pending Bug
 - RCEinfo
 - Off By One
 - Contracts Deployed With No Code

VIEW THIS DOCUMENT

LAST EDITED BY: eth-wiki 8/11/2020

major-issues-resulting-in-lost-or-stuck-funds

Major issues resulting in lost or stuck funds

Ethereum has had expensive bugs, such as the following:

The DAO

3,600,000 ETH stolen (recovered through hard fork)

To read about [the DAO vulnerability](#), press **Ctrl** + **F** in the linked Wikipedia article and search for vulnerability.

Ethereum Classic replay attacks

Ethereum Classic replay attacks also occurred ensuing the DAO hard fork, which were subsequently fixed by [EIP-155](#).

Parity multisig library contract issues 1

[1](#) [2](#) [3](#) [4](#)

155,057 ETH stolen

The first Parity multisig library bug was fixed in [this pull request](#).

On Wednesday 19th July 2017 a bug found in the multi-signature wallet ("multi-sig") code used as part of Parity Wallet software was exploited by parties unknown... The bug was in a pair of extremely sensitive functions designed to allow the set-up of "multi-sig" wallets in the Parity Wallet software. The functions should have been protected in order that they be usable only in one specific circumstance, as the contract was being created. However, they were entirely unguarded, which allowed the attacker to reset the ownership and usage parameters of existing wallets arbitrarily.

Parity multisig library contract issue 2

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#)

513,736 ETH stuck

Following the fix for the [original multi-sig vulnerability](#) that had been exploited on 19th of July (function disability), a new version of the Parity Wallet library contract was deployed on 20th of July. Unfortunately, that code contained another vulnerability which was undiscovered at the time - it was possible to turn the Parity Wallet library contract into a regular multi-sig wallet and become an owner of it by calling the [setOwner](#) accidentally on 06th Nov 2017 02:33:47 PM -UTC and subsequently a user deleted the [library turned into wallet](#), wiping out the library code which in turn rendered all multi-sig contracts unusable and funds frozen since their logic (any state-modifying function) was inside the library.

All dependent multi-sig wallets that were deployed after 20th July functionally now look as follows:

```
contract Wallet {
  function () payable {
    deposit(...)
  }
}
```

This means that currently no funds can be moved out of the multi-sig wallets.

which paper contracts can be transformed into smart-contracts is a **programming theory** problem

making sure the smart-contract corresponds exactly to the paper contract is a **formal verification** problem.

The economy of Blockchains

The economy of Blockchains

Economic models for blockchain developers

the financing of infrastructures faces two issues

- **the tragedy of the commons** : everyone ones wants to use the infrastructure but nobody wants to pay for it
- **monopolies** : once you find someone that agrees to pay for the infrastructure, they want to (ab)use it in order to generate as much profit as possible

generic blockchain frameworks are software infrastructure

as such they face the same challenges around financing and control as other infrastructures

- financed ideally by a large amount of people
- controlled by neutral entities, ideally by the users
- operation and evolution paid by taxes

in Tezos, the following choices were made

- **financing** by pre-sales of collateral at a discounted price
 - the collateral allows you to start a block production business
- **control** by community vote
 - decisions related to the blockchain are taken by a community vote, with voting rights proportional to the amount of tokens held
 - no entity has the authority to control Tezos
- **operation and evolution** paid by inflation
 - block producers operate the network and are paid by inflation
 - evolution proposals submitted to the community are voted and can include an invoice in tokens

(collateral = the cost of operation, more details later on)

each additional user of a service adds value to every other user

generic blockchains like all infrastructures **have strong network effects**

there is a lot of value in everyone using the same blockchain platform (economies of scale for businesses, etc).

in open-source software, a **fork** is a variant of a software that is developed independently from the people developing the original version of the software

it allows trying new ideas on top of an existing software, without inheriting the constraints of the original team like regularity of the releases, conservative release of new features, business and technical processes, legacy constraints, etc.

forks in blockchain **work against the network effect**

the problem is the absence of **preference signaling**

- I may prefer the benefits of the network effect over those of the fork
- I may prefer the benefits of the fork over those of the network effect

... usually, the network effect has the advantage

a possible solution: amendment voting in Tezos

the voting in Tezos

- only applies to technical changes that requires coordination (e.g. protocol but not the software for the node)
- can include a reward in tokens for the proponent
- is 3 month long, divided into 4 steps

The economy of Blockchains

Economic models for blockchain nodes ?

nodes are owned by individuals / companies.

maintaining a node incurs costs

- hardware (PoW)
- bandwidth (PoW, PoS)
- electricity (PoW)
- stake / tokens (PoS)

these companies and individuals need to make a profit to continue offering their services

one problem that needs to be solved is **how to control that nodes are doing their work properly 24/7**

creating an organisation to do that control would require

- different currencies
- taxes in multiple countries
- employees all around the world

one key idea in blockchain is to **embed the payment of the nodes within the work they have to do** :

in each block produced, the nodes should include their fees

- no more need for control
- payment proportional to the amount of work being done

(De)incentivisation of economies of scale ?

another problem to be addressed is the **incentives model** for **economies of scale**

should there be

- incentives to have nodes with large collateral
- incentives to have a large number of nodes with smaller collateral
- no preference with respect to the size of the collateral or the number of nodes

ideally one would like at the same time nodes with large collateral (**as it secures the transactions in the network**)

but also a large amount of independent nodes (**to avoid collusion**)

blockchains have decided to operate with a **global economic model**

- nodes are **paid** every time they **create a new block**
- **node selection** to produce blocks is **proportional to the collateral**
- payment is done in a **global electronic currency**
- payment per block produced is a **fix value per block** and a **variable value for each transaction**, it doesn't change per geography

the limitations of the global model are progressively appearing

- difference between local costs and global payments makes **operation in cheaper countries more profitable**,

hence a concentration of operators in some countries

- e.g. Bitcoin miners in China
- having a separate currency for the system
 - creates operational risk due to fluctuation of its value (e.g Bitcoin)
 - attracts speculators, scammers and hacker
 - concerns regulators

a side note on the economy of smart-contracts

is it clear that in this setting, deploying, executing a smart-contract is not free?

it costs money!

evaluating/anticipating these costs, minimizing them, producing the best low-level smart-contracts that make the job **is clearly challenging!**

again, this resorts to **programming language and compiler theory.**

The economy of Blockchains

Economic models for blockchain businesses ?

an application like Google Docs is a distributed application but runs on a private platform that provides

- access from any part of the world
- instant data replication
- payment

can we have applications that don't rely on a specific platform vendor ?

two views of the blockchain

- **maximalists view** : the blockchain is the new internet
 - all applications are possible and should be on blockchain
- **minimalists view** : the blockchain allows automating some simple and specific tasks
 - payment systems
 - time-stamping and authentication of data
 - automation of simple contracts

The blockchain is the new internet



Figura: <https://www.youtube.com/watch?v=j23HnORQXvs>

The blockchain is the new internet

The screenshot shows the Dfinity website homepage. At the top left is the Dfinity logo, and at the top right are navigation links for FAQ, Foundation, Ecosystem, Careers, Media, Sodium, and a Developers button. The main heading is "Build On The Internet. No Dependencies." Below this is a paragraph explaining that the Internet Computer extends the public Internet to host websites, software systems, and DeFi services, created by independent data centers worldwide running the ICP protocol. There are two buttons: "START CODING" and "LEARN MORE". A central white box titled "The Internet Computer in a timeline" contains the sub-heading "The evolution of the internet". To the right, a text block states "Tech Giants have hijacked the internet. It's time for a reboot." Below this are three feature cards: "DFINITY aims to replace the \$3.9 trillion dollar legacy IT stack", "SDK" (Build secure software and open internet services) with a grey infinity symbol, and "Motoko" (A powerful new programming language) with a colorful eye-like logo.

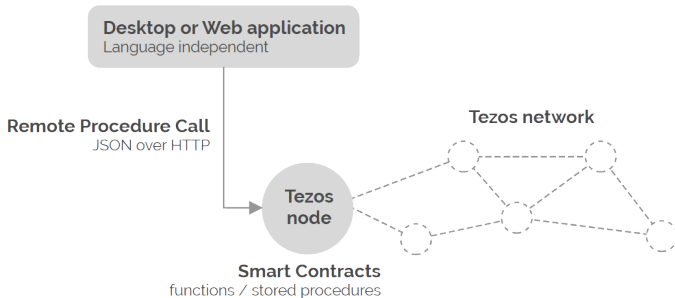
Figura: Dfinity, the internet computer

The blockchain is the new internet

but... existing technology doesn't permit having all computing power on the cloud/blockchain and all applications using it.

a simple application like Cryptokitties almost halted Ethereum due to its resource consumption

A blockchain is a web-service



blockchains are very slow and limited

- 95% of the application should be outside of the blockchain (**on-chain/off-chain**)
- only the 5% critical part of the application should be on the blockchain
 - payment
 - transfer of property
 - etc.

Financial Technology & Automated Investing

FINANCIAL TECHNOLOGY

AUTOMATED INVESTING

▶ BLOCKCHAIN TECHNOLOGY

FINANCIAL TECHNOLOGY & AUTOMATED INVESTING > BLOCKCHAIN
TECHNOLOGY

Blockchain-as-a-Service (BaaS)

By [JAKE FRANKENFIELD](#) | Reviewed By [JULIUS MANSIA](#) | Updated Jul 12, 2020

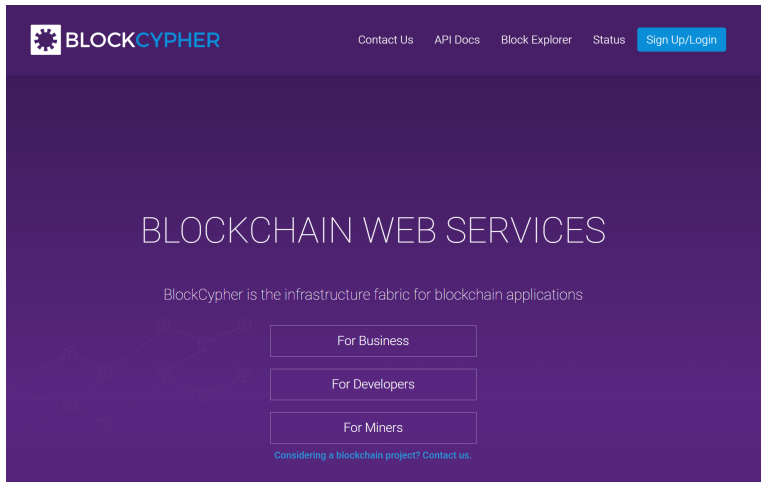
What Is Blockchain-as-a-Service (BaaS)?

Blockchain-as-a-Service (BaaS) is the third-party creation and management of cloud-based networks for companies in the business of building blockchain applications. These third-party services are a relatively new development in the growing field of blockchain technology. The business of blockchain technology has moved well beyond its best-known use in cryptocurrency transactions and has broadened to address secure transactions of all kinds. As a result, there is a demand for hosting services.

KEY TAKEAWAYS

- Blockchain-as-a-service (BaaS) refers to third-party cloud-based infrastructure and management for companies building and operating blockchain apps.
- BaaS functions like a sort of web host, running the back-end operation for a block-chain based app or platform.
- BaaS may be the catalyst that leads to the widespread adoption of blockchain technology.

A blockchain is a web-service



The screenshot shows the BlockCypher website homepage. The header features the BlockCypher logo on the left and navigation links for 'Contact Us', 'API Docs', 'Block Explorer', 'Status', and a 'Sign Up/Login' button on the right. The main content area has a dark purple background with the text 'BLOCKCHAIN WEB SERVICES' in large white letters. Below this, it states 'BlockCypher is the infrastructure fabric for blockchain applications'. To the left of three service boxes is a faint, stylized world map. The three boxes are labeled 'For Business', 'For Developers', and 'For Miners'. At the bottom, there is a link: 'Considering a blockchain project? Contact us.'

BLOCKCYPHER

Contact Us API Docs Block Explorer Status [Sign Up/Login](#)

BLOCKCHAIN WEB SERVICES

BlockCypher is the infrastructure fabric for blockchain applications

For Business

For Developers

For Miners

[Considering a blockchain project? Contact us.](#)


A blockchain is a web-service

introducing the district0x Network' and 'How it works'. In the center is the Ethereum logo. Below it is the headline 'The future of work is now' and the sub-headline 'hire or work for Ether cryptocurrency'. There are four buttons: 'BECOME A FREELANCER', 'BECOME AN EMPLOYER', 'FIND WORK', and 'FIND CANDIDATES'. At the bottom, there are three icons with text: a '0%' icon for '0% service fees', a network icon for 'Fully decentralised on blockchain', and a 'FREE' icon for 'No restrictions Free membership'."/>

Ethlance

Participate in Ethlance's governance processes: [introducing the district0x Network](#)

How it works



The future of work is now

hire or work for Ether cryptocurrency

BECOME A FREELANCER BECOME AN EMPLOYER

FIND WORK FIND CANDIDATES

0% 0% service fees Fully decentralised on blockchain **FREE** No restrictions Free membership

Conclusion

Smart-contracts, Blockchains and their Challenges

blockchains and enforceable electronic contracts are **very young technologies** with **many challenges and research questions** yet to be tackled.

they have enough maturity to be used, with success, in a big variety of applications

but, **despite of all the buzz**, they are **not ready** for all the cases we would like to use them

neither they are the solution for every problem we think blockchains can solve

nevertheless blockchain and smart-contracts are subject to evolution and to active research

to my point of view, there are three lines of future works, evolution about blockchain related technologies :

1. fundamental research
2. applied research
3. new blockchain applications and dissemination

1. fundamental research

- the duality between computation and the verification of a computation in a smart-contract
- fundamental research and tooling for better consensus algorithms (**see “extra material”**)
- new incentive and economical models for blockchains
- formal verification and programming language theory for blockchains and smart-contracts

2. applied research

- scalability issues (speed and number of transactions, size of the blockchain, etc.)
- security and privacy issues
- low-level smart-contract optimization
- smart-contract software architectures (layer 4, common formats, VMs etc.)
- software engineering for smart-contracts based software systems
- the duality *off-chain* versus *on-chain* in smart-contracts based software systems

- smart-contracts
 - interaction and reactive behaviour : **Project SMARTY**
 - functional correctness and security : **Whyson, Project FRESCO**
- cryptographic primitives, formally designed privacy mechanisms: bullet proofs, new elliptic curves for extended zero knowledge proofs, multiparty computation... **Prefab project but also “ask Prof. Paul Crocker”**
- virtual machine : **Project COPES and project CRISP**
- better consensus algorithms : **Project COPES and project CRISP**

- new languages, new compilers for smart-contracts : **Projects SMARTY, FRESCO and Prefab**
- optimization, program transformations : **projects FACTOR, LEAF and Prefab**
- stronger type systems (see Ligo, Archetype)
- active security analysis and profiling, such as static analysers : **TezCheck, Project FRESCO**
- new intermediate representation of smart-contracts that open integration of new tools such as security managers, security runtime monitoring, secure execution on credential : **Tezla and project FRESCO**

Economical models and privacy layers for the blockchain

Let's also cite the work of a PhD student of the Release team who works on new economical models for blockchains and how new privacy enhancing technologies can enable these new models

Extra Material

where all the fun is!

Distributed Ledger Technology

Distributed Ledger Technology

Distributed Databases

distributed databases are **everywhere**

- search engines (Google, Bing)
- online edition of documents (Google Docs, Office 365)
- bank accounts
- RAID storage

the architecture of distributed databases is extremely diverse

- banking systems
 - don't keep a copy of all data in each node : your bank account only exists in the database (node) managed by your bank
 - don't expect nodes to fail in an irrecoverable way
 - work with very heterogeneous nodes (each bank)
- RAID
 - mirrors all the data on multiple hard drives (RAID ≥ 1)
 - expects nodes (hard drives) to just fail and be replaced with new (blank) ones

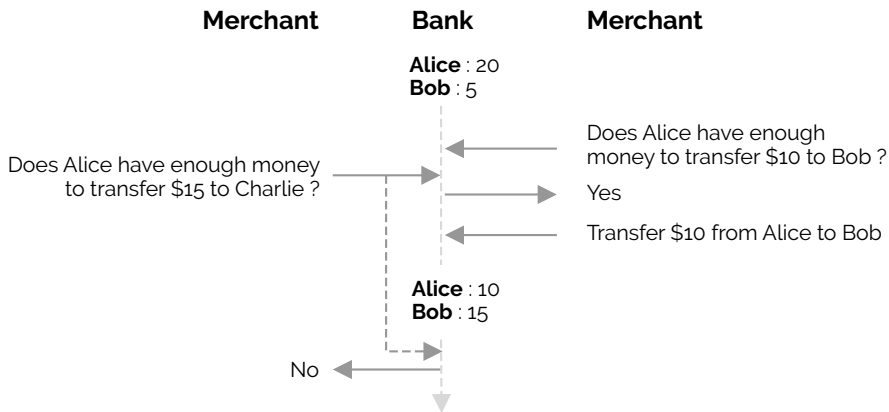
problems in distributed databases

- **consistency** : the system has to always return the latest version of the data
- **availability** : the system always answers to requests
- **reliability** : the request to the system have to succeed
- **capacity** : the system has to accommodate a high volume of data and requests
- **performance** : the system has to answer quickly

keeping data consistent in a database requires the **serialization of operations**

- in a (central) database, a **lock** is put on the data, allowing only one process to access to it
the process guarantees the sequential and consistent handling of the data
- in a distributed database, a **leader** is elected for a limited time, to manage the database in a sequential way

Distributed databases - Serialized access



consistency of simple arithmetic operations (bank accounts) requires serialization

the problem consisting in all nodes agreeing on a piece of information (who is the leader, what transaction to execute, etc.) is called the **consensus problem**

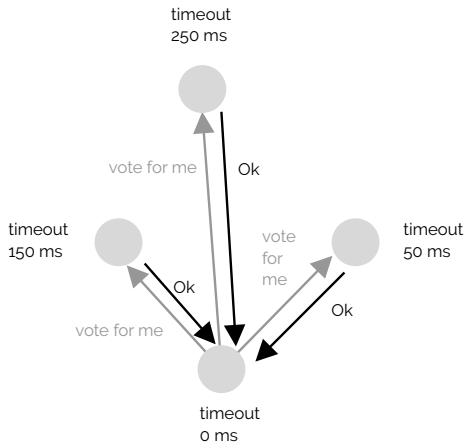
many consensus algorithms have been created

- BFT (Lamport, 1982)
- Paxos (Lamport, 1989)
- pBFT (Castro & Liskov, 1999)
- Raft (Ongaro, 2014), simplification of Paxos
- Tendermint (Buchman, 2016), simplification of pBFT

most consensus algorithms for databases are based on **voting**

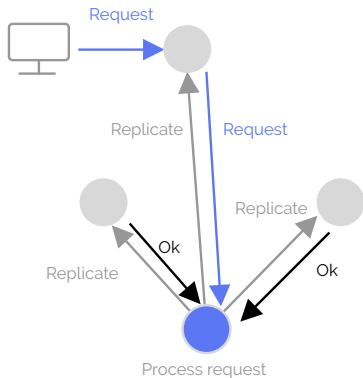
Raft is a typical, simple and popular consensus algorithm for databases

- a node that wants to perform an operation on the database sends it to the current leader for processing. After some time it will receive confirmation the operation was executed
- If there is no current leader, the node contacts other nodes asking to be elected the leader



election of a leader in Raft.

if there are multiple nodes requesting to be elected and none gets the majority, the nodes wait a random amount of time before asking again



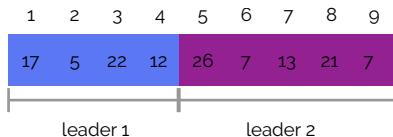
replication in Raft

client requests are saved uncommitted by the leader, sent to all peers for confirmation, and committed upon reception a majority of OKs. Then client then gets a confirmation

log structure in Raft

Raft synchronizes an append-log entry

messages have of the form $log[i] = v$ which makes them idempotent



if messages received by a node from the leader don't match their current log (e.g. the node crashed and recovered), they work together to resynchronize

recovery in Raft

Raft also requires some recovery measures

- a heartbeat is sent by the leader to prove it hasn't crashed
- if nodes don't hear from the leader, they start a new election round after a timeout
- nodes can only vote for a leader that has a longer history than them
as a result only values that are committed by a majority of nodes survive a leader crash

Distributed Ledger Technology

Blockchains

a blockchain is a distributed database with an **open set of nodes**

- anyone can join or leave the database at any point
 - like internet
- no hypothesis is made on the honesty of a group of nodes
 - messages intercepted or ignored
 - answering false information
 - deliberate and coordinated attacks

having a set of open nodes creates requirements over the architecture

- **data redundancy** : If a node that has the only copy of a piece of data leaves, the system loses its consistency
- **local verification of state** : because anyone can send any fake message, the state of the database needs to be double-checked in each node

blockchain represent data with a **delta representation**

- **initial state** : an initial state (genesis) is shared by all participants by an external mean (e.g. news paper)
- **list of past deltas** : each new participant receives from other nodes the list of the past deltas to apply to the initial state to obtain the latest state
- **new deltas** : periodically new deltas are sent to all participants for synchronization

Blockchains - Processing of updates 1/8

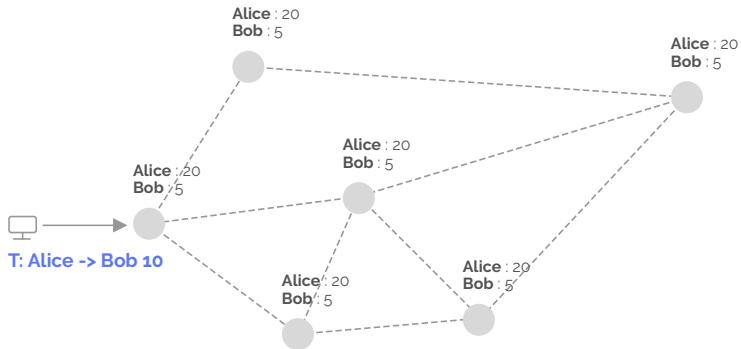


Figura: An application connects to a node and creates a transaction

Blockchains - Processing of updates 2/8

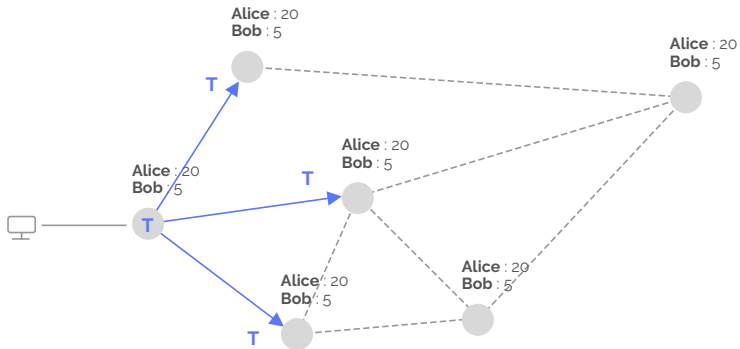


Figura: All peers are informed of the transaction and put in their pool

Blockchains - Processing of updates 3/8

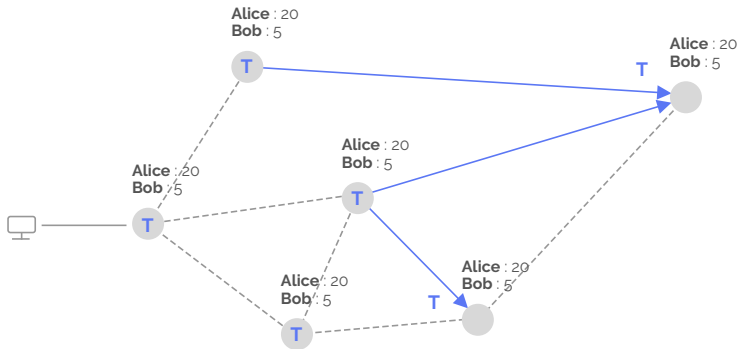


Figura: The nodes inform their own peers

Blockchains - Processing of updates 4/8

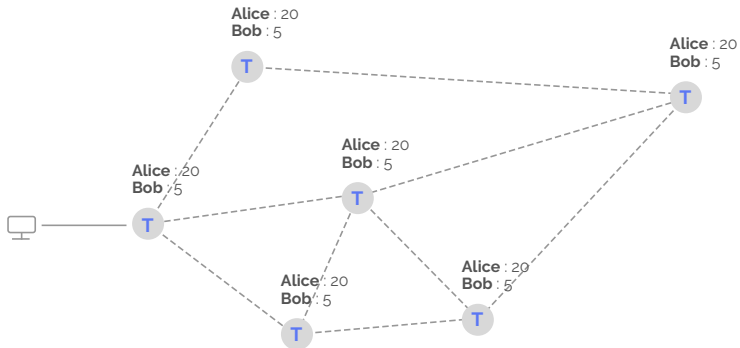


Figura: At some point all the nodes know about the transaction

Blockchains - Processing of updates 5/8

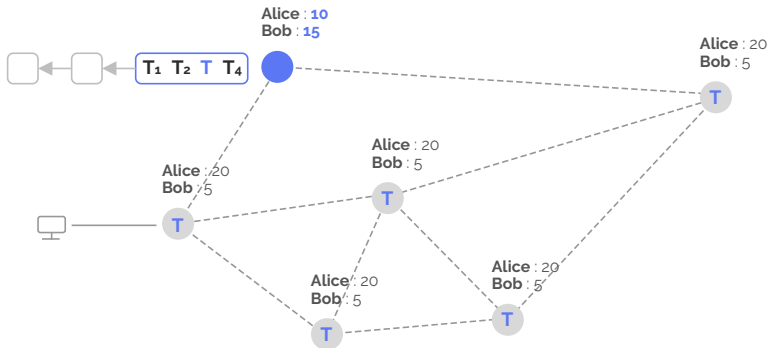


Figura: A node is randomly selected to execute all transactions in the pool

Blockchains - Processing of updates 6/8

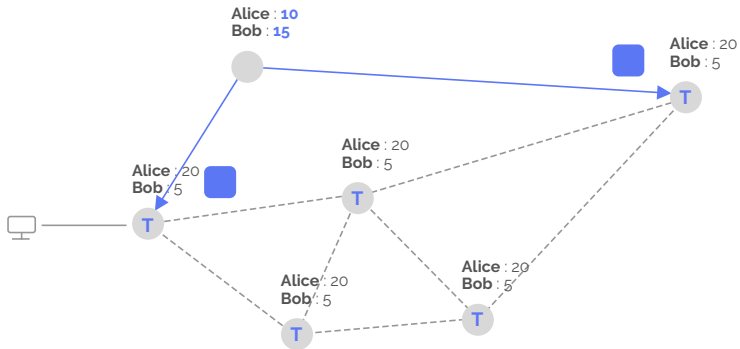


Figura: The new block of deltas is sent to the node peers

Blockchains - Processing of updates 7/8

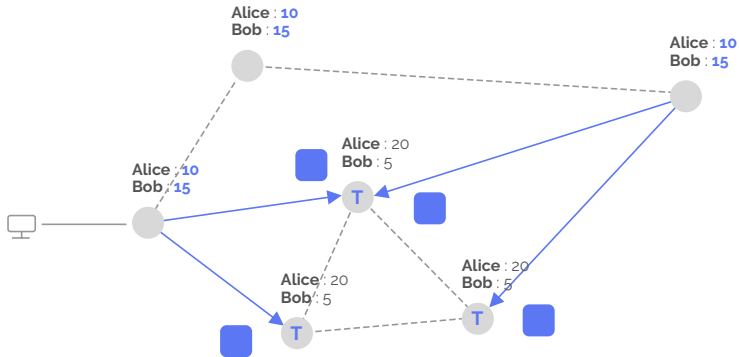


Figura: The new block of deltas is sent to from peer to peer

Blockchains - Processing of updates 8/8

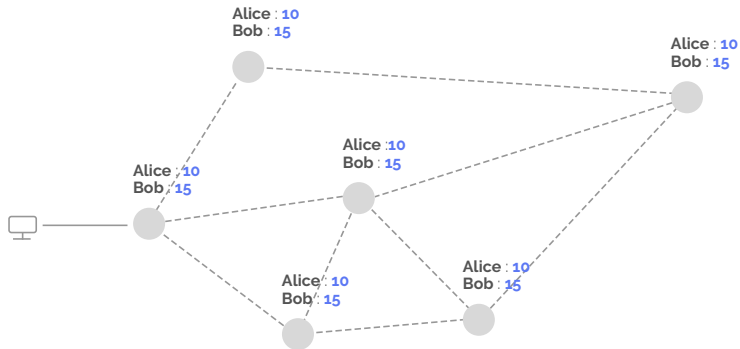


Figura: At some point all the nodes know about the block of deltas

Blockchains - Chain of blocks (of deltas)

the blockchain name was given because of the transactions being executed by blocks that point towards the previous block



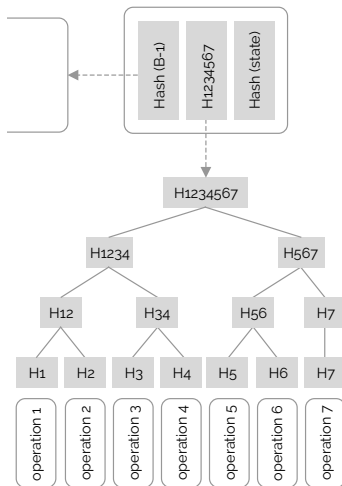
this was done to easily identify if there are concurrent blocks being broadcast in the network



to save bandwidth, blockchains don't exchange data (transactions, blocks, states) but **hashes** (of transactions, blocks, states).

if a node doesn't have the original object (e.g. transaction) it requests it to the node that send the hash. This **lazy data transfer** mechanism ensures an object is only sent once.

a block usually contains the root of the Merkle tree of the transactions executed, the hash of the previous block, and a hash of the final state after applying the transactions



Distributed Ledger Technology

Consensus in Blockchains

consensus in blockchains cannot be based on voting

a **Sybil attack** is the creation by an attacker of a large number of identities (nodes) that can vote in the consensus algorithm and thereafter bias it in the sense desired by the attacker

the only known solution to sybil attacks is **collateral-based voting** : base the voting system on something valuable and difficult to fake electronically

Satoshi Nakamoto is the first person to succeed in designing a collateral-based consensus algorithm with **Bitcoin** (2008)

the core idea of the Nakamoto consensus is

- make participants pay to vote (the amount of money of each participant is independent of the number of nodes they create)
- reimburse them for their participation if they follow the consensus algorithm (do not reimburse them if they deviate)

however, the voting in Nakamoto consensus is much more indirect than in database consensus

we will divide the Nakamoto consensus in 3 steps

- **collateral** : form of value used in the algorithm
- **sortition** : selection of a leader in charge of creating the next block
- **consensus** : selection of the current state in the history of blocks received

collateral is **electricity**

- nodes are required to burn electricity and compute something
- other nodes have to be able to check that the electricity was burnt by checking the result of the computation
 - the solution of an NP-complete problem requires an exponential time to find but can be checked in polynomial time

the problem chosen by Nakamoto to burn electricity is hash reversal

the leader is any node that can **reverse a given hash**

- there can be multiple leaders at the same time. Ties will be solved by the consensus part of the algorithms
- the hash to be reversed is the hash of the next block + predefined suffix $h^{-1}(block + suffix)$ and is sent with the block to other nodes
- other nodes can probabilistically check a node has burnt the electricity required to participate

the constants are adjusted for a block to be created every 10 minutes

Consensus - Nakamoto consensus / consensus

the current block is the block with **the longest chain**

- every time a block is added to a chain, its block producer spent electricity to reverse its hash
- the longest chain is the chain that contains the most used electricity / used computing power / used computing time

it is almost impossible to create a parallel chain of same length (it would take the same amount of computing power as the real chain, meaning it would cost almost the same amount of money and take almost the same amount of time)

Consensus - Nakamoto consensus / reimbursement

to reimburse participants for their expenses, Nakamoto embedded a currency (a list of accounts with 'money') in the database

- block producers add to their own account a predefined number of "virtual coins" every time they create a block
- operations on the database can be done only if the creator of the operation has "virtual coins" to pay for the operation fees
- block producers can then sell the "virtual coins" they created for themselves when they created a block, to people that want to use the database

Consensus - Nakamoto consensus / punishment

the fact that both the **proof of money expenditure** and **reimbursement** are in the block makes the algorithm completely local

actors that misbehave are punished by ignoring their blocks.

Nakamoto consensus is a voting system

Nakamoto consensus is a voting system

- **uniqueness** : a given amount of electricity can only "vote" / create a block once (possibly zero if it loses the hash reversing race)
- **proportionality** : statistically the number of "votes" / blocks created by a node is proportional to the total amount of electricity the node has spent
- **outcome** : the outcome of the vote is the current chain

nodes "vote" by spending money to add blocks to the chain of their choice

Nakamoto consensus is a voting system

if a vote was organized among block producers to choose which block they want as the current one

- each block producer would vote for the chain containing the largest amount of money to be reimbursed to them
- the chain with the largest amount of money to be reimbursed would get the most votes

in other words the longest chain would be chosen

it makes no economic sense to choose as current node any node other than the head of the longest chain

- the longest chain has the highest probability to survive
- any block added to a chain that is not the longest has a negative expectancy (cost money and is likely not to be ever reimbursed as the chain will just be ignored)

all **economically rational** actors chose the longest chain as the current state of the database

it is very difficult to rewrite the history of a blockchain with Nakamoto consensus. It requires a **51% attack**

in Nakamoto consensus, the space of hashes is explored in parallel by the miners. It is not a perfect partition, but rather a random sampling. Nevertheless, the **net computing power of the network** is significantly larger than the one of a miner

in order to rewrite the history of the database, an attacker would need more computing power than the net computing power of the network, to create blocks in < 10 minutes, in order to create more blocks and reveal a longer chain, making the network swap

Distributed Ledger Technology

Beyond Nakamoto consensus

- consumption of electricity
 - has reached unacceptable levels even for a very modest usage of the database
- cycle of exchanges
 - using electricity as a collateral means that block producers need to enter a perpetual cycle of exchanges money → electricity → virtual currency → money which makes the system fragile to exchange rates
- speed
 - simple solutions like decreasing the time between blocks by making the hash reversal problem don't work (because it generates an order of magnitude more simultaneous leaders)

due to the limitations of Nakamoto consensus, new algorithms known as Proof-of-Stake (PoS) tried to replace the electricity with the virtual coin as collateral

- **Peercoin** (King et Nadal 2012) : idea of using the virtual currency as **collateral** and first implementation
- **Chains of Activity** (Bentov et al. 2014) : basic algorithms for virtual currency collateral including **sortition** (*follow the satoshi*) and clear analysis of attacks to be prevented
- **Slasher** (Buterin, 2014) : **punish** nodes that deviate from the algorithm with destruction of their collateral

the combination of Peercoin, CoA and Slasher algorithms prevents all the main attacks and provides a basic PoS algorithm

- nothing at stake \Rightarrow Slasher punishment
- long range attack \Rightarrow CoA checkpoints
- stake grinding \Rightarrow CoA follow-the-satoshi

however, no clean document was published explaining the merge of these algorithms created a basic PoS algorithm at the exception of the white paper of Tezos (2014)

modern PoS algorithms implement the basic PoS algorithm with minor changes

- Tezos **Emmy** (Goodman 2014) basic PoS algorithm with follow-the-satoshi sortition, simple vote by committee
- **Ouroboros** (Kiayias et al. 2019) basic PoS algorithm with VRF sortition, simple vote by committee

once the basis of PoS were established new algorithms tried to collateralize database consensus algorithms

- **Tendermint** (Kwon 2014, Buchman 2016) : basic PoS algorithm with round-robin sortition, BFT-style vote by committee
- **Algorand** (Chen et Micali 2016) : basic PoS algorithm with VRF sortition and BFT-style vote by committee

modern Nakamoto-based and BFT-based PoS algorithms are only superficially different (committee simple voting vs BFT voting)

in some sense it shows that the solution to the Sybil attack problem (Nakamoto consensus, its PoS "simulation") is independent from the BFT consensus problem

but because Nakamoto consensus is an "all in one" algorithm, it reuses the same tool (hash reversal) to solve various orthogonal problems

in 2018, Emin Gun Sirer published the **Avalanche** algorithm

the idea is that consensus can be achieved by each node polling an increasing number of neighbours until the margin of error of the poll is low enough (and swapping accordingly)

the polling family of algorithms is **leaderless**

3 families of blockchain consensus algorithms have emerged

- patched versions of Nakamoto Consensus
 - Tezos (Emmy, Emmy+), Cardano (Ouroboros)
 - they remove the hash reversal and try to patch accordingly
- patched versions of database consensus algorithms
 - Cosmos (Tendermint), Algorand (BFT), Tezos (Tenderbake)
 - they add collateral to database consensus to avoid Sybil attacks
- polling algorithms
 - AVA (Avalanche)
 - each node does a poll on an increasing number of neighboring nodes until the margin error of the poll is small enough

Distributed Ledger Technology

Proof of Stake

Proof of Stake (PoS) algorithms are algorithms that replace the electricity collateral with virtual coins

Slasher in a lot of ways, although not all, makes proof of stake act like a sort of simulated proof of work

Vitalik Buterin, Oct 2014

PoS algorithms behave like Nakamoto consensus (PoW, Bitcoin) without the hash reversal, and patched accordingly

PoS algorithms require

- **Collateral** : a freeze / unfreeze mechanism to hold the coins hostage and destroy them in case of misbehavior
- **Sortition** : a mechanism to select the next block producer in a way that is random and proportional to the amount of coins owned by each participant
- **Consensus** : a mechanism to select the current head (the block on top of which all nodes will continue building the chain) and a way to punish the nodes that deviate from the consensus

Tezos (Goodman, 2014) introduced **Emmy**, the first PoS algorithm to put all the elements of Peercoin, Chain of Activity and Slasher together (freeze / unfreeze, sortition, accusations)

Tezos also added the concepts of delegation (inspired by BitShares), inflation, governance, a virtual machine designed to facilitate formal verification of smart-contracts, and the use of functional languages (OCaml) to write the core software and open the possibility of formally proving the code

- **Delegation**: if you don't want to create blocks, you can delegate your **right to create blocks** to someone else
- **Inflation**: in Bitcoin, creation of new coins (inflation) is just a way to jump-start the system, and stops after a while. Tezos sees inflation as a fundamental economic mechanism.
- **Governance**: the ability to vote for changes in the code to avoid technical separations of the community

Tezos Emmy (2014) works as follows

- **Collateral** : tokens are frozen for a given time
all claims of misbehavior need to happen in that window
- **Sortition** : a precomputed random calendar of block producers (and backups) is generated from a random snapshot of the amount of coins each node has
- **Consensus** for each level / round, a committee of 32 nodes is randomly selected (using the calendar system) to vote for the block they believe is the current one
all other blocks are invited to follow the committee majority, but are free to do what they want as long as they choose only one block

there are 3 attempts to cheat in Emmy

- bias the random calendar
 - **Failure to disclose seed**
- being part of a committee and vote for more than 1 block
 - **Double endorsement**
- not wanting to select only one head, and when it is your turn to create a block, create multiple different blocks
 - **Double baking**

in Nakamoto consensus, all these problems are solved by hash reversal which costs money (= collateral) hence economically rational actors don't try to cheat

any node that notices a violation can create a **accusation** (with proof) and trigger a punishment

- 1/2 of the coins hostage are destroyed,
- 1/2 of the coins hostage are awarded to the accuser

(if all the coins hostage are awarded to the accuser, it could team with the violator and denounce after a timeout to try to recover all the coins before someone else does)

liveness is achieved with a list of backups for the block producer

the rank of the backup in the list is called its **priority**

- if after 1 minute the backup hasn't seen a block from the node producer for level n it can create the block (and successively)

any block from the k -th backup that arrives before the k -th minute is invalid

Emmy doesn't use the block producer of node $n + 1$ as backup for node n because it wants to encourage node producers to be present in the network

- the calendar is known 1 week advance
- you want to discourage nodes to be present just for the minute they need to bake and disconnect after that

in Emmy, there are **block steals** (a block producer of priority > 0 may be the selected producer for a block and collect the reward)

the committee vote is implemented in a way that doesn't block the advancement of the algorithm

- votes are broadcasted as separate operations (each member announces he votes for bloc B on level n)
- the votes for level n are included in the block of level $n + 1$
- the recommendation function (score / fitness) is based on the number of votes of the previous block

the score function (fitness) is the recommendation done to the nodes about which chain they should consider as current (just like Nakamoto's longest chain)

it could depend on

- the number of endorsements of the previous block
- the priority of the node producer
- the level (round) of the block

in Emmy the recommended chain is the one that maximizes `length + nb_endorsements`

in summary Emmy is

- a **calendar** of leaders, backups and committee (from CoA)
- rules stating nodes have to chose one head / **create only one block** and committee members **vote only once** (from Slasher)
- a freeze / unfreeze and accusation based **punishment mechanism** to avoid deviations (from Slasher)
- endorsement **votes** for the current block by the committee
- a **recommendation** for all nodes to follow the committee

Tezos tends to see Emmy as a template algorithm (like Paxos) where some implementation details can be changed without fundamentally changing the algorithm

- sortition algorithm (follow-the-Satoshi, VRF)
- details of calendar (one "vertical" backup per round, next leader backup for previous leader)
- vote of the committee (simple vote, BFT)
- blocking or non-blocking committee vote
- recommendation function

Algorand

(Micali 2016) introduced the idea of using verifiable random functions (Micali et al 1999) to do sortition on-the-fly

- BFT is used inside of the committee instead of simple voting
- sortition is done with VRF
- punishment is not implemented yet

outside of the VRF and use of BFT vote for the committee, the algorithm is similar to Emmy

As discussed, at a very high level, a round of Algorand ideally proceeds as follows. First, a randomly selected user, the leader, proposes and circulates a new block. (This process includes initially selecting a few potential leaders and then ensuring that, at least a good fraction of the time, a single common leader emerges.) Second, a randomly selected committee of users is selected, and reaches Byzantine agreement on the block proposed by the leader. (This process includes that each step of the BA protocol is run by a separately selected committee.)

The agreed upon block is then digitally signed by a given threshold (T_H) of committee members. These digital signatures are circulated so that everyone is assured of which is the new block. (This includes circulating the credential of the signers, and authenticating just the hash of the new block, ensuring that everyone is guaranteed to learn the block, once its hash is made clear.)

Chen et Micali - Algorand (2017) section 4 page 22

Ouroboros are various PoS algorithms published between 2016 and 2019

- Ouroboros Classic (2016) : basics
- Ouroboros Praos (2017) : VRF and empty slots
- Ouroboros Genesis (2018) : checkpoint validation
- Ouroboros BFT (2018) : BFT voting
- Ouroboros Chronos (2019) : removal of clock

on most points Ouroboros is similar to Emmy. So far, only Ouroboros classic has been implemented.

Emmy+ solves some limitations of Emmy

- minor flaw discovered during the review of the algorithm done by the INRIA before the mainnet launch in 2018
- minor annoyances for block producers
- simplification of the analysis of the algorithm

Emmy+ also introduces some new fundamental ideas

in Emmy, a block producer for level n needs to decide between

- wait till more endorsements arrive for level $n - 1$ to increase the score of the block it has produced
- broadcast the block to maximise the probability his block will be seen by the committee at level n before a block (potentially) made by his backup if he waits too much

Tezos - Emmy+ (2019) convergence reasoning

in Emmy, the fitness of a block (recommendation to nodes) is $\text{length} + \# \text{endorsements}$

this makes it difficult to reason on it

for instance proving it makes no economic sense to deviate from the recommendation (just like in Nakamoto consensus)

Tezos - Emmy+ (2019) propagation speed

Emmy+ introduces the idea that nodes shouldn't select the current chain based on a complex fitness function but based on **time** (the first block that arrives)

the committee vote is not used to establish a numeric recommendation, but to **delay the blocks** in such a way the most approved one is propagated quicker in the network

the main idea behind Emmy+ is to emulate the behavior of Nakamoto Consensus where **the minority chain advances slower** than the majority chain

this temporal behavior protects Nakamoto consensus from attacks (as attackers need to have enough computing power to produce blocks quicker than 10 minutes)

- in Emmy+, attackers need to have enough votes in the committee to delay the other chains

Emmy+ was implemented within the existing Emmy framework for simplicity

- the fitness function is the level of the block
- a block is valid after $(32 - e) * T_0$ seconds of its production time (which still follows the k -priority * 1 minute rule)

there is no obvious way to add a punishment for broadcasting a block before its validity time (it would require a heartbeat giving a random unique information at precise times). Therefore punishment for these violations hasn't been added.

TenderBake is a variation of Tendermint to make it closer to Emmy (it could be considered the Tendermint → Emmy approach, as opposed as Emmy → BFT)

TenderBake won't be deployed in Tezos as is it considered inferior to Emmy+

instead research is orienting towards a **finality gadget** like in Ethereum's Casper on top of Emmy+

Distributed Ledger Technology

Exploration of the space of consensus algorithms

the space of Emmy and Emmy+ like algorithms is huge, there are many variants that need to be implemented and simulated in order to chose the best combination of features

our problem is to quickly prototype, simulate and compare (in realistic conditions) many of such variants : **Projects COPES and CRISP**

- today, Emmy's calendar computes a leader, a backup list and a committee per level
 - use leader for level $n + 1$ as backup for level n
 - don't select a leader, let each participant in the committee creates a block and selection of final leader with round-robin
- today Emmy's committee makes a non-blocking simple vote leading to a probabilistic finalization
 - blocking vote
 - more complex vote (pBFT, Tendermint, Paxos, Raft)
 - deterministic finalization
- today Emmy+ doesn't penalize for violating the delays rule
 - introduction of a heartbeat + Slasher style penalty