POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

The Gasper Protocol: a Proof of Stake Era for Ethereum

Relatori
prof. Danilo Bazzanella
dott. Andrea Gangemi

Candidato
Giulia Pititto
279737

Anno Accademico 2021-2022

Summary

Nowadays, it is crucial to ensure the secrecy of shared data and the validity of received information. This is the case of money exchange between two different parties, in which one must keep the data of its own bank account hidden, or the exchange of information in the military field, where an interception must be prevented. For such reasons, cryptography was born. Because of its primary characteristic of assuring secrecy and reliability, one of its main applications regards blockchains: a network composed of people who do not know or trust each other uses it to ensure valid transactions.
In 2009, Bitcoin was developed: a shared, permissionless blockchain whose structure and integrity are based on a Proof of Work (PoW) protocol. Miners, users that perform the protocol, compute a value that must be lower than a specific target to prove that they have done a sufficient amount of work. This type of protocol leads to a certain centralization, due to the ever-increasing difficulty of the task, so that only few users, combining their resources in a mining pool, have the appropriate computational power to carry it out; moreover, since it is a trial-and-error kind of task, it consumes a lot of energy, so it has a great negative environmental impact.
In 2013, the idea of another type of blockchain, Ethereum, was conceived. Its main purpose was not only to assure safe transactions between users, but also the possibility of implementing Smart Contracts on a distributed platform. When Ethereum went live for the first time in 2015, it used a PoW consensus protocol; only recently, on September 15th, 2022, Ethereum replaced it with a Proof of Stake (PoS) one. This change was made to improve decentralization and scalability and to limit the environmental impact of the blockchain. PoS is based on the idea that all those who take part in the protocol, known as validators, have a stake in the network, a frozen amount of the cryptocurrency managed by the blockchain, and their decisional power is proportional to that stake. The PoS protocol used by Ethereum is Gasper, obtained combining Casper and GHOST. Casper is a finalization gadget used to mark certain blocks as finalized, so that even a user with only partial information can be sure of the validity of that specific block.
GHOST is based on the idea that ommer blocks, valid blocks that are in the chain but do not belong to the main one, contribute to the heaviness of a chain from a computational point of view. The chosen branch in a fork will not be the longest one, as in Bitcoin, but the heaviest one.
Gasper combines these two elements in a complete PoS protocol: a user becomes a validator when he stakes a fixed amount of its Ether, Ethereum's cryptocurrency. Validators are then split into committees and in each slot, the unit in which time is divided into, one is randomly chosen to propose a block, while the others must attest to the block obtained using GHOST. If the block receives at least two thirds of attestations, messages with which validators vote for the next head of the chain, it is finalized and added to the chain. Gasper possesses the properties of safety and liveness: two conflicting blocks, thus when neither is an ancestor of the other, can never be finalized and the set of finalized

blocks will always grow.
The purpose of this master thesis is the in-depth description of the Gasper protocol and its properties and the demonstration of how and why PoS seems like a better alternative to PoW.

Contents

List of Figures

Chapter 1

Blockchain basics

Blockchains are gaining importance and are being used in multiple sectors, such as finance, art, healthcare, identity and so on. Blockchains base their functioning on a consensus protocol, a series of rules users commit to follow to grant the right performance of the network. The most used is the Proof of Work protocol, but it fails to achieve the blockchain trilemma, that is at the same time it must possess the properties of decentralization, security and scalability; indeed, a blockchain must not be controlled by a central authority, it must ensure users' privacy and, because of its ever-increasing fame, it must be able to process a lot of information. An alternative to the Proof of Work protocol is the Proof of Stake one, which seems to be better at resolving the trilemma. The aim of this thesis is the study of one of the most important blockchains, Ethereum, specifically of its Proof of Stake consensus protocol Gasper. In this first chapter, the concept of blockchain is presented along with its main cryptographic tools. Then a generic description of the differences between Proof of Work and Proof of Stake is stated. Eventually, a description of the two most important blockchains, Bitcoin and Ethereum, is served.

Blockchain is a **Distributed Ledger Technology**: the database or its copies are shared among different physical locations, *nodes*, like in a classic distributed database; this secures the network from informatic attacks, because it is more difficult and extremely expensive to attack a system with multiple breach points. The blockchain is referred to as a particular type of distributed ledger technology because of three fundamental aspects:

1. the control of the database is *decentralized*. There is not a central authority that grants the reliability and integrity of data and transactions, but those are achieved by applying a specific consensus protocol. Because of its decentralized nature, all the decisions are made by the entirety of users, so a single user or a group of them cannot make changes in the network.

2. This consensus protocol renders the blockchain *reliable in trustless environments* even if the users do not completely trust one another.

3. To grant both decentralization and reliability in all environments, the ledger uses *cryptographic techniques*.

In a blockchain data are saved in different blocks, each one connected to the other by cryptographic tools. This dependence of each block from the previous ones makes it impossible to retroactively alter the database, rendering it a *"perpetual chain of immutable records"*[2].

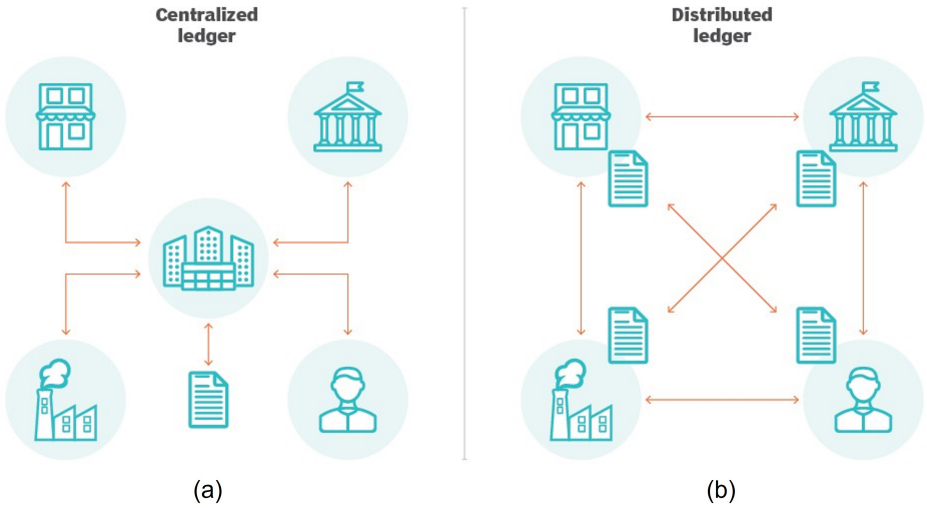


Figure 1.1. Difference between centralized and distributed ledger: (*a*) there is only one copy of the database owned by the central authority that rules over every user; (*b*) each user has its own copy of the database and there is no central authority [21].

Blockchains have found wide use in many fields, from art to finance, and their functioning is based on cryptocurrencies. The term cryptocurrency refers to a digital representation of value based on cryptography: unlike traditional money, it does not exist in physical form and is not managed by a central authority. Notions on cryptocurrency transactions are registered in a decentralized digital ledger, typically based on blockchain technology. To manage the cryptocurrency, a *wallet* is needed. Various types of wallets can be distinguished as follows:

- *Hot wallets*: the private keys are stored online.
  - *Desktop wallet* is a software installed on the user's computer necessary to manage the bitcoin wallet.
  - *Mobile wallet* is the most common one and consists in the installation of an application on a smartphone. This type of wallet is lighter and simpler, but often it is not a 'complete' wallet because there are fewer possible actions with this kind of wallet than with a desktop one.
  - *Web wallet* is accessible by the browser, without the need to download any software.

- *Cold wallets*: the private keys are not stored online, but in offline places.
  - *Hardware wallet* is a mobile device that can be connected to a computer to manage a wallet.
  - *Paper wallet* consists in the storage of the login credentials on paper.

Obviously, the security of the wallet depends not only on the type of wallet used, but also on the user itself. The keys can be obtained from the seed, but it is extremely important not to lose this seed. If a user loses it, he will no longer be able to access the wallet.
Wallets can be either *non deterministic* or *deterministic*. All keys of a non deterministic wallet are generated independently from each other, hence there is no link between them. This means that if the owner of the wallet forgets one of these keys, he looses access to the wallet. Thus, a backup is continuously needed, generating storage problems.
Deterministic wallets use keys generated from a common *master key*, which is derived from a *seed*. With this type of wallets, the problem of forgetting a key does not present itself because every key can be derived from the seed. It is extremely important not to loose this seed, so the user is asked to remember a series of English words from which it can be easily derived. A *Hierarchical Deterministic (HD)* wallet is the most used process to derive keys. It is based on a tree structure where each key can be obtained from the previous one through cryptography. Through a fixed phrase, the list of words to remember is generated; from these the seed is derived and then the master key is obtained from it. The master key is at the base of the generation of the various parent keys, from which descend the child keys, and so on. In this way, it is possible to generate a sequence of public keys even without knowing the respective private keys. This increases the security of the wallet, because the private key can be stored in a cold wallet without ever being entered online.

All the keys of a HD wallet are identified through a *path*, a sequence of symbols separated by /; each one of these symbols classifies a distinct level of the tree. BIP-44 is the standard utilized to define a path; it generates a path with five levels of depth of the tree:

$$m \text{ or } M/44'/CoinType'/Account'/Change/AddressIndex$$

- m if the key is derived from the private master key, M if it is derived from the public master key;
- 44 because the standard utilized is BIP-44;
- *Coin Type* represents the specific cryptocurrency involved;
- *Account* identifies the different accounts run by a user: one for donations, one for investments, and so on;
- *Change* is 0 if the address is needed to receive a transaction; it is 1 if it is needed to send a transaction and receive the corresponding change;
- *Address Index* represents the specific address of the account.

Finally, we can distinguish two kinds of users on a blockchain: *light nodes* and *full nodes*. Full nodes store the entirety of the blockchain and manage every aspect of the protocol, such as the validation of new transactions. Light nodes do not save the whole blockchain, but they connect to a full node to retrieve information about transactions. They can validate new transactions, but first they have to ask a full node to give them the information needed for the verification.

1.1 Elliptic Curve Cryptography

The integrity of the chain and the validity of transactions are assured by cryptography.
In 1976, asymmetric key cryptography, also known as public key cryptography, was invented. In this case the key generator produces a pair of keys, k_p and k_s, for each user: k_p is the public key, which is known by everyone and is used by the owner to encrypt a message; k_s is the secret key, known only by its assigned user and needed to decrypt the message.
First suggested independently by Miller [15] in 1986 and Koblitz [11] in 1987, the use of elliptic curves revolutionized public key cryptography. To better understand how Elliptic Curves Cryptography (ECC) works, first a brief introduction on elliptic curves has to be done.

1.1.1 Elliptic curves

Definition 1.1.1 *An elliptic curve E_K defined over a field K of characteristic $\neq 2, 3$ is the set of solutions $(x, y) \in K^2$ of the equation*

$$y^2 = x^3 + ax + b, \quad a, b \in K \tag{1.1}$$

together with the point at infinity, identified with O [11].

The curve must be non-singular, so it is required that

$$4a^3 + 27b^2 \neq 0.$$

If the field K has characteristic $= 2$ *or* 3, the equation of the curve changes. For example, if $\text{char}(K) = 2$, the equation of the curve is

$$y^2 + cxy + dy = x^3 + ax + b, \quad a, b, c, d \in K. \tag{1.2}$$

This is a generalization because it is possible to obtain the specific case of equation (1.1) by setting $c = d = 0$ in equation (1.2).
The set of points belonging to the curve E_K forms a group with respect to the sum, with identity element O: it is possible to carry out operation on elliptic curves such as computing the opposite of a point, adding two different points or calculating multiples of a given point.
Let us consider the elliptic curve E_K with equation (1.1) and two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ belonging to it.

The opposite of point P is the second point belonging to E_K having the same abscissa as P.

Taking both P and Q different from the point at infinity and neither of them the opposite of the other, we can easily compute the sum $R = P + Q$ such as

$$\begin{cases} x_R = \lambda^2 - x_P - x_Q \\ \\ y_R = \lambda(x_P - x_R) - y_P \end{cases} \tag{1.3}$$

where

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}. \tag{1.4}$$

This could also be done geometrically if K represents the real numbers: first we draw the secant line passing through P and Q that intersects the curve in a third point, then we take R as the opposite of the intersection point.

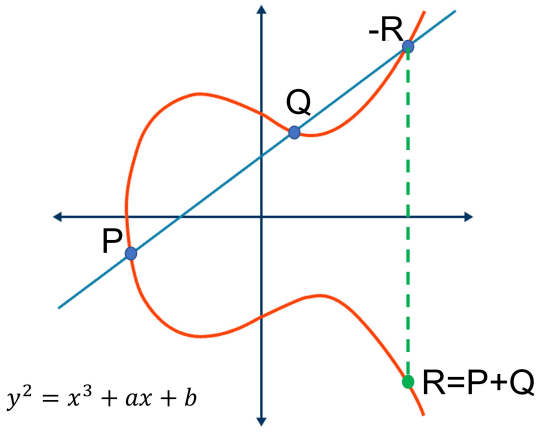


Figure 1.2. Sum of two different points on an elliptic curve.

Let us observe that if P or Q are the point at infinity, for example $Q = O$, then $P + Q = P + O = P$ and that if $Q = -P$, then $P + Q = P + (-P) = O$.

If $P = Q$, then calculating $R = P + Q$ equals computing $R = 2P$, so a multiple of P. In this case, the formulas become:

$$\begin{cases} x_R = \lambda^2 - 2x_P \\ \\ y_R = \lambda(x_P - x_R) - y_P \end{cases} \tag{1.5}$$

where

$$\lambda = \frac{3x_P^2 + a}{2y_P}. \tag{1.6}$$

Geometrically, doubling a point means considering the tangent line to the curve in P, which is unique because of the hypothesis of regular curve, and then taking the opposite of the point of intersection between the tangent and the curve.

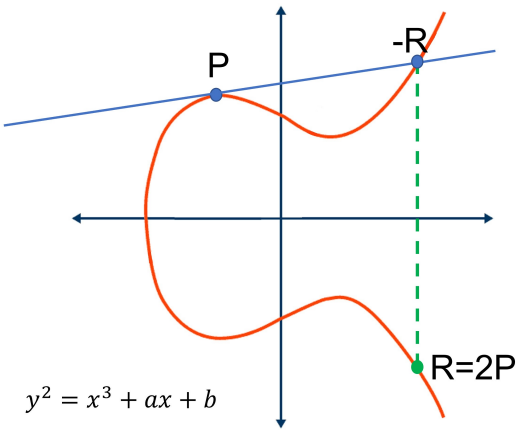


Figure 1.3. Doubling of a point on an elliptic curve.

Using the addition formulas, we can calculate a multiple kP of the point P in the same computing time that it takes to compute the exponential a^k, that is with $O(\log k)$ doublings and additions, with the so-called *Fast Exponentiation Algorithm*. For example, if $k = 9$ and we want to compute $R = kP$, we can consider it as $R = 9P = 2^0P + 2^3P = P + 8P = P + 2(2(2P))$:

1. first we compute $2P = P + P$ with (1.5) and (1.6);
2. then we obtain $4P = 2P + 2P$;
3. then we calculate $8P = 4P + 4P$;
4. finally we get $9P = P + 8P$.

So we used an addiction and three doublings instead of eight addictions.

It must be noted that if E_K is a curve of equation (1.2), then the formulas (1.2, 1.4, 1.5, 1.6) become:

$$\begin{cases} x_R = \lambda^2 + c\lambda - x_P - x_Q \\ y_R = \lambda(x_P - x_R) - y_P - cx_R - d\,, \qquad \text{if } P \neq Q \\ \lambda = \frac{y_Q - y_P}{x_Q - x_P} \end{cases} \tag{1.7}$$

$$\begin{cases} x_R = \lambda^2 + c\lambda - 2x_P \\ \\ y_R = \lambda(x_P - x_R) - y_P - cx_R - d\,, \qquad \text{if } P = Q \\ \\ \lambda = \frac{3x_P^2 + a - cy_P}{2y_P + cx_P + d} \end{cases} \tag{1.8}$$

We are interested in elliptic curves defined on finite fields $\mathbb{F}_p$.

1.1.2 Text encoding on elliptic curves

The use of ECC is complicated by the problem of text encoding. In classic public key systems, a text is divided in blocks that are processed with coding functions: the message becomes a sequence of integer numbers that can be decrypted to obtain the plaintext. Elliptic curve cryptography raises the problem of how to codify integers into points of a curve. To avoid this problem, probabilistic coding systems with an extremely low probability of failure are used.
Let us report an example of probabilistic encoding of a text.

Given an elliptic curve of equation (1.1) defined on $\mathbb{F}_p$ with $p \equiv 3 \mod 4$ and given a message m, we need to find the corresponding point P_m on the curve and vice versa. Since $p \equiv 3 \mod 4$, calculating quadratic residues in $\mathbb{F}_p$ means to compute the power $(p+1)/4$. We cannot simply take $P_m = (m, (m^3 + am + b)^{(p+1)/4})$, because we are not sure that the $y-$coordinate is a quadratic residue. To solve this problem:

1. first we define k appending three digits to m until we find one for which $k^3 + ak + b$ is a quadratic residue. By doing so, we are increasing our possibility of finding an appropriate point, because from a single integer m we obtain 1000 values k ($m|000$, $m|001$, ..., $m|999$).
2. Now we can fix $P_m = (k, (k^3 + ak + b)^{(p+1)/4})$.
3. To carry out the inverse procedure, that is to decode the point P_m in the integer m, we simply have to remove the three least significant digits from the $x-$coordinate of point P_m.

The algorithm fails only if none of the values k is a quadratic residue.

As previously said, the set of points belonging to a curve forms a group; therefore, after coding a message m into a point of the curve P_m, it is possible to use all the encryption systems operating on groups to encode P_m on another point of the curve and then to decrypt it.

1.1.3 Elliptic Curve Discrete Logarithm Problem

The safety of elliptic curves schemes is assured by the Elliptic Curve Discrete Logarithm Problem (ECDLP): given an elliptic curve E, defined over a field K, and two points

$P, Q \in E$, find x such that $Q = xP$. It is the analogue of the Discrete Logarithm Problem (DLP) on finite fields. ECDLP is much more difficult than DLP, in fact the strongest and fastest techniques used to solve the latter seem to not be applicable to solve the elliptic version: it is possible to choose smaller fields for elliptic curve cryptography than those needed for cryptosystems based on the discrete logarithm problem. This translates into high security guaranteed even using much smaller keys.

1.2 Hash functions

One of the main characteristics of a blockchain, its immutability, is guaranteed by an extremely important cryptographic tool: hash functions.
Any function that maps inputs of arbitrary size to fixed-size outputs can be considered a hash function. In mathematical terms:

Definition 1.2.1 *Indicating with Σ the alphabet and with Σ^* the set of all possible words of arbitrary length obtainable with the alphabet, we define* ***hash function*** *the transformation*

$$h : \Sigma^* \to \Sigma^n,$$

with n being a fixed integer corresponding to the length of the output.

Hash functions typically find their main application into the informatic field, hence the most common used alphabet is the binary one $\Sigma = \{0,1\}$ with $n = 160, 256, 384$ or 512. If x is the input, the output $h(x)$ is said *hash* or *digest* of x.

We will now analyze some properties of hash functions.

Property 1.2.1 *The* ***avalanche effect*** *entails that even the smallest change in the input of the hash function generates a completely different digest.*

We will consider as cryptographic hash functions only those for which this effect is valid.

Property 1.2.2 *A hash function h is a* ***one-way function****: given a digest z, finding x such that $h(x) = z$ must be a computationally infeasible problem.*

The previous property states that it must be easy to hash an input, but doing the inverse must be extremely difficult.

Property 1.2.3 *A cryptographic hash function must operate at a* ***reasonable speed*** *because, in many algorithms, it is necessary to compute the digests quickly.*

Definition 1.2.2 *A pair of elements $(a, b) \in \Sigma^*$, $a \neq b$, such that $h(a) = h(b)$ is a* ***collision****.*

Property 1.2.4 *A hash function h is* ***weakly collision-free*** *if fixed $a \in \Sigma^*$, finding $b \neq a$ such that $h(a) = h(b)$ is a computationally infeasible problem.*

Property 1.2.5 *A hash function h is* ***strongly collision-free*** *if finding any collision (a, b) is a computationally infeasible problem.*

Due to the finite size of the set of possible digests compared to the dimension of the set of all possible inputs, it is obvious that a collision must exist. From this derives the necessity of the existence of these last two properties, which shield the function from brute-force attacks: a person should not be able to try all the possible inputs and find the correct digest in a computationally acceptable time, lest he breaks the hash function.

Let us consider the SHA (Secure Hash Algorithm) family, composed of six different hash functions, developed since 1993 by the National Security Agency and published by the National Institute of Standard Technology: SHA−0, SHA−1, SHA−224, SHA−256, SHA−384 and SHA−512. SHA−0 is the first version of SHA−1, changed briefly after its publication because of a flaw in its algorithm; function SHA−1 produces a digest of 160 bits and it has been broken; the other four functions, referred collectively as SHA−2, produce digests of length equal to the number in their names. These functions have not been broken yet. In 2012 the hash function SHA−3 has been added to the family; it is a version of the Keccak function where some parameters are changed.

Hash functions have several fundamental uses in blockchains, such as granting the identity of a user or even creating the structure of the blockchain itself.

1.3 Digital Signatures

In cryptography, it is of fundamental importance to be able to prove that you are the true sender of a message; for example, if Alice sends a message to Bob, the latter must be able to verify that Alice is truly the one that sent the message. This is the aim of a digital signature: it guarantees the identity of the person who performs an action. The digital signature was created to prevent an attacker pretending to be another person sending a message, because in public key cryptography everyone knows a person's public key, so they can use it to impersonate someone else. In blockchains this act of prevention becomes crucial.

1.3.1 Elliptic Curve Digital Signature Algorithm

One of the most diffused digital signature system is the Digital Signature Algorithm (DSA), because it is more efficient and it uses shorter signatures. DSA also has an analogue on elliptic curves: Elliptic Curves Digital Signature Algorithm (ECDSA). The algorithm is composed of the following steps.

- An elliptic curve defined on a finite field $\mathbb{F}_p$, a point G of the curve, the prime number n of points of the curve and a hash function h must be given.
- Each user chooses its private key d and generates its public key as the point $P = dG$ on the curve.
- The signature protocol is the following:
  - the sender computes the hash of the message M: $h = h(M)$

  - he randomly chooses a value $k \in \mathbb{Z}_n$
  - then he calculates the point of the curve $kG = (x, y)$
  - he fixes $r \equiv x \mod n$ and finally computes $s = (h + rd)/k \mod n$.
  - The signature is the pair (r, s).

- To verify the signature the recipient:
  - calculates the inverse of s: $w = s^{-1} = k/(h + rd)$
  - then $u = wh$ and $v = wr$
  - finally he computes the point $Q = uG + vP$.
  - The recipient accepts the signature if and only if $r \equiv x_Q \mod n$.

- The protocol works because

$$Q = uG + vP = whG + wr(dG) = \Big(\frac{kh}{h + rd} + \frac{krd}{h + rd}\Big)G = kG = (x, y).$$

It is important to note that only the real addresser knows his private key d, so only he can determine the signature (r, s). Furthermore, the signature is also determined using the message digest $h = h(M)$, so it cannot be used by an attacker to sign other fraudulent messages.

1.3.2 Boneh-Lynn-Schacham Signature Scheme

Another important digital signature algorithm is the *Boneh-Lynn-Schacham (BLS)* signature scheme; it was conceived in 2001 and is based on a pairing between elliptic curves. It is also known as *short signature* due to its dimension of only 32 bytes.

In the 90s, some researchers discovered the *Pairing*, an operation that could map the Elliptic Curve Discrete Logarithm Problem into the Discrete Logarithm Problem on a finite field, in polynomial time. This function, at the start of the millennium, was used to build safe cryptographic protocols.

Definition 1.3.1 (Pairing) *Given two abelian groups $(G_1, +)$ and $(G_2, \cdot)$, a pairing is a bilinear, non-degenerate and computationally efficient map*

$$e : G_1 \times G_1 \to G_2.$$

Bilinear means that

$$\forall\, P, Q \in G_1,\ \forall\, a, b \in \mathbb{Z},\quad e(aP, bQ) = e(P, Q)^{ab}.$$

Non-degenerate means that

$$\textit{if}\ \ e(P, P)^m = 1,\ \textit{then}\ \ mP = 0.$$

Some properties of the pairing are useful to better understand the functioning of the BLS signature scheme.

Property 1.3.1 *A pairing is symmetric.*

Indeed, if G is a generator of G_1 and P, Q are public keys with respective private keys p, q, then:

$$e(P,Q) = e(pG, qG) = e(G,G)^{pq} = e(G,G)^{qp} = e(qG, pG) = e(Q,P).$$

Property 1.3.2 *With pairings, quadratic constraints can be checked.*

On elliptic curves, only linear constraints could be verified; pairings add the possibility of also verifying quadratic ones. For example, checking that $e(P,Q)e(G,5G) = 1$ is equivalent to check that $pq + 5 = 0$. In fact,

$$e(P,Q)e(G,5G) = e(G,G)^{pq}e(G,G)^5 = e(G,G)^{pq+5} \text{ and } 1 = e(G,G)^0.$$

Property 1.3.3 *Due to its bilinearity, a pairing is such that*

$$e(P+Q,R) = e(P,R)e(Q,R) \quad \textit{and} \quad e(P,Q+R) = e(P,Q)e(P,R).$$

The existence of this type of function is greatly problematic due to the following theorem.

Theorem 1.3.1 *If $e : G_1 \times G_1 \to G_2$ is a pairing, then an algorithm that efficiently solves the Discrete Logarithm Problem on G_2 does so even on G_1.*

This means that if a pairing between an elliptic curve G_1 and a finite field G_2 can be found, then solving the DLP on the latter results in solving it on the former. This feature could break many cryptographic algorithm based on the ECDLP.

The BLS signature scheme is built on a Gap Group. Before defining what a Gap Group is, the following definitions are needed.

Definition 1.3.2 *The Decisional Diffie-Hellman (DDH) problem consists in verifying that cG=abG, knowing G, aG, bG and cG.*

Definition 1.3.3 *The Computational Diffie-Hellman (CDH) problem entails computing abG given G, aG and bG.*

Theorem 1.3.2 *If a pairing exists between $(G_1, +)$ and $(G_2, \cdot)$, then the DDH problem is trivial on G_1.*

It is possible now to define what a Gap Group is:

Definition 1.3.4 (Gap Group) *A Gap Group is a cyclic group where the CDH problem is unfeasible, while the DDH problem is trivial.*

The Boneh-Lynn-Schacham signature algorithm is the following:

- A pairing e between two groups G_1 and G_2 must be given. Both groups must have order p, with p prime. G_1 must be an elliptic curve defined on the finite field $\mathbb{F}_p$, with generator G. G_2 must be a finite field.
- Each user chooses $d \in \mathbb{F}_p$ as its private key and computes its public key as the point $P = dG$ of the curve.
- The signature protocol is the following:
  - the sender computes the hash of the message M: $H = h(M)$. Then, this hash is coded into a point of the curve, that is $H = \alpha G$ for some $\alpha \in \mathbb{F}_p$
  - the sender computes the point $S = d_A H$ of the curve, which will be the signature.
- To verify the signature the recipient simply checks if $e(P_A, H) = e(G, S)$.
- The protocol works because

$$e(P_A, H) = e(d_A G, \alpha G) = e(G, G)^{d_A \alpha} = e(G, d_A \alpha G) = e(G, d_A H) = e(G, S).$$

The algorithm uses a pairing for the verification of the signature, that is a point of an elliptic curve. The protocol is completely deterministic, nothing is randomly chosen and every parameter is chosen by the signer.

A multi-signature scheme is a cryptographic protocol that permits a certain number of users to share control over a single account, each with the same importance. The signature is a combination of the signatures of all the participants rather than a collection of them; this is done to increases safety. If the total number of participants is n, then a scheme where everyone is needed to sign a message is said to be a *multisig n-of-n scheme*; if not everyone is needed, but only a fixed number $m < n$ of participants is, the scheme is called a *multisig m-of-n scheme*. The latters are not always efficient due to the need to keep trace of all possible combinations of the multiple public keys; thus, they are built only on certain digital signature protocols. One of the main merits of multi-signature schemes is that the signature algorithm changes, but the verification process is the same; therefore, a multisig signature is indistinguishable from a classical signature.

The BLS signature allows to generate multi-signature $n-$of$-n$ schemes easily.

- Compute the sum of all the n digital signatures and public keys, multiplied for a number $a_i = h(P_i||P_1||\dots||P_n)$, depending on all the public keys: $S_{sum} = \sum_{i=1}^{n} a_i S_i$, $P_{sum} = \sum_{i=1}^{n} a_i P_i$.
- The verification of the signature is easy; it implies checking that $e(G, S_{sum}) = e(P_{sum}, H)$.

- The protocol works because:

$$e(G, S_{sum}) = e(G, a_1S_1 + \cdots + a_nS_n) = e(G, a_1S_1) \dots e(G, a_nS_n) =$$

$$= e(G, a_1d_1H) \dots e(G, a_nd_nH) = e(a_1d_1G, H) \dots e(a_nd_nG, H) =$$

$$= e(a_1P_1 + \cdots + a_nP_n, H) = e(P_{sum}, H)$$

The BLS signature is one of the few that allows to build a multisig $m-$of$-n$ scheme, $m < n$. These schemes are based on other cryptographic systems such as the Shamir Secret Sharing algorithm [18].

To use a digital signature based on a pairing, the signature algorithm must be based on a *pairing friendly curve*, a property deriving from the *embedding degree*.

Definition 1.3.5 *The embedding degree of a curve is the smallest $k \in \mathbb{Z}$ such that, given the number N of points of the curve and the cardinality q of the finite field on which the curve is defined, it is true that N divides $q^k - 1$.*

It can be shown that, if $k \leq 6$ then the DLP can be solved efficiently on the field $G_2 = \mathbb{F}_{q^k}$, with $G_1 = E(\mathbb{F}_q)$ and that if k is too big, the pairing is computationally inefficient.

1.4 Block structure

The fundamental unit of a blockchain is the block. A blockchain is a series of blocks in which transactions, metadata and other useful information are stored. Each block is connected to the previous one, up until the first block called the *genesis block*. Blocks have a particular structure consisting of two parts: the header and the body.

1. In the header we can find general management fields like

   - the hash of the block, computed with the data stored in the block itself;
   - the hash of the previous block header, used to connect the current block to the prior one;
   - a timestamp to know exactly when the block was created;
   - the version of the currently used protocol;
   - the Merkle root, a value used to verify the validity of a transaction.

   We can also find certain fields depending on the specific blockchain, such as

   - a difficulty parameter measuring the complexity of the creation of the block in relation to the genesis one; this parameter is often changed to guarantee a constant interval between the creation of two consecutive blocks.
   - The nonce and the target that are two particular values used in specific protocols to create a new block.

2. In the body we find the list of transactions stored in the block: their number may vary according to their size and to the settings of the specific blockchain.
   It has to be noted that blocks have a fixed maximum dimension.

Some of the information that can be found inside the block are of utmost importance for the management of the network.
As mentioned above, each block carries the hash of the previous block header; this creates an actual chain which cannot be modified without everyone being aware of it. This is due to the avalanche property of hash functions: if an attacker wants to change a valid block n, with a fraudulent one n', the hash of the header of block n' will be different from the one of block n, so there will not be correspondence between the field -hash of the previous block header- contained in block $n+1$ and the hash of the header of block n'.

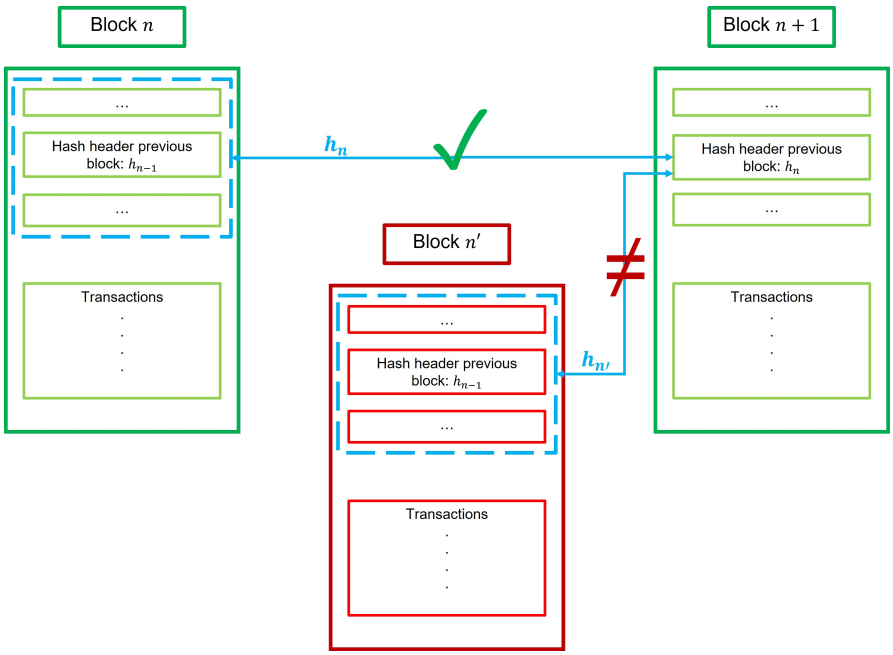


Figure 1.4. Fraudulent attempted block replacement.

Great importance is held by the Merkle root field: this value is calculated by combining the hashes of the transactions contained in the block, generating the so called *Merkle tree*. Each pair of hashes of adjacent transactions are concatenated and hashed to generate a layer of intermediate hashes; if the number of transactions is odd, with one remaining pairless, the following hash is computed duplicating the odd transaction and combining it with its copy. This process is carried out until only a pair of hashes remains in the last layer. Finally, these two hashes are combined to calculate the Merkle root.

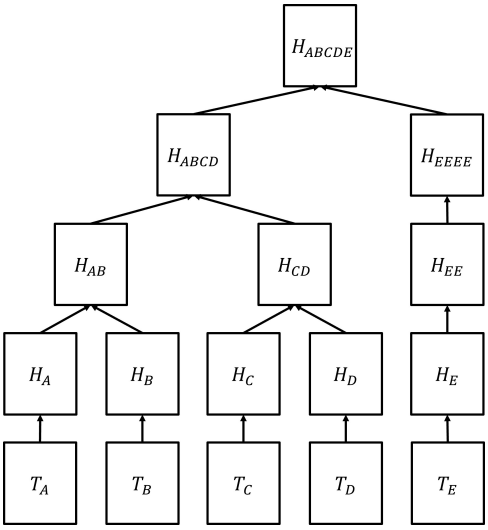


Figure 1.5. Example of Merkle tree with five transactions.

The Merkle tree can be used to verify the presence of a transaction in a block without having to compute the hashes of every transaction inside the block: it is sufficient to know the hash of the transaction to be verified, along with the hashes of the adjacent transaction and those on the path leading from the one in doubt to the Merkle root. For example, if we want to verify the presence of transaction T_B, we need the hashes H_A, H_{CD} and H_{EEEE} to easily compute the missing ones needed to get to the root H_{ABCDE} stored in the header of the block.

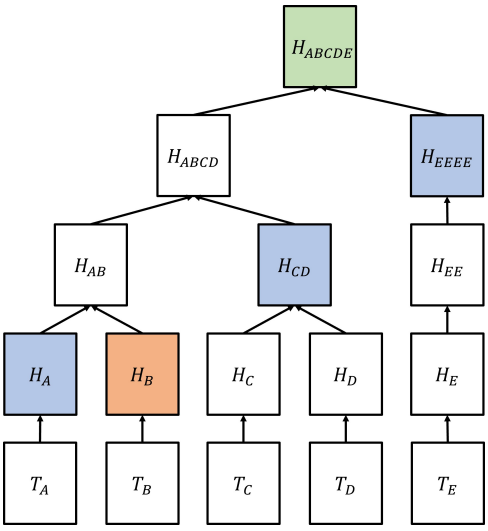


Figure 1.6. Verification process with Merkle tree.

1.5 Consensus protocols

After understanding the structure of a block, the next thing to analyze is how said block is created; this process is one of the most important and difficult concepts related to blockchains. We may recall that the main goal of a blockchain is to create a network between unknown people that do not necessarily trust each other. Hence, the decision regarding who creates a new block, how to include it in the chain, and also the achievement of an agreement on the network's current state, can be extremely problematic. This is why blockchains use *consensus mechanisms* or *protocols*: an algorithm used to approve fair transactions and reject fake or fraudulent ones. The algorithm is previously established and depends on the blockchain; which protocol a blockchain uses is of public knowledge, so a user that wants in in a blockchain must accept the protocol before entering the network. The algorithm is run when a new block must be added to the chain: this is how the blockchain gets updated. These protocols are often used as methods of issuing new currencies on the blockchain: the reward in the form of currency is an additional value to incentivize those involved to keep the network safe.
There are different protocols, each with its own strengths and flaws; we will focus our study on two of them: *Proof of Work (PoW)* and *Proof of Stake (PoS)*.

1.5.1 Proof of Work

Proof of Work is the oldest and most widely used protocol; it is performed by miners that mine new blocks. They generate a block with a specific structure, with the peculiarity of having the so-called coinbase as the first transaction. The coinbase has two main goals: to fix the reward that the miner of that block will receive if the block is added to the chain and to increase the number of cryptocurrency in the market. Indeed, mining a block is the only way to produce new units of cryptocurrency.
Each transaction has an established *fee*: the amount of currency the author of the transaction is willing to pay to have it inserted in the blockchain. These fees are useful to prevent a Denial of Service attack: ill-intentioned users could flood the network with a huge number of transactions; by demanding a fee for each one, malicious actions are scarce.
A transaction must be verified before it can be inserted in a block: all users sharing the same copy of the blockchain must verify that the sender of the transaction actually has the funds to cover its costs. Once verified, the transaction can be added to a block.

Now the block must be validated.

All the verified transactions are stored in a registry from which miners choose which ones to insert in their block. The size of a block is often limited, so miners choose which transactions to insert according to their fees: if the block will be validated and entered in the chain, the miner would receive not only the reward established in the coinbase transaction, but also all the fees of all the transaction in the block.

Now the Proof of Work begins. Miners have to generate a value computing the hash of the whole block; the output digest must be lower than a fixed target value. If the hash

of the block is not appropriate, the miner varies the nonce and tries again. This nonce is a value that can be found in the header of the block and is used only once. Due to the properties of hash functions, the miner cannot know the right input to use to obtain a valid output; therefore, to solve the PoW miners have to make a process of trial-and-error, hence they have to spend a lot of computational resources and energy. This discourages a malicious miner to try and validate one or more fraudulent blocks.

The first miner that solves the PoW sends its block to the network that accepts it as valid and all users have to update the chain adding it to their copy of the database. Once a block is mined, miners move on to the new block and begin to work on the next one. If two blocks are generated at the same time, be it for casualty or for lagging of the network, a fork is created. Most PoW based blockchain choose to work on both branches up until one becomes longer: at that point the longer branch is chosen to be the continuation of the main chain and the other branch is abandoned.
Every two weeks the target value is updated to keep the insertion speed of new blocks under control. Indeed, the goal is to assure that a new block will be mined on average every 10 minutes, whatever computational power the miner holds.
One of the strengths of PoW is its security and the fact that it has been used since the beginning. However, this protocol presents two great flaws: a huge waste of energy and the danger of centralization. In fact, miners that do not have a sufficient computational power gather into a single mining pool to combine all their assets to increase their probability of gaining the reward for mining a block. Proof of Work is becoming increasingly difficult to solve, so that fewer and fewer users have the necessary resources to participate; as a consequence, those who have enough resources could gain control of the network.

1.5.2 Proof of Stake

Proof of Stake is the second best known consensus mechanism. While in PoW miners exploit energy to mine blocks, in PoS validators commit a stake to attest blocks. Validators have two tasks: to propose and to attest blocks on a blockchain.
To become a validator, a user must stake a certain amount of the blockchain currency on the network, that is to freeze it hoping to earn more by proposing a winning block.
There are several mechanism to choose a validator based on:

- randomness, in which the probability of being chosen increases with the amount of currency staked;
- seniority, which represents the relation between the coins owned and the time for which they were owned: seniority is annulled once the validator adds a block and lapses if the tenure time becomes excessive;
- velocity, intended as the rate of use of the currency.

Once chosen, the validator proposes a block that must be attested to by a selected committee: a set of randomly chosen validators that attests to the block. If a sufficient number of members of the committee attests to that block, it is added to the chain. Only after

the block is attested and appended, the validator and the attesters earn their reward. Validators are punished if they do not act on their task when they are selected. For example, if a validator is offline when selected, it is penalized, because to grant an adequate level of scalability of the blockchain the majority of nodes must always be active. These penalties are mild, hence the amount becomes significant only if the validator is offline for the majority of the time.

In a PoS based systems the penalties for misbehaving are higher than with the PoW protocol: users that act maliciously are slashed, meaning that they are deprived of a portion, or even the totality, of what they staked.
PoS based blockchains are more decentralized then PoW ones, because every user that owns a computer and internet connection can be a validator, without the need to spend a lot of money on extremely powerful machines, like in PoW. It is, however, true that they have to stake a certain amount of currency, and for some people the amount could be too high. There are other ways one can get rewards from adding a block even without staking the whole sum: *staking pools*. Users that cannot or do not want to stake the entirety of the amount needed to become a validator can give a portion of it to a node that collects different percentages of the sum from several users. If that node is chosen to be a validator and its block is attested and added to the chain, all the users that staked a certain amount of currency in it get a reward proportional to how much they have staked.
One of the main problems of PoS is the *nothing-at-stake problem*: when a fork is generated, a validator could try to work on both branches because he has stakes on both and it is free. In this way, this validator could try to trick the network and, for example, spend the same currency simultaneously in two different transactions.

1.6 Forks

As already mentioned, if two blocks are generated roughly at the same time, a fork is created: because of the asynchronous nature of the network it is not possible to determine which block was generated a moment before the other.
Each blockchain has a specific rule to solve forks. Both blocks generating a fork are valid and have the same height; each user works to add a block after the one it receives first.

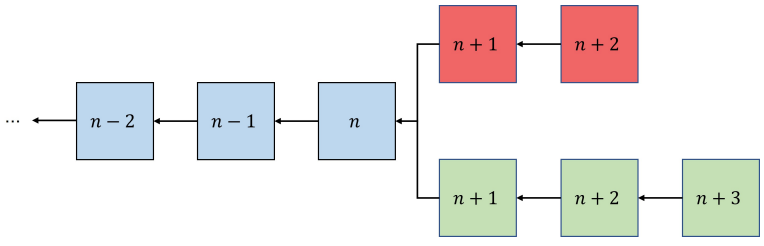


Figure 1.7. Regular fork. Blue blocks are the ones preceding the fork, in green are the accepted ones, in red the orphan blocks.

Usually regular forks resolve themselves fairly quickly because, even if at the start each branch is valid, the moment a new block is appended to one or the other, all users start working on the longer chain, abandoning the other one which is no longer valid.
The blocks that compose the abandoned branch are called *orphan blocks* and the transactions they store are not valid.

A fork is not always generated by the contemporary creation of different blocks. For example, if a group of users proposes to alter the protocol, there are three possibilities:

1. if the entirety of the network accepts the changes, no fork is formed, the protocol is updated and everything works with the new rules;

2. there can be a *soft fork*: the innovators propose a new protocol; there is a period of time in which both the old and new protocols coexist, until everyone changes to the new one. A soft fork is not a real fork, because the blockchain remains unique.

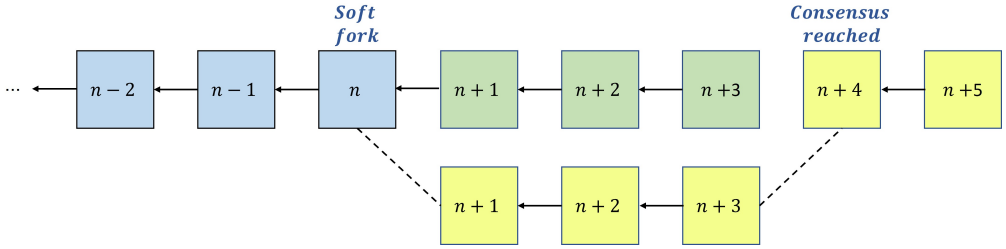


Figure 1.8. Soft fork. Blue blocks are the ones preceding the fork, in green are the blocks that still have to update their system, in yellow the ones already following the new protocol.

3. If the innovators want to drastically change the protocol or if, after a soft fork, some users refuse to exchange protocols, an *hard fork* takes place. After this type of fork there are two different blockchains that continue to develop independently and only share the blocks prior to the fork.

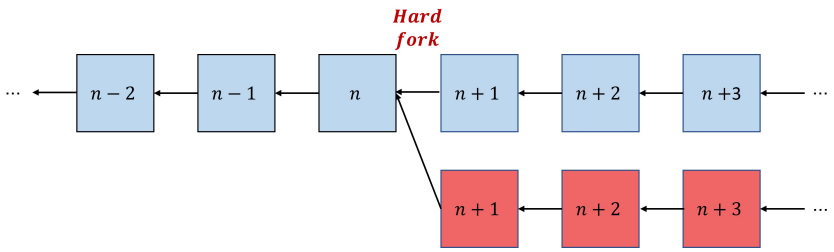


Figure 1.9. Hard fork. Blue blocks are the ones preceding the fork and the one that remains in the starting chain; red blocks are the one that separate from the main chain and form a new blockchain.

1.7 Types of blockchains

Blockchains can be *permissionless, hybrid or permissioned*. In permissionless blockchains, also called public or trustless, everyone can operate as a full node and validate a block using the specific consensus protocol of that blockchain. Everyone can become a user of a public blockchain without the need to be authorized by a central authority. Examples of permissionless blockchains are *Bitcoin* and *Ethereum*. Permissioned or private blockchains, instead, can only be accessed by specific users that need to have a special kind of permission. These users can only do specific actions that are granted to them by the central administration that runs the blockchain. For example, *Ripple* is a permissioned blockchain because it was designed to be used only by mainstream banks. Hybrid blockchains try to use the best features of both permissioned and permissionless blockchains: they are private, but they still guarantee integrity, security and transparency. An example of hybrid blockchain is *XDC*.
Two of the most important blockchains are Bitcoin and Ethereum.

1.7.1 Bitcoin

Bitcoin (B̈) is a digital, anonymous and distributed cryptocurrency developed in 2009 by an anonymous inventor known with the pseudonym of Satoshi Nakamoto. Bitcoin is managed by a peer-to-peer network, it is completely decentralized and its value only depends on its supply and demand. On January 3rd 2009 the genesis block was created and nine days later the first transaction between Satoshi Nakamoto and Hal Finney was registered. This cryptocurrency is managed by a shared permissionless blockchain with the same name. Bitcoin's blockchain is made of blocks with the structure explained before.

Bitcoin does not really have a checking account, because all the transactions are public, so everyone would know the exact amount of bitcoin another user possesses. Each Bitcoin user creates an address and the transactions are simply exchanges of bitcoins from one address to the other. The secrecy and the validity of the transactions are assured by elliptic curve cryptography on the curve $y^2 = x^3 + 7$ ($Secp256k1$) defined on the field $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

For example, if Alice wants to transfer 1 BTC to Bob, first Alice must have received that bitcoin from another person, then she must inform Bob that she wants to send him a bitcoin. Bob creates a pair of keys and computes his address with his public key and sends it to Alice. Alice writes the transaction setting Bob's address as the output address and hers as the input address, where she previously received the bitcoin. Alice sends the transaction to Bob, signed with ECDSA; Bob can now prove to anyone that Alice sent him a bitcoin.

One of the main problems regarding Bitcoin is that it is a blockchain based on a Proof of Work protocol. Bitcoin's miners have to generate a hash value, starting from the nonce, that begins with a fixed number of zeros. This is a really difficult task, hence there is a huge consumption of energy and costs.

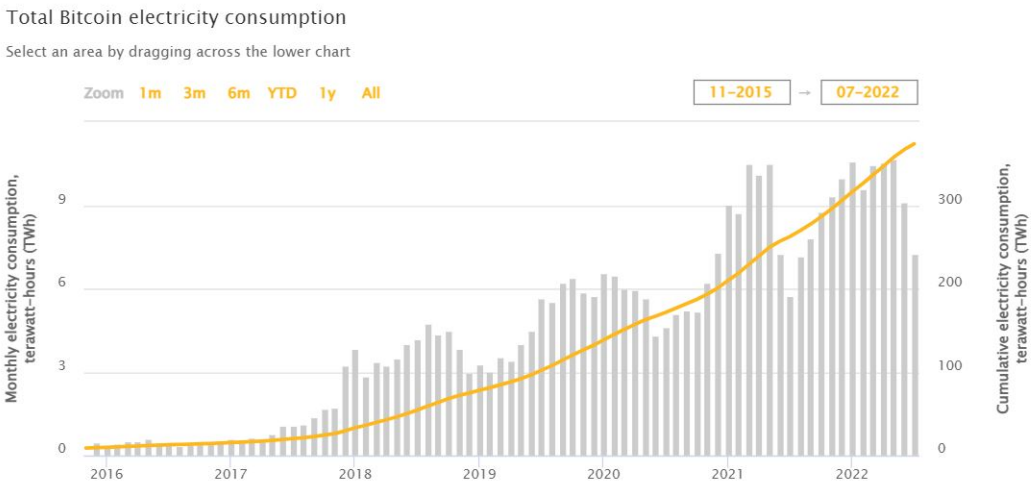


Figure 1.10. Total Bitcoin electricity consumption. Chart taken from Cambridge Center for Alternative Finance.

Another problem is the scalability of the blockchain and its slowness: blocks have a maximum dimension of 1 MB and on average only one block every ten minutes is added to the chain. This implies that if there is a high demand, the blockchain could not manage it in a reasonable time. To fix this problem there are two different solutions:

1. to increase the maximum dimension of a block, to process more transaction at a time; this could translate in a greater centralization because the blockchain could become too big to let common users store its entirety;
2. to decrease the interval in which a block is added to the chain, but this could lead to longer forks that could result in security problems.

1.7.2 Ethereum

First mentioned in the Bitcoin Magazine in 2013 and then in the White Paper from its co-founders Vitalik Buterin and Gavin Wood in the same year, Ethereum's first version was published for the first time on July 30th 2015. It is considered a blockchain 2.0.

Ethereum is a permissionless blockchain with its own cryptocurrency Ether (ETH). This cryptocurrency can be used as bitcoin to make transactions and trading, but more importantly it is used to aid Ethereum's main goal: to create a distributed platform in which *Smart Contracts* can run. Once a Smart Contract is launched, it cannot be stopped or changed.
Common web applications are used by different users in various places, but the application runs on a central server which contains all the necessary data. A Smart Contract is an application run on a peer-to-peer decentralized network. The main characteristic of Smart Contracts is the decentralized code that runs over thousands of servers and is run

in parallel: if a node fails or refuses to run the code, it is still run on the computers of all the other users of the network. Blockchains, with Smart Contracts, assure that all the users of the decentralized network run the same applications with the same data; at the same time, they also guarantee that all users obtain the same output. Smart Contracts transform the terms of an agreement into algorithms that are automatically executed when the terms of the agreement are met. Therefore, trust is no longer necessary for a contract to be fully respected by all parties.

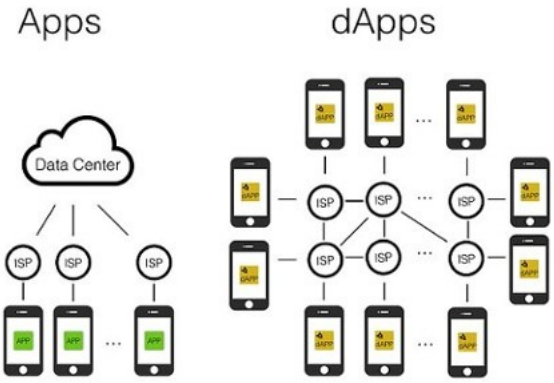


Figure 1.11. Common apps VS decentralized apps. Picture taken from the Cryptonomist.

Ethereum provides a virtual machine *(EVM, Ethereum Virtual Machine)* used to carry out the scripts. EVM is an emulator that can perform on any computer and allows to run applications designed specifically for this environment. There are various languages used to program a Smart Contract within EVM, the most popular being Solidity and Vyper. Even on Bitcoin it is possible to run scripts, but these are simple and with scarce possible operations, while EVM does not have limits. Indeed, loops are not programmable on Bitcoin, while EVM is Turing complete. Because of this characteristic, Ethereum is susceptible to attacks aimed at blocking the network, which are prevented by charging a tax, *gas*, to execute a Smart Contract: it can be carried out as long as there is gas available. Each operation consumes a specific amount of gas, proportional to the amount of computational work required to carry it out; for example, a simple ETH transfer consumes 21000 units of gas. The maximum amount of gas that can be used by a Smart Contract is bounded by the blockchain itself; users determine how much gas they are willing to pay to carry out their Smart Contract and define it in the Smart Contract itself: if the operations contained in the code consume a quantity of gas that is less than or equal to what the user is willing to pay, then the contract is carried out and the difference between what was available and what was consumed is returned to the user; if the user establishes a maximum amount that is not sufficient to carry out the whole contract, it is interrupted and the user can decide either to raise it or abandon the Smart Contract.
Gas is used to measure the computational cost of an operation carried out by the EVM.

There is a 1:1 correspondence between gas and ether: the latter is converted into gas just to carry out the operation; then, if there is an excess of gas, it is turned into ether and given back to the user.

A *Decentralized Application (dApp)* is an application composed of a collection of Smart Contracts that run on a distributed platform without a central authority.

Starting an organization with unknown people is not easy, trust must be established and money has to be exchanged without any guarantees; thus, this is a natural environment for blockchains, especially for Smart Contracts. Trust between people is not needed, trust in the Smart Contract is enough. This is where *Decentralized Autonomous Organizations (DAOs)* come into focus: a DAO is an organization owned by a collective of people, who share the same goal, controlled by a blockchain and ruled by Smart Contracts. The specific Smart Contract is public for everybody to see and check; once it is online, it cannot be changed, except through a vote. The Smart Contract also administrates the funds of the DAO, so no one can transfer money without everyone's permit. If a DAO generates a profit, it is divided intro equal parts to all the member of the organization.

Two types of account are tied to the Ethereum platform: user accounts and contract accounts. User accounts are externally owned ones (EOA), managed by private keys; they can send messages both to other externally owned ones and to contract ones: if the message is sent between two users account, it consists simply in a transaction which must be signed with the algorithm ECDSA in order to be valid; if the message is sent from an externally owned account to a contract one, it activates the contract's code and enables it to perform different actions. An EOA address always starts with $0x$ and then presents the last 20 bytes of the hash function Keccak-256 of the public key, obtained from the private key with ECDSA. Contract accounts are Smart Contracts, controlled by the contract code they contain and are the only ones with an associated set of functions; they cannot create new transactions and they can only send messages in response to the transactions they received. The contract address is assigned only when the contract is sent into the network; it is of the same format of the user ones but it is generated from the creator's address and the number of transactions he sent.

A wallet is always needed to handle a cryptocurrency tied to a blockchain. This is also the case of ether on Ethereum. Wallets are not storage units, because ETH are not really saved in them, they are just software needed to manage the keys of the accounts containing the cryptocurrency. A user can, at any time, switch wallet providers; some wallets also allow to use multiple Ethereum accounts on the same application.

Since the beginning, Ethereum's development was organized into four phases, each one representing a major update from the previous version, therefore each one can be considered a hard fork.

1. **Frontier phase.** The main goal of this phase was to launch the network and provide it with two basic functions: to mine new ether and to run Smart Contracts. In this

phase a soft fork happened, called Frontier Thawing, whose purpose was to limit gas costs to prevent centralization.

2. **Homestead.** Frontier's security problem was brought to light by the DAO hack: DAO was an idea to allow users to store funds but, due to a bug in its Smart Contract, ill-intentioned users stole the organization's funds. Therefore the Homestead hard fork was made to increase network security.

3. **Metropolis.** This phase was aimed at improving Ethereum's security, privacy and scalability. Because of its complexity, this update was performed in two steps: Byzantium and Constantinople. The first introduced the so called Ethereum improvement protocols; the latter, in addition to fixing small problems of Byzantium, introduced the basis for the transition from PoW to PoS.

4. **Serenity.** The goal of this phase is to render the network more scalable, secure and sustainable. This phase involves the actual passage to Proof of Stake, to reduce the waste of resources of PoW, and the introduction of shard chains, to divide the workload of the network on different chains running in parallel.

Up until the start of September 2022, Ethereum was referred to as *execution layer* and was based on a Proof of Work protocol named *Ethash*. As for Bitcoin, Ethereum's blocks contains a block header and a certain number of transactions; the protocol for validating them consisted in finding two information that had to be stored in the block's header: the *mixHash*, a 256-bit hash, and the *nonce*, a 64-bit value. Combining these two quantities one could prove that a sufficient amount of work had been carried out on that block. The algorithm was composed by the following steps:

1. calculation of a *seed* for each block;

2. use of this seed to compute a pseudo-random *cache*, a small memory from which high-speed retrieval is possible;

3. generation of the *DAG*, starting from the cache: a dataset where each element depended on a small number of data from the cache, chosen pseudo-randomly. To become a miner, a user had to generate the whole dataset that grew linearly with time;

4. generation of the mixHash through the combination of multiple hashes of different subsets of the dataset;

5. confrontation of this mixHash with the target nonce: if the mixHash was smaller than the nonce, the latter was considered valid and the block could be added to the chain; if it was not, the process had to be repeated.

Ethereum's blocks also stored a list of *ommer block* headers: headers of blocks whose parent had the same height of the current block's parent's parent; they were also informally referred to as uncle blocks.

Due to Ethereum's structure, the average block time is low, circa 12 seconds, unlike Bitcoin's one that is approximately 10 minutes. Therefore, Ethereum's miners process transactions way faster than Bitcoin's ones: this feature could cause multiple blocks to be generated at the same time, hence, many orphaned blocks. To avoid wasting miners' work, ommer blocks were introduced: their main goal was to reward miners for including these blocks, even if they did not get chosen to be part of the main chain. These blocks had to be valid, hence they could not be an uncle of a block with more than six child blocks; after reaching the sixth generation, the ommer block could no longer be considered. Miners got a smaller reward for mining an ommer block than a full block; this reward was regulated by the *GHOST protocol*.

This version of the network has always been a transitional one: the main purpose was to generate one aimed at resolving the trilemma of blockchains, that is achieving decentralization, scalability and security, known as *consensus layer*. Its launch takes place in three main steps:

1. the Beacon Chain;
2. the Merge;
3. fully functioning shards.

On December 1st, 2020, the Beacon Chain was created: a separate Proof of Stake blockchain that runs in parallel to the Proof of Work Mainnet one. At first, its task was to register the addresses of all active validators and keep track of their account balances and it did not enter into the merits of processing Mainnet transactions. To become a validator, a user must stake exactly 32 ETH; staking more than 32 ETH in a single validator is not useful: the probability of being chosen of a validator with a balance of 32 ETH is the same of one with a balance of 64 ETH. If a user wants to stake more than 32 ETH, it is better if he stakes it activating a separate validator.
Together with the Beacon chain, we can introduce a temporal division of the blockchain. The period of time in which a block can be proposed by a validator is a *slot* and it amounts to, approximately, 12 seconds. An *epoch* is composed of 32 slots, so it takes circa 6.4 minutes to go through one. At the start of each epoch, *validator committees* are regrouped and each validator is assigned new responsibilities, for security reasons. During each epoch, the chain has the possibility of being *finalized*, meaning that a set of transactions prior to the time of finalization cannot be changed or reverted. A validator committee is a group of at least 128 randomly chosen validators, whose duty is to validate blocks in each slot.

At 8:45 of September 15th, 2022, the Merge was completed and the PoW consensus mechanism was replaced with a Proof of Stake one. This transition reduced energy consumption by $\sim 99.95\%$ and rendered the network significantly more secure and decentralized. This process consisted in the merging of the original Ethereum Mainnet and the Beacon Chain in a single chain. After the unification, the Beacon Chain became the centre of blocks production.
To help in this changeover from a Proof of Work based blockchain to a Proof of Stake

one while minimizing the risks, *Casper The Friendly Finality Gadget (Casper-FFG)* was introduced. It is a Proof of Stake finality system that is paired with a Proof of Work based proposal mechanism.
In a Proof of Work based network, ommer blocks were useful to prevent a prolonged fork: if miners got rewards from mining ommer blocks, they would be more accepting of the fact that their block would not be appended to the main chain. Under Proof of Stake, the network will know automatically which branch of a fork is the appropriate one. We note that, even if greatly rare under PoS, reorganization of the network could always occur due, for example, to latency.

The last step for the complete realization of the consensus layer is the introduction of shard chains. The main problem in scalability that every blockchain faces is that each node must verify each and every transaction. Sharding could help solve this problem: the idea is to place side by side the Beacon Chain with these shard chains, secondary chains used for the verification of certain transactions only. These shard chains operate simultaneously and communicate with each other through specific Smart Contracts. The possibility of working on different transaction simultaneously could help increase the scalability of the chain. Shard chains are expected to be fully functioning in 2023.

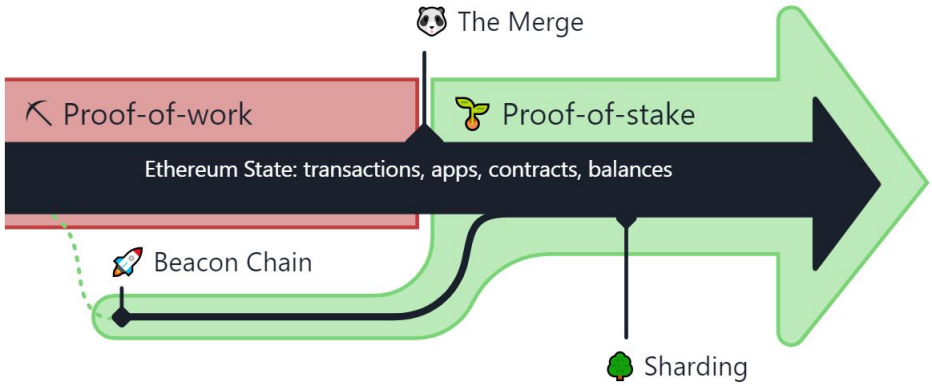


Figure 1.12. The Merge.

The centre of our work is to understand the functioning of the *Gasper protocol*: a Proof of Stake consensus mechanism deriving from the combination of LMD-GHOST, a variant of the GHOST fork choice algorithm, and Casper-FFG.

Chapter 2

Greedy Heaviest-Observed Sub-Tree

With time, blockchains are gaining importance and consensus; this is why one of their main problem is scalability: more and more people use blockchains to make safe transactions, so the throughput a network is subject to is always bigger. Consequently, the need to process an ever-increasing number of transactions in a short period of time rises. Another problem that emerges is the synchronization of the network: if a node receives two conflicting blocks a fork is generated. The goal of this chapter is to present the first element that composes Ethereum's Gasper Proof of Stake protocol, the *Greedy Heaviest-Observed Sub-Tree* fork choice rule. It is an alternative algorithm to solve forks, which also eases the scalability problem of the network.

Same as Bitcoin, Ethereum's execution layer utilized the fork choice rule consisting in the selection of the longest chain, determined by the chain's total mining difficulty. With the introduction of the Beacon Chain, which represents the start of the Proof of Stake era for Ethereum, the GHOST fork choice rule was adopted.
One of the problem the longest chain rule led to is the 51% *attack*: if an attacker holds at least 51% of the total mining power of the network, he would be able to generate a fork with a chain longer then the one created by the honest nodes, which hold less than the 50% of the total power. Thus, the malicious chain would be longer than the honest one and would be adopted as the main one by the protocol.
With the introduction of a Proof of Stake consensus protocol, Ethereum stopped using the longest chain rule and started following the *Greedy Heaviest-Observed Sub-Tree (GHOST)* rule: instead of choosing the longest chain, the one corresponding to the heaviest sub-tree is chosen.

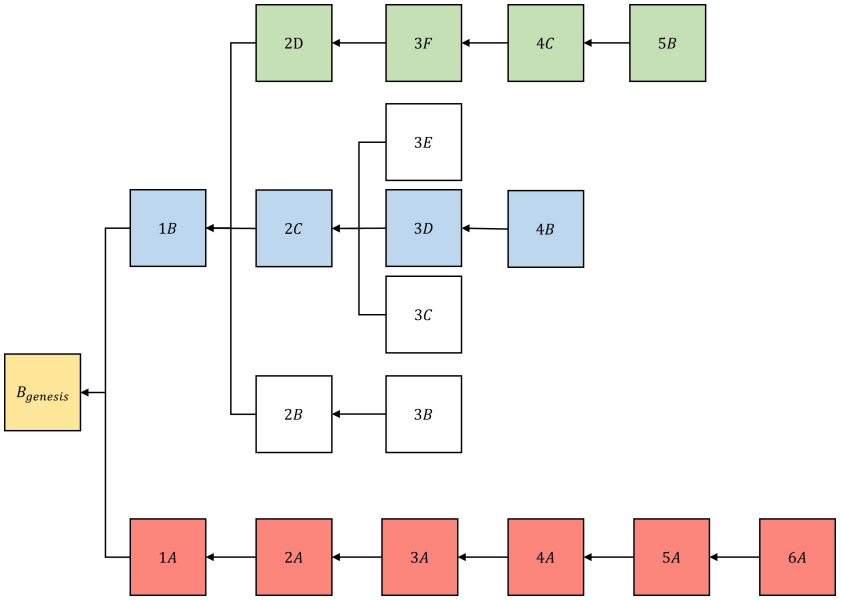


Figure 2.1. Comparison between chains: the longest chain is composed of the genesis block, block $1B$ and then the green blocks; the attacker's chain is composed of $B_{genesis}$ and then the red blocks; the GHOST chain is composed of the genesis block and the blue blocks.

This change was necessary because it helped solve the scalability problem, making it possible to safely increase the number of transactions processed in the network in a short period of time and fix the 51% attack, giving importance not only to the blocks on the main chain, but also to the ommer blocks, which contribute to the heaviness of the sub-tree.

It can be proved that a blockchain that follows the longest fork choice rule, like Bitcoin, is more vulnerable, in case of high transaction flow, than one that follows the GHOST rule, like Ethereum.

It is possible to verify the rightfulness of the previous claim with the following reasoning.

Bitcoin's network can be modeled as a directed graph $G = (V, E)$, where V is the set of users v and E is the set of edges e between users. Each user $v \in V$ possesses a percentage p_v of the total computational power of the network. These shares must be such that

$$\sum_{v \in V} p_v = 1.$$

The whole network inserts new blocks following a Poisson process with rate λ; therefore, each individual user's creation rate is proportional to the percentage of computational power held, that is $p_v \lambda$.

Definition 2.0.1 *For every edge $e \in E$, the delay d_e is the time needed to send a block along that specific edge.*

Definition 2.0.2 *Given a block B, $time(B)$ is its creation time t; $subtree(B)$ is the tree rooted in B; $depth(B)$ is its depth in the network tree.*

Let $T = tree(t)$ be the network state at time t, with set of validators V_T and set of edges E_T. The deepest leaf in tree T at time t is indicated by $longest(t)$.

Definition 2.0.3 *A function $s(\cdot)$ that maps the tree T to a block $B \in V_T$ represents the policy by which each node chooses a parent block, i.e. block B, for the new block it has just created.*

Definition 2.0.4 *The time required for the length of the tree to change from $n-1$ to n is defined as a random variable τ_n.*

This random variable represents the needed time to add a block to the tree. It is possible to define τ as:

$$\tau = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \tau_i.$$

Defining the following parameters

Definition 2.0.5 *λ as the rate of block addition to the tree and $\beta = \frac{1}{\mathbb{E}[\tau]}$ as the rate of block addition to the main chain.*

Denoting with b the maximal block size, measured in KB, the number of transactions per second is

$$TPS = \beta(\lambda, b) \cdot b \cdot K,$$

where K is the expected number of transaction per KB.

Suppose that an attacker possesses a block creation rate equal to $q\lambda_h$, with $0 < q < 1$ and λ_h being the creation rate of honest users.

- If $q\lambda_h > \beta$, the attack is always successful, given enough time, despite the length of the actual main chain the attacker has to surpass, by The Law of Large Numbers.
- If $q\lambda_h < \beta$, the attack never succeeds, because the probability of the malicious chain surpassing the honest one goes to zero exponentially with the growth of the honest chain's length.

Therefore, the quantity $\frac{\beta}{\lambda_h}$ serves as a safety threshold.

The network throughput is influenced by both the block creation rate λ and the maximal block size b. To increase it, one could think to raise both these parameters, but this leads to a security problem, due to the increased fork generation rate. Indeed:

- if b is increased, this leads to a bigger delay time d_e for each edge e; therefore, a block will need more time to be broadcast across the entire network. However, in this period of time nodes not aware of the already existing block will generate new ones, so a large number of forks will be generated;
- if λ is increased, nodes will generate a lot of blocks in the time required from one to be sent across the entire network; this will lead to multiple valid blocks, causing the generation of many forks.

Both cases raise the number of generated forks and therefore a decrease of security: the main chain will be shorter, so an attacker will need less computational power to generate a longer, malicious chain.

This is why the GHOST protocol is so important: it can be proved that GHOST will increase the throughput of transaction processed by the network, increasing both block size and block creation rate, without affecting the security of the system.

2.1 GHOST protocol

GHOST - Greedy Heaviest-Observed Sub-Tree - is a new parent selection policy with security threshold $\frac{\beta}{\lambda_h} = 1$. It is based on the concept that even ommer blocks can contribute to the importance of a subtree making it heavier, in the sense of the total computational power spent in there.
Given a block B in tree T and being $Children_T(B)$ the set of blocks belonging to T, whose direct parent is B, the protocol adopts the following algorithm:

1. set B as the genesis block
2. if $Children_T(B) = \emptyset$ then $return(B)$ and $exit$
3. else update B with $\arg\max_{C \in Children_T(B)} |subtree_T(C)|$
4. go to line 2.

The algorithm starts at the genesis block and goes on choosing the block that leads to the heaviest chain, until finding a leaf; this leaf will be the new parent block. For example, in the case of figure 2.1 the process is the following:

1. set $B = B_{genesis}$ as the genesis block
2. $Children_T(B) \neq \emptyset$, so continue to the third step
3. update B with the node that is parent to the heaviest sub-tree, that is $B = 1B$
4. go back to step 2
2. $Children_T(1B) \neq \emptyset$, so continue to the third step
3. update $B = 2C$

4. go back to step 2

2. $Children_T(2C) \neq \emptyset$, so continue to the third step

3. update $B = 3D$

4. go back to step 2

2. $Children_T(3D) \neq \emptyset$, so continue to the third step

3. update $B = 4B$

4. go back to step 2

2. $Children_T(4B) = \emptyset$, so the end of the algorithm has been reached: the parent of the new block will be block $4B$.

One of the main properties of a blockchain that follows the GHOST protocol is the *Convergence of History*, which states that all nodes of the network will, ultimately, adopt the same history, which is the canonical chain.
Defining the collapsing time ψ_B of a fork at block B as the earliest moment in which the block is either accepted or abandoned by the whole network, the following property holds:

Property 2.1.1 (Convergence of History) $\mathcal{P}(\psi_B < \infty) = 1$ *and* $\mathbb{E}[\psi_B] < \infty$.

The property also states that the collapsing time is finite.

Previously, it has been stated that the security threshold of the GHOST protocol is equal to 1; this is the main advantage of this protocol: it is safe even with a high throughput of transactions. This is due to the following property:

Property 2.1.2 (Resilience to 51% attack property) *Suppose that the block creation rate of the honest nodes is λ_h and that of an attacker is $q\lambda_h$, with $0 \leq q < 1$. The probability of block B being abandoned from the main chain after time$(B) + \tau$, given that it was accepted on it at time$(B) + \tau$, goes to zero as τ grows to infinity.*

This property reports that, after a time τ sufficiently higher than the block creation time, the likelihood of the block being abandoned, after being previously adopted, can be made arbitrarily small.

It is possible to give an estimation of the rate of collapse in GHOST from the length of a fork's point of view. If we consider the case of a network with only two forks, each one having equal computational power, the following theorem is valid:

Theorem 2.1.1 *Consider a network with two nodes, u and v, that equally create blocks at a rate of $\frac{\lambda}{2}$, which are connected by a single link with delay d. For any block B, $\mathbb{E}[n_B] = \frac{(d\lambda)^2}{8} + \frac{d\lambda}{2}$, where $n_B \coloneqq subtree_T(B)$ for $T = tree(\psi_B)$. [19]*

The theorem gives an upper bound for the size of a subtree in the worst case possible of only two nodes. In a realistic scenario, collapses occur faster than what the theorem estimates.

2.2 Comparison between longest chain rule and GHOST

To give an appropriate comparison between the longest fork choice rule and GHOST, parameter β must be analyzed in both protocols. It represents the growth rate of the main chain. Since this value is highly influenced by the topology of the network, which can be either unknown or quite difficult to measure, first a lower and an upper bound to the growth rate are given, then the protocols must be tested with randomly sampled topologies and the resulting networks must be measured.

Consider a section of the network composed by a set of greatly connected nodes, with delay diameter D, that owns a portion $0 \leq \alpha \leq 1$ of the total computational power. In this section of the network, due to the connectedness of the components, blocks travel at great speed from a node to another; thus, there is a tight lower bound for both the longest chain and the GHOST rules.

Let $G = (V, E)$ be the considered subgraph of the entire network and assume that G produces blocks at a rate $\lambda' = \alpha\lambda$.

- If the longest chain rule is applied, then the longest chain grows at a rate $\beta(\lambda) \geq \frac{\lambda'}{1+\lambda' D}$.

  This can be deduced from the following reasoning: after block B, with $depth(B) = n$, is created, it is sent to the whole network in D seconds. It has to be expected that another block B' will be created after $\frac{1}{\lambda'}$ seconds; due to the fact that the creator of block B' already received block B, block B' must have depth at least equal to $n+1$. This means that the longest chain grows at a rate that can be bounded by $\frac{1}{D+\frac{1}{\lambda'}}$, that is $\frac{\lambda'}{1+\lambda' D}$.

- If the network adopts the GHOST rule, then the longest chain grows at a rate $\beta(\lambda) \geq \frac{\lambda'}{1+2\lambda' D}$.

  Due to the fact that GHOST does not choose the longest chain, but the heaviest one, a growth rate lower that the longer chain rule's one can be expected. A formal demonstration of this follows from the claim that the expected waiting time for the creation of the last child of a block is upper bounded by $2D + \frac{1}{\lambda'}$. This means that the growth rate is lower bounded by $\frac{1}{2D+\frac{1}{\lambda'}}$, that is $\frac{\lambda'}{1+2\lambda' D}$.

The system reaches the exact lower bound if it is a complete network composed of n nodes, with $n \to \infty$, where each edge has exactly a delay equal to D and possesses a fraction of $\frac{1}{n}$ of the total computational power. This would be the case of an ideal total decentralized network where all nodes have the same computational power.

As aforementioned, the network's topology is of little knowledge: it could be very difficult to compute, in real cases, or it could be constantly changing - for example, nodes can be either connected or disconnected in different instants. This limited knowledge can

be useful to build a system with a specific desired security; indeed, if a network that follows the longest chain rule has a block creation rate of $\alpha\lambda$, block size b and delay diameter $D(b)$, then the protocol achieves both a high number of TPS and an high security threshold, for appropriate choices of λ and b.

A network using the GHOST choice rule has a security threshold always equal to 1. This could lead to think that there is no limit to the maximal throughput the system could be subject to. However, this is not true: a high number of TPS consumes bandwidth, hence, reduces the efficiency of the system. To keep track of this important characteristic of the network, the ratio between the block size and the block creation rate is taken into consideration.

The upper bound is the same for both the longest chain rule and GHOST. Taking two partitions, $S, T \in V$, of the entire set of nodes with different portion of computational power, $p_S \neq p_T$, and such that $\forall s \in S$ and $\forall t \in T$, $d_{\{s,t\}} \geq d$, it can be shown that the growth rate β can be bounded as follows:

$$\beta(\lambda) \leq \frac{(p_S\lambda)^2\, e^{p_S\lambda 2d} - (p_T\lambda)^2\, e^{p_T\lambda 2d}}{p_S\,\lambda\, e^{p_S\lambda 2d} - p_T\,\lambda\, e^{p_T\lambda 2d}}.$$

The equality is reached only by a network with only two nodes.

Running some tests on made-up systems following Bitcoin's model, it can be observed that the security threshold for GHOST is, actually, always 1, while that of the longest chain rule greatly decreases even when the block creation rate is slightly increased. Also, the difference in efficiency using the GHOST protocol or the longest chain rule is rather small.

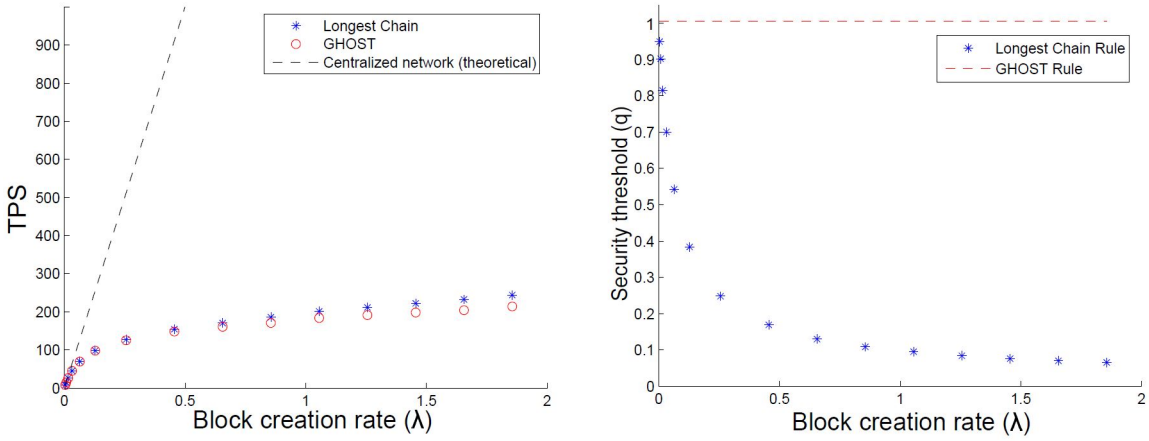


Figure 2.2. Transaction per seconds and security threshold in relation to block creation rate. Graphs taken from [19].

Chapter 3

Casper the Friendly Finality Gadget

The Gasper protocol contemplates, besides a fork choice rule, a mechanism to establish valid blocks that will not change their status over time. This mechanism is *Casper the Friendly Finality Gadget*. To better understand Casper's functioning, firstly a basic version of it is explained, with just the property of accountability; secondly, a dynamic set of validators is introduced, to improve the link between the theoretical mechanism and reality. Lastly, defenses against two types of attack are presented as a result of Casper's properties.

Casper is a Proof of Stake finality system whose goal is to finalize blocks generated by a proposal mechanism based on a Proof of Work consensus protocol. Casper follows the *Byzantine Fault Tolerant (BFT)* based Proof of Stake school of thought, introducing new features not necessarily owned by classical BFT protocols.

Property 3.0.1 *Byzantine Fault Tolerance is the property of a system to withstand failures derived from the Byzantine Generals Problem.*

The BFT intervenes in systems where it is necessary to reach a consensus between different information that can conflict with each other. The link existing between BFT and blockchains becomes clear: in the latter it is fundamental to reach a distributed consensus regarding validation of new blocks and selection of a single chain, in situations where users of the network can be malicious.
The Byzantine Generals Problem is an hypothetical scenario in which a certain number of troops of the Byzantine empire's army surrounds an enemy region. Troop's generals must reach consensus on either attacking or retreating. To avoid complete failure, the majority of them have to perform the same action. The problem rises because different generals communicate with each other only through a messenger that could be captured; thus, the message could never be delivered. Another problem could be if some generals would be traitors and send different messages to different honest generals. To solve this problem, Lamport, Shostak and Pease [12] developed the *Oral Messages (OM)* algorithm. For the

algorithm to work it is necessary that once sent, each message reaches its rightful receiver, the addressee must exactly know who is sending the message and it must be possible to know when a message is missing.
Lamport, Shostak and Pease came to the conclusion that the problem is solvable if the following theorem holds.

Theorem 3.0.1 *Given m malicious generals, the OM algorithm reaches consensus if the number of total generals is* $> 3m$.

Byzantine Fault Tolerance based Proof of Stake is a particular protocol in which several staked users, those who stake part of their balance to participate in the protocol, are selected to propose a block. Once all blocks are proposed, all staked users must attest to the block they want to add to the canonical chain; several rounds of voting may be necessary before a block is chosen. This variation of the protocol adds complexity to the classical Proof of Stake because all staked users have an active role on the selection of each new block added to the network.

As previously said, Casper is a gadget used on a Proof of Work proposal mechanism; the latter is responsible for the liveness of the network, the property that ensures that something good will eventually happen; meanwhile, Casper is in charge of the safety of the system, the property that assures that something bad will not happen.
The liveness property can be referred to as *termination* in a network where the main goal is to reach consensus: each staked user, at the end of the voting process, will decide on a block. This property cannot be violated because that 'something good' can still happen afterwards. The safety property can be referred to as *agreement*, meaning that in the network there will never be the case where two blocks are decided upon.

Casper is not a classical BFT based PoS; indeed it introduces four main properties to the protocol: accountability, dynamic validators, defenses and modular overlay.

Property 3.0.2 (Accountability.) *It is always possible to know if a user violates a rule. If this happens, the system will always know which user acted maliciously.*

The accountability property enables the network to penalize malicious validators, with the *slashing* of its currency, fixing the nothing-at-stake problem. Slashing means to deprive a malicious user of part or the totality of their balance.

Property 3.0.3 (Dynamic Validators.) *The set of validators may not be static, indeed it can change over time.*

With this property, Casper introduces the possibility for a validator to leave the network or to join in at various time. It is important to note that once a validator leaves the network, he could not join the system again later, it is permanent, because its public key will be forever banished from the set of validators' keys.

Property 3.0.4 (Defences) *The protocol is protected against long range attacks and catastrophic crashes.*

Casper introduces resolutions for these two types of attack, rendering the network safer.

The last property is intrinsic in Casper's definition itself:

Property 3.0.5 (Modular overlay) *Casper is a gadget: it is used as an overlay system over an already existing consensus protocol.*

3.1 The Casper Protocol

Since Ethereum was first born as a Proof of Work based blockchain, Casper's first version is a hybrid PoW/PoS system: blocks are proposed by Proof of Work and finalized by Casper. With Ethereum's passage to Proof of Stake, Casper must be adapted to be used with a Proof of Stake proposal mechanism.

3.1.1 Simple version

The simple version of the protocol considers a fixed set of validators and a Proof of Work proposal mechanism.

Under normal circumstances, the proposal mechanism generates, for each parent block, only one child block; it can happen that, under attack or because of network latency, the mechanism proposes two child blocks for a single parent one: Casper's duty is to choose, through a *fork choice rule*, one of these two child blocks.

The enunciation of this rule must be preceded by some definitions.

Definition 3.1.1 *A checkpoint is the first block validated in each epoch.*

For efficiency, Casper considers only the subtree of checkpoints.
An absolute principle is that the genesis block is a checkpoint, because it is the first block of the slot 0 in epoch 0; thus, it is the root of the checkpoint tree.

Property 3.1.1 *If m is the spacing between checkpoints, every block with height block equals to a multiple of $(m+1)$ is a checkpoint.*

The spacing m should be big enough to reduce the overload of the algorithm but small enough to assure an efficient computational time. The number k for which $m+1$ is multiplied is the checkpoint height of that specific checkpoint, i.e. the number of blocks in the chain from the root to the checkpoint itself - not counted.

The deposit of a validator is the amount of coins deposited; this quantity can be increased through rewards and decreased with penalties for acting badly. In each voting round, validators cast a vote

$$\langle v, s, t, h(s), h(t) \rangle.$$

Each vote consists in a message containing four information:

1. a checkpoint s denominated the *source*;
2. a checkpoint t called the *target*. It must be chosen such that s is one of its ancestor, lest the vote be invalid;
3. $h(s)$, that is s' height;
4. $h(t)$, that is t' height.

There exists two reasons for which a vote could be invalid: either s is not an ancestor of t, or the public key of the validator that casts the vote is not in the validator set.

Definition 3.1.2 *A supermajority link is an ordered pair (a, b), also symbolized by $a \to b$, such that a set of validators who owns, together, at least $\frac{2}{3}$ of the total deposit have cast a vote with a as source and b as target.*

Definition 3.1.3 *Two checkpoints a and b are said to be conflicting if and only if neither a nor b is one the ancestor of the other, hence they belong to two different branches of the tree.*

Definition 3.1.4 *A checkpoint c is justified if it is the root or if there exists a supermajority link $c' \to c$ between c and a justified checkpoint c'.*

Definition 3.1.5 *A checkpoint c is finalized if it is the root or if there exists a supermajority link $c \to c'$ between the justified checkpoint c and one of its direct children c'.*

This means that checkpoint c is finalized if and only if it is justified, there exists a supermajority link between c and a checkpoint c', non-conflicting with c, for which it is valid that $h(c') = h(c) + 1$

Casper's safety is based on two *Commandments*, also referred to as *slashing conditions*. Any validator that violates one of these conditions gets its deposit slashed. The two slashing conditions are:

I. no validator can cast two different votes

$$\langle v, s_1, t_1, h(s_1), h(t_1)\rangle \text{ and } \langle v, s_2, t_2, h(s_2), h(t_2)\rangle \tag{3.1}$$

where the targets have same height, that is $h(t_1) = h(t_2)$;

II. no validator can cast two distinct votes for which the following inequality holds

$$h(s_1) < h(s_2) < h(t_2) < h(t_1); \tag{3.2}$$

If a validator violates either of these slashing conditions, another validator can send the proof of this violation to the network as a transaction, and the validator's whole deposit is taken away. The validator that submitted the proof of the malicious action receives a reward.

Casper's main properties are :

Property 3.1.2 ***Accountable safety***. *Two conflicting checkpoints a_m and b_n cannot both be finalized, unless at least $1/3$ of validators by weight[1] violates one of the two slashing conditions.*

Property 3.1.3 ***Plausible liveness***. *Supermajority links can always be created to generate new finalized checkpoints, as long as the finalized chain can be extended.*

This last property connotes the ever-present possibility of finalizing new checkpoints if at least 2/3 of validators follow the protocol without violating any slashing condition.

Assuming that less then a third of validators violates one of the commandments, from the slashing conditions immediately follows that:

1. if (s_1, t_1) and (s_2, t_2) are two distinct supermajority links, then necessarily the targets have different height, $h(t_1) \neq h(t_2)$, and the inequality $h(s_1) < h(s_2) < h(t_2) < h(t_1)$ cannot be verified;
2. for any fixed height n, there exist at most one checkpoint of that specific height; also there exists at most one supermajority link where the target has height n.

Casper's fork choice rule can be enunciated through the following theorem:

Theorem 3.1.1 *Follow the chain containing the justified checkpoint of the greatest height.*

This rule is correct by construction because it is based on the plausible liveness property: the set of finalized checkpoints will always grow so that one with greatest height will always exist.

Note that with the introduction of the Gasper protocol, this rule was changed. This shift is described in the next chapter.

3.1.2 Inclusion of the dynamic set of validators

It is extremely important to be able to enable a dynamic set of validators in the Casper mechanism, because in real systems users can be online for some time and then decide to leave the network. Therefore, a new validator should be able to join and an existing one should be able to leave.

Definition 3.1.6 *Dynasty of a block B is the number of finalized checkpoints from the root to B's parent.*

When a new validator wants to join the set, it must send a *deposit message*. If this message is included in a block with dynasty d, then the validator officially joins the set at the first block with dynasty $d + 2$. That is, indicating with $DS(v)$ the *validator's start dynasty*,

$$DS(v) = d + 2.$$

[1] *The term validators by weight indicates validators that own a certain portion of the total deposit.*

Similarly, when an existing validator wants to leave the network, it has to send a *withdraw message*. If this message is included in a block with dynasty d, then the validator officially leaves the set at the first block with dynasty $d + 2$. Hence, indicating with $DE(v)$ the *validator's end dynasty*,

$$DE(v) = d + 2.$$

Once a validator v leaves the set, its public key is forever banned from the validator set, meaning it cannot join again.

Definition 3.1.7 *The withdrawal delay is a long period of time, at the start of the validator's end dynasty, in which a validator's deposit is locked.*

If during this time the validator violates at least one of the commandments, its deposit is slashed.

Definition 3.1.8 *For any given dynasty d, we can define*

- *the forward validator set as*

$$V_f(d) \equiv \{v : DS(v) \leq d < DE(v)\};$$

- *the rear validator set as*

$$V_r(d) \equiv \{v : DS(v) < d \leq DE(v)\}.$$

Note that, by construction, we have $V_f(d) = V_r(d + 1)$.

3.1.3 Defenses against attacks

Casper provides defenses against two of the main attacks regarding Proof of Stake: *long range revisions* and *catastrophic crashes*.

Long range revisions

Introducing the withdrawal delay at the start of the end dynasty of a validator calls for a synchronicity assumption to be made: suppose that a group of validators, that once possessed more than 2/3 of the deposit, withdraws its deposit; these validators can use their once owned supermajority power to finalize conflicting checkpoints without fear of being slashed, because they have already withdrawn their deposit. This is the so called long-range revision attack.

This type of attack is prevented both by the property for which a finalized block can never be reverted and the supposition that each user will receive, at some fixed frequency, a complete up-to-date copy of the chain. Finalized blocks older than a certain amount will simply be ignored because all users will have already seen a finalized block with that specific height.

Catastrophic crashes

Assume that more than a third of validators are not connected to the network due to different reasons, from computer failure to malicious nature. In this scenario, it is not possible to finalize any more checkpoints, because it is impossible to reach at least 2/3 of the consensus. To fix this problem, an *inactivity leak* is instituted: the deposit of any logged-off validator is slowly drained, until the deposit size of the validators that are active on the network is greater than 2/3 of the total. When this quote is reached, the network is able to finalize new checkpoints.

The drained coins are either burned or returned to the validator after a few days.

This solution for catastrophic crashes is not optimal, because it introduces the possibility of finalizing two conflicting checkpoints. Indeed, if a subset V_A of validators votes on chain A and a subset of validators V_B votes on chain B, it is obvious that on chain A V_B's deposits will be drained, and vice versa, so V_A will have the supermajority of power on chain A and V_B will have the supermajority on chain B. This will lead to two finalized conflicting checkpoints without anyone being punished.

Chapter 4

Gasper

Since the beginning, even if Ethereum was born as a Proof of Work based blockchain, its developers' aim was to build a Proof of Stake protocol to rule the network. They conceived the Gasper protocol, a Proof of Stake consensus protocol based on the GHOST fork choice rule and on some of Casper's concepts. The transition to the PoS protocol was made official on September 15th, 2022 with the Merge. This chapter's goal is to present the Gasper protocol and how Casper and GHOST are implemented in it. At first, some basic information are presented; then the main protocol is described. After that, the fork choice rule actually utilized in Gasper is described and the safety and liveness property are demonstrated. Finally, how to incorporate a dynamic set of validators in the Gasper protocol is shown.

Gasper is a Proof of Stake consensus protocol obtained by combining Casper FFG and LMD GHOST. Casper is used as a gadget, a finalization mechanism over an already-existing proposing protocol, to mark certain blocks as finalized, so that even a user with partial information knows for sure if a block belongs to the main chain or not; LMD GHOST is a fork choice rule that is slightly different from GHOST and that is used to solve forks. The aim of the Gasper protocol was to replace the Proof of Work consensus protocol that characterized the execution layer, and to be carried out on the consensus layer.

4.1 Basic background knowledge

The goal is to create a consensus protocol, that is a series of rules and actions, formerly established, that all users of the network must follow. The participants of the network are called *validators*, because their prime task is to validate data in the Ethereum Beacon chain.

Definition 4.1.1 *An honest validator is a participant that follows the protocol, a byzantine validator does not.*

Validators exchange messages, packets of data, written in some chosen language, regarding the network, with one another. With messages, validators can have various purposes:

they can *propose* blocks; they can make *attestations* to a precise block, voting for it; they can *activate* another validator in the network; they can also state another validator is a byzantine, proving its malicious actions and *slashing* it. Regarding this last type of messages, the following property is essential.

Property 4.1.1 *Once a message is sent from an honest validator to another one, it is broadcasted across the entire network. Also, all messages are digitally signed by their sender.*

This property, indeed, states that there is no possibility of a message being sent to only a section of the network, unless a validator is a byzantine. Furthermore, the author of a fraudulent message will always be recognized through the digital signature. This prevents the network from being vulnerable to attacks in which byzantine validators pretend to be honest ones. A *dependency* is a message appended to an already existing one; each message can have none or multiple dependencies.

Definition 4.1.2 *A message is said to be accepted if and only if all its dependencies are accepted.*

The genesis block, the first block of the blockchain, represents a blank state, whereas the following blocks are descriptions of the state transitions. Each user shares the genesis block, $B_{genesis}$, as its initial state, and proceeds to update this state with messages he sees. Due to network latency and to byzantine users, different validators may not always simultaneously see the same state of the network; this justifies the following definitions.

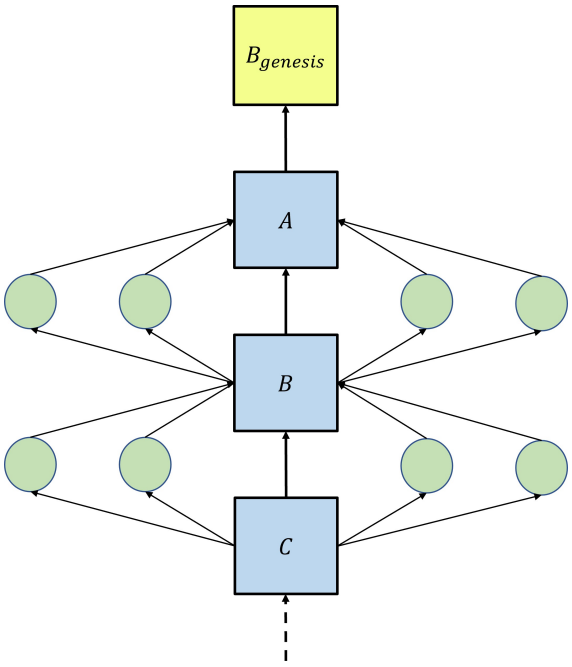


Figure 4.1. View of blocks with respective attestations. The yellow block is the genesis one; blue blocks are accepted blocks; green circles are attestations.

Definition 4.1.3 *The view of validator V, at a fixed time T, is the set of accepted messages the validator sees at that given time. It is referred to as view(V,T) or, to make writing easier, as view(V).*

Definition 4.1.4 *The network view is the set of all the accepted messages on the network, seen by a virtual validator at time T, with no latency. It is symbolized by view(NW,T) or view(NW).*

Given that each block, with the exception of the genesis one, refers to its parent block, $view(V)$ can be seen as an acyclic graph, with root in $B_{genesis}$, in which there is an edge $B \leftarrow B'$ if B is the parent of B'. In this case, we say that B' descends from B an that the latter is an ancestor of the former. Two different blocks, B and B', are conflicting if neither is an ancestor of the other. So, each block B defines a chain, *chain(B)*, which goes from $B_{genesis}$ to B, that is the same in every view that includes B.

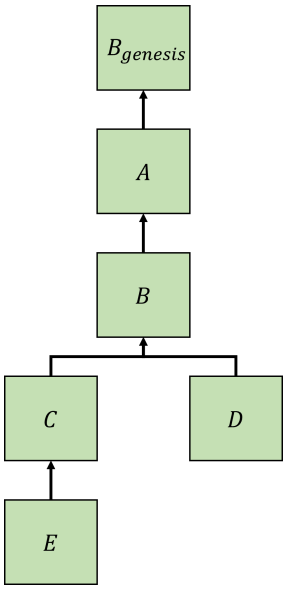


Figure 4.2. Only blocks are represented in this view. They form a tree with root in the genesis block; the arrows represent parent-child edges; blocks C and D are conflicting; E and D are conflicting too; $chain(C) = (B_{genesis}, A, B, C)$.

Time is measured in slots, a constant number of seconds, and in epochs, a fixed amount C of slots. Epoch j of slot i is symbolized by[1]

$$ep(i) = j = \left\lfloor \frac{i}{C} \right\rfloor .$$

[1] $\lfloor \cdot \rfloor$ is the *floor function*; it returns the integer number immediately lower that the input.

Blocks belonging to epoch j have slot numbers $jC + k,\ k \in \{0,1,\ldots,C-1\}$. The genesis block has slot number 0 and is the first block of epoch 0.

As previously mentioned, validators can have different views of the network at the same time, be it because of latency or due to malicious actions made by byzantines. Different systems can have different synchrony conditions:

Definition 4.1.5 *A synchronous system is characterized by fixed upper bounds for time needed to broadcast a message from one node to another.*

Definition 4.1.6 *An asynchronous system does not have set upper bounds for time needed to broadcast a message among nodes.*

Definition 4.1.7 *A partial synchronous system is either:*

1. *a system in which the upper bounds exist but are not prior known*
2. *a system in which the existence of these upper bounds is known only at a certain undefined time T.*

A Proof of Stake protocol is based on the idea that each validator has a voting power proportional to their stake in the network. To create the wanted Proof of Stake protocol, the following hypothesis is made.

Hypothesis 4.1.1 *Assume that the system is composed of N validators, $\mathcal{V} = \{V_1,\ldots,V_N\}$, each with a respective real and positive quantity of stake $w(V_i),\ \forall i = 1,\ldots,N$. Then, the average amount of stake of each validator is equal to 1, so that the total amount is N.*

This hypothesis is not restrictive, because the various function in the protocol in which the stake is utilized will be linear.

Definition 4.1.8 *A blockchain is said to be p-slashable if a validator with the network view can slash a byzantine that holds a total stake of pN.*

Two of the main properties of the defined protocol are *safety* and *liveness*.

Property 4.1.2 (Safety) *A protocol possesses safety if two conflicting blocks can never be finalized, in any view.*

Property 4.1.3 (Plausible liveness) *A protocol has plausible liveness if the set of finalized blocks can always grow, regardless of previous circumstances.*

Property 4.1.4 (Probabilistic liveness) *A protocol has probabilistic liveness if it is probable that the set of finalized blocks may grow, in spite of previous events.*

Since in a Proof of Stake system proposing blocks is free, additional tools that prevent from perverse behaviour are needed.

A fork choice rule is needed to solve forks: it receives in input a view G and gives as

output a leaf B, which will become the *head of the chain* in view G. This is GHOST's role.

The GHOST - Greediest Heaviest-Observed Sub-Tree - rule is a fork choice rule proposed to replace the longest chain one. It states that the head of the chain is the leaf belonging to the heaviest subtree, computed taking into consideration also ommer blocks. The Latest Message Driven Greediest Heaviest-Observed Sub-Tree rule (LMD GHOST) is a slightly different protocol that is "correct-by-construction".

Definition 4.1.9 *Given a view G and a set of the latest attestations M, one per validator, the weight w(G,B,M) is the sum of the stake of validators whose latest attestation is either block B or one of its descendants.*

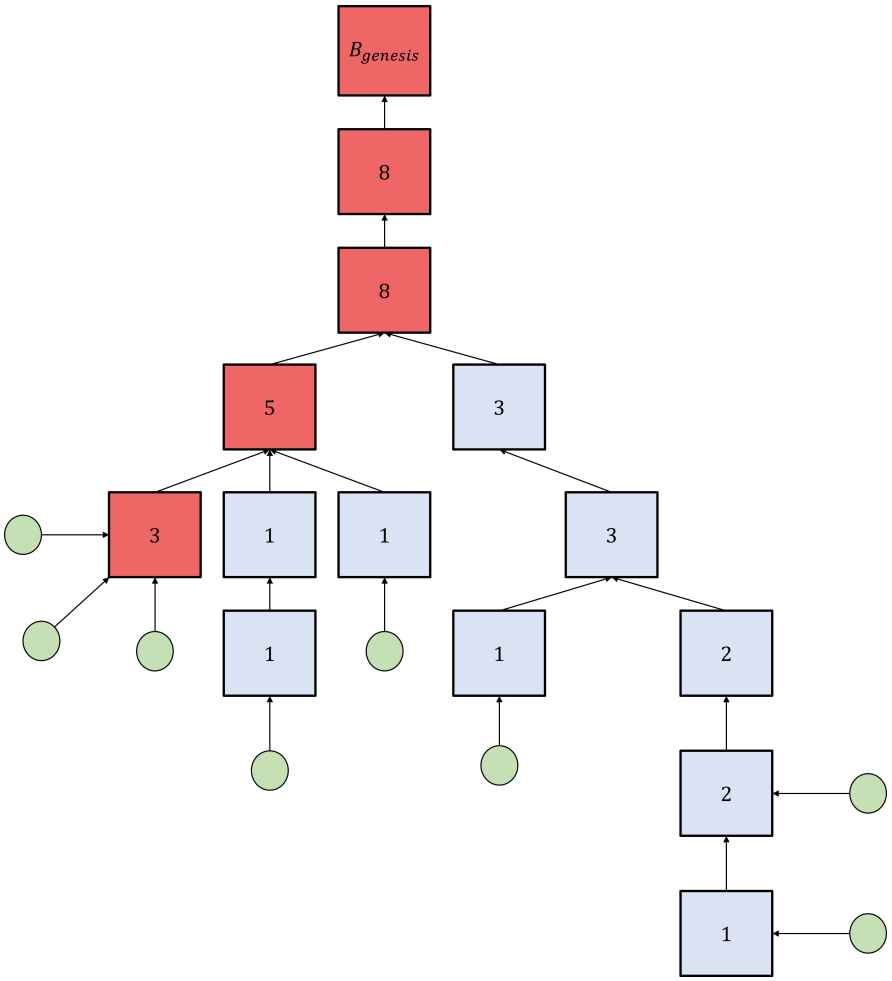


Figure 4.3. Example of LMD GHOST fork choice rule. The number in each block represents the weight of that block, computed as the sum of all attestation to that block, each with unit weight; the red blocks belong to the main chain, according to LMD GHOST.

The **LMD GHOST** rule obeys the following algorithm:

Given a view G in input

1. set B as the genesis block
2. set M as the set of latest attestations of the validators, taking one from each one
3. **while** B is not a leaf in view G **do**
4. update B with $\arg\max_{B' \in Children_T(B)} w(G, B', M)$
5. ties are broken in favour of the block with smaller hash of the block header
6. **return** B

Other tools needed are the concept of finalization and slashing conditions; both are given from Casper FFG: Casper the Friendly Finality Gadget. It is a gadget used on a preexisting block proposal protocol, regardless of it being a Proof of Work or Proof of Stake mechanism. Casper looks at trees made of only checkpoints, blocks whose height is a multiple of a certain fixed number H. Casper defines the concepts of *justification* and *finalization*. Defining with $J(G)$ the set of justified blocks in view G and with $F(G)$ the set of finalized block in view G, $F(G) \subset J(G)$, then:

Definition 4.1.10 *In a view G, a block is justified if it is a descendant of a justified block and if the attestations that go from the justified parent block to the considered one have a weight of at least $\frac{2}{3}$ of the total stake; this last condition can be expressed as the existence of a supermajority link between the parent block and the current one.*

Definition 4.1.11 *In a view G, a block is finalized if it is justified and if there exists a supermajority link from the block to one of its direct descendants. That is:*

$$A \in J(G) \text{ and } \exists\ A \xrightarrow{J} B \text{ such that } \ h(B) = h(A) + 1 \implies A \in F(G).$$

Casper also provides the slashing conditions 3.1 and 3.2, needed to recognize a byzantine and punish it.

4.2 HLMD GHOST fork choice rule

The Gasper protocol actually uses an hybrid version of LMD GHOST. The definition of the *Hybrid Latest Message Driven Greediest Heaviest-Observed Sub-Tree (HLMD GHOST)* algorithm must be preceded by some basic definitions that explain how the Gasper protocol implements Casper's concepts of justification and finalization.
The Gasper protocol applies these concepts not to checkpoints, but to pairs. These pairs are the *epoch boundary pairs*.

Definition 4.2.1 (Epoch boundary pairs) *One or more blocks are chosen from each epoch to be a checkpoint in Casper. Due to the fact that a block can be selected multiple times as a checkpoint, in different epochs, the ordered pair (B, j) is defined as the epoch boundary pair, where B is the chosen block and j is the epoch in which the block is a checkpoint.*

It is now possible to define the notions of justification and finalization on epoch boundary pairs.

Definition 4.2.2 (Justification) *The set of justified pairs J(G), in a fixed view G, is defined as the set of pairs connected to a justified pair by a supermajority link, plus the pair $(B_{genesis},0)$.*

Definition 4.2.3 (Finalization) *A pair (B_0, j) is k-finalized, in a fixed view G, if $(B_0, j) = (B_{genesis},0)$ or if there exists an integer number $k \geq 1$ of blocks, $B_1, B_2, \ldots, B_k \in view(G)$, such that $(B_0, j), (B_1, j+1), \ldots, (B_k, j+k)$ are all justified pairs, adjacent epoch boundary pairs in chain(B_k) and there exists a supermajority link between (B_0, j) and $(B_k, j + k)$.*

Finalization is an extremely strong concept, much more than justification: once a pair (B, j) is finalized in any view, in some epoch j, then in no other view a block conflicting with B can be finalized, unless the network is 1/3-slashable.

For every block B and epoch j, the following concepts can be defined:

Definition 4.2.4 *The j-th epoch boundary block of B, EBB(B,j), is the block whose slot number is higher than that of all the others, but still lower or equal to jC, where C is the number of slots that compose an epoch.*

For example, $EBB(B,0) = B_{genesis}$, because $j = 0$, hence, the slot number of the epoch boundary block must be lower or equal to 0, but the only block that can satisfy this request is the genesis one, because $B_{genesis}$ is the only block in epoch 0 that has slot number equal to 0.

It is also important to notice that each block B, whose slot number is equal to jC, for some epoch j, is an epoch boundary block in all the chains that contain B.

Definition 4.2.5 *The latest epoch boundary block of block B, LEBB(B), is the latest epoch boundary block belonging to chain(B).*

Definition 4.2.6 *Given a block B, view(B) is the set of B and all its ancestors, with all their dependencies; view(B) concentrates on chain(B). The FFG view of B, ffgview(B), is the set of ancestors of LEBB(B), that is view(LEBB(B)), the view of the latest epoch boundary block of B; it is a frozen picture of view(B), taken at the time the last checkpoint is produced.*

To better understand the HLMD GHOST fork choice rule, a prototype version is first given.

This function consisted in starting from the last justified block in a given view G and then in running LMD GHOST.

The **prototype of HLMD GHOST** algorithm is the following:

Let the input be a view G

1. set the pair (B_J, j) as the justified pair with greater attestation epoch j
2. set block $B = B_J$
3. generate M, the set of latest attestations, taking one from each validator
4. **while** $Children_G(B) \neq \emptyset$ **do**
5. update B with $\arg\max_{B' \in Children_G(B)} w(G, B', M)$
6. ties are broken in favour of the block with smaller hash of the block header
7. **return** B

This version was not optimal because it had two problems. The first regards the different concept of two main ingredients of the algorithm: the latest justified pair in a view can be seen as a pair in a *non-changing chain*, fixed at the start of each epoch; meanwhile, the pair (B_J, j) can change many times during the same epoch. To solve this problem, a variant of the pair (B_J, j), that does not change during an epoch, is used.
The other problem arises with forks, because the last justified pairs in the view of the two forking blocks can differ. To solve this problem, the algorithm generates an auxiliary view G' from G that does not carry the tricky blocks. To create this auxiliary view, first, time is frozen at the start of each epoch to define the pair (B_J, j). This means defining the pair considering the FFG view of a block leaf B_l, rather than the entire view. Second, the algorithm shuns the branches that contain a block leaf B_l such that its last justified pair is not caught up with (B_J, j), yet.

The **HLMD GHOST** algorithm is the following:

Given a view G in input

1. generate the set $\mathcal{L}$ of leaf blocks B_l in G
2. set the pair (B_J, j) as the justified pair with greater attestation epoch j in $J(\text{ffgview}(B_l))$, with $B_l \in \mathcal{L}$
3. generate the set $\mathcal{L}'$ of leaf blocks $B_l \in G$ such that the pair $(B_j, j) \in J(\text{ffgview}(B_l))$
4. create the view G' as $\bigcup_{B_l \in \mathcal{L}'} chain(B_l)$
5. set $B = B_J$
6. generate M as the set of latest attestations, taking one from each validator

7. **while** $Children_{G'}(B) \neq \emptyset$ **do**
8. update B with $\arg\max_{B' \in Children_{G'}(B)} w(G', B', M)$
9. ties are broken in favour of the block with smaller hash of the block header
10. **return** B

4.3 Main protocol

Finally, the Gasper protocol can be described. It consists of two phases.
Firstly, validators are split up into *committees* in each epoch, one for each slot that composes that epoch. This is done to distribute the various responsibilities they must take. The committees are generated by a random permutation of fixed length, ρ_j, that is a specific function that can be used only in epoch j. This permutation is assumed to be provided by an external oracle. ρ_j's role is to pseudo-randomly split validators, in each epoch, into C committees $S_0, S_1, \ldots, S_{C-1}$ of equal size, assuming that the total number of validators can be divided by C. If N is the total number of validators,

$$S_k \text{ is the set of } \frac{N}{C} \text{ validators } \mathcal{V}_{\rho_j(s)}, \text{ where } s \equiv k \mod C.$$

These sets form, in each epoch, a partition of the set of validators.

Secondly, validators must perform the task they have been assigned to. In a committee, validators have two roles: proposing a block and attesting to one.

1. One validator per slot is chosen to propose a block by adding a corresponding proposing message to its own view and then broadcasting it to the entire network.
   In more specific terms:
   at the beginning of each slot, with slot number $i = jC + k$, permutation ρ_j selects a validator $V_{\rho_j(k)}$ as the *proposer* for that slot. This validator becomes the first member of the committee S_k of epoch j. $V_{\rho_j(k)}$ calculates the canonical head of the chain, i.e. block B', through the HLMD GHOST function, using as input its own $view(V_{\rho_j(k)}, i)$. Then, $V_{\rho_j(k)}$ proposes block B sending a proposal message, including:
   - the slot number $i = slot(B)$;
   - an indicator to the parent block B' of the proposed block B, $P(B) = B'$;
   - a set, *newattests(B)*, of pointers to all the attestations $V_{\rho_j(k)}$ has already accepted that are not included in any set of attestations of ancestors of B yet;
   - the transactions the validator wants to add to the main chain.
2. The other validators of the committee have to attest to the block they see as the head of the chain; they have to add a message, an attestation, to their views and then send them to the whole system.
   In detail:

at slot $i + \frac{1}{2} = jC + k + \frac{1}{2}$ the other validators of committee S_k compute a block B', the block they see as the canonical head of the chain, with function HLMD GHOST, using as input their view at time $i + \frac{1}{2}$. Then, each validator V sends an attestation α, a message containing:

- $slot(\alpha)$, that is the slot during which the validator is making attestation α;
- the block α attests to, $B' = block(\alpha)$; note that $slot(block(\alpha)) \leq slot(\alpha)$;
- a checkpoint edge between the last justified pair and the last epoch boundary pair of α, that is $LJ(\alpha) \xrightarrow{V} LE(\alpha)$, considered a "FFG vote".

Definition 4.3.1 *The last justified pair of an attestation α, $LJ(\alpha)$, is the justified pair that has highest attestation epoch j, that is, the last justified pair in ffgview(block(α))= view(LEBB(block(α))).*
The last epoch boundary of an attestation α is LE(α)=(LEBB(block(α)),ep(slot(α))).

The slashing conditions 3.1 and 3.2, given by Casper, can be formalized with a notation more suited to the contest they are applied to. The conditions become:

I. *No validator can make two different attestations α_1 and α_2, in the span of the same epoch.*

II. *No validator can share two different attestations α_1 and α_2*, such that

$$aep(LJ(\alpha_1)) \ < \ aep(LJ(\alpha_2)) \ < \ aep(LE(\alpha_2)) \ < \ aep(LE(\alpha_1)).$$

Validators either receive a reward for acting good or a penalty for misbehaving. Rewards can be due to the inclusion, in a proposed block, of valid attestations, *proposer reward*; for attesting to the block that will be justified and finalized, *attester reward*; for proving the faultiness of a byzantine. Penalties can results from laziness, that is if a validator is often offline, or from being caught acting badly.
The protocol is based on the assumptions that honest validators, those who follow the protocol, will never erroneously violate a slashing condition and will always slash a caught dishonest validator.

4.4 Proving safety and liveness

It can be proved that the described protocol grants safety, plausible liveness and probabilistic liveness.

4.4.1 Safety

The property of safety follows from the following theorem:

Theorem 4.4.1 *A given view G is $\frac{1}{3}$-slashable if some finalized pair changes its status, becoming not finalized, when G is updated, or if a finalized pair (B, j) is composed of a block that does not belong to the canonical chain of the view.*

To prove this theorem, first the following lemma is needed.

Lemma 4.4.1 *Given, in a view G, a justified pair (B_J, j) and a finalized pair (B_F, f), with $f < j$, then either $B_F \in chain(B_J)$ or the network is $\frac{1}{3}$-slashable. That is, there exist two subsets of validators, $\mathcal{V}_1$ and $\mathcal{V}_2$, where validators belonging to both transgress one of the two slashing conditions.*

Now, it is possible to demonstrate Theorem 4.4.1.

Proof. A network is not 1/3-slashable if a finalized pair remains so, even when the view is updated. This follows directly from the definitions of the sets of justified and finalized pairs. From the HLMD GHOST algorithm, it is evident that the justified pair with highest attestation epoch always belongs to the canonical chain. From Lemma 4.4.1, it follows that also the finalized pair with highest epoch always belongs to the canonical chain. So, to prove the second condition of Theorem 4.4.1, it is sufficient to demonstrate that two finalized blocks are never conflicting, thus they all belong to the same chain. Assume that two finalized pairs (B_1, f_1) and (B_2, f_2), $f_2 > f_1$, are composed of conflicting blocks; then, the system must necessarily be 1/3-slashable. Since (B_2, f_2) is a finalized pair, it is also a justified pair; applying Lemma 4.4.1, this means that B_1 must be an ancestor of block B_2, but this is impossible because of the conflicting nature of the blocks. So, the lemma attests that the view is 1/3-slashable.

■

4.4.2 Plausible liveness

Theorem 4.4.2 *Assuming a total of N validators, if at least $2N/3$ of stake is held by honest validators, than it is always possible for new blocks to be finalized, in spite of previous events.*

To prove this theorem, first the definition of a *stable chain* is needed.

Definition 4.4.1 *Given a block B in epoch j, $chain(B)$ is said to be stable at epoch j if $LJ(EBB(B, j))=(B', j-1)$, for some block B'.*

Now, it is possible to prove the theorem.

Proof. Starting at epoch j, the first slot number is $i = jC$. Assuming that the network possesses adequate synchronicity, it is reasonable to assume that all validators have the same view of the network. In slot i, a possible honest proposer proposes block B as the child of the block obtained with the HLMD GHOST function. Introducing the proposed block in the adjourned view G and defining $c = chain(B)$, it is likely that c is made to be stable at epoch $j+1$, by the next block added to the chain; indeed, due to the hypothesis of having at least two third of honest validators, they attest either for block B or one of its descendants. This creates the supermajority link $LJ(B) \xrightarrow{J} (B, j)$, that justifies B. In the next epoch, $j+1$, it is plausible that the new proposed block, B', with slot number $(j+1)C$, contains all these attestations, because of the good synchronicity

assumption. This means that $LJ(B') = (B, j)$ and $chain(B')$ is stable at the current epoch. Hence, it is possible for the chain of the first epoch boundary block of the new epoch to be stable. But this means that it was possible to consider the starting chain c stable from the beginning. So, going back to epoch j, there existed a supermajority link $(B', j-1) \xrightarrow{J} (B, j)$ in ffgview(B). Thinking recursively, for the next epoch, it is not wrong to think that the next epoch boundary block B'', with slot number $(j+1)C$, is also stable, with supermajority link $(B, j) \xrightarrow{J} (B'', j+1)$, that finalizes the pair (B, j).

■

4.4.3 Probabilistic liveness

It is best to treat probabilistic and plausible liveness separately because, even if the former implies the latter, probabilistic liveness needs some strong assumption to be valid, while plausible liveness does not. So it is best to separate the two to emphasize that even if the hypothesis for probabilistic liveness do not hold, plausible liveness is still valid. Indeed, in order to be valid, probabilistic liveness needs fragile probabilistic assumptions, such as good network synchrony and an equal stake for all validators. These assumptions can be easily not satisfied.

The proof of probabilistic liveness is divided into three steps, forming a sequence, so that each one would remain valid even if some assumptions were changed.

1. The first step aims to demonstrate that, under favorable assumptions, honest validators would likely find a block with a high number of attestations after the first slot, that is, with a high weight.

2. The second step shows that if in the first step, after the first slot, a block with high weight is found, then this block's weight keeps growing in the following slots of the epoch. Thus, the block or one of its descendants is probably going to be justified.

3. The third and last step proves that with a high probability of a block being justified in every epoch, as shown in the previous step, the probability of at least a block being finalized grows with the number of epochs.

4.5 Dynamic set of validators

It must be noted that assuming a static set of validators is a restrictive hypothesis, because validators may want to exit or enter the set in a subsequent time. So, a *dynamic set of validators* must be introduced. This introduction influences the safety property of the network, reducing the number of validators that can be punished for violating a slashing condition: a byzantine could act maliciously and immediately exit the set of validators, without being slashed.

Considering a dynamic set of validators, it is possible to define two distinct subsets:

Definition 4.5.1 *Given two sets of validators $\mathcal{V}_1$ at time t_1 and $\mathcal{V}_2$ at time t_2, with $t_1 < t_2$, then the set $A(\mathcal{V}_1, \mathcal{V}_2)$ is the set of validators who activate at time t_2, that is they belong to $\mathcal{V}_2$ but not to $\mathcal{V}_1$; similarly $E(\mathcal{V}_1, \mathcal{V}_2)$ is the set of validators that deactivate after time t_1, that is they belong to $\mathcal{V}_1$ but not to $\mathcal{V}_2$.*

The safety property is still based on a lemma, similar to Lemma 4.4.1:

Lemma 4.5.1 *Assuming having a dynamic set of validators, if there are two conflicting finalized pairs (B_1, f_1) and (B_2, f_2), in a view G, then the network must be 1/3-slashable.*

This means that there must exist two justified pairs (B_L, j_L) and (B_R, j_R) in view G and two sets of validators $\mathcal{V}_1 \in \mathcal{V}(B_L)$ and $\mathcal{V}_2 \in \mathcal{V}(B_R)$ with at least 2/3 of the stake of the corresponding attestation epoch, where validators belonging to both violate one of the two slashing conditions.

The following theorem gives a lower bound for the size of slashable validators of two conflicting blocks.

Theorem 4.5.1 *Consider a parent block $B_0 \in G$ with validator set $\mathcal{V}_0$ and two of its children, the conflicting finalized blocks $B_L, B_R \in G$, with respective set of validators $\mathcal{V}_L$ and $\mathcal{V}_R$. Defining a_i as the weight of the set of validators that are active in $\mathcal{V}_i$ but not in $\mathcal{V}_0$, that is $a_i = w\Big(A(\mathcal{V}_0, \mathcal{V}_i)\Big)$, for $i = L, R$, and e_i as the weight of the set of validators that are active in $\mathcal{V}_0$ but not in $\mathcal{V}_i$, that is $e_i = w\Big(E(\mathcal{V}_0, \mathcal{V}_i)\Big)$, for $i = L, R$.*
The size of slashable validators must be at least equal to

$$\max\Big(w(\mathcal{V}_L) - a_L - e_R, w(\mathcal{V}_R) - a_R - e_L\Big) - w(\mathcal{V}_L)/3 - w(\mathcal{V}_R)/3.$$

Note that in the case where there is a static set of validators, $a_i = e_i = 0$ for $i = L, R$ and we obtain the usual limit $w(\mathcal{V}_L)/3 = N/3$.

The strength of the constraint depends on the rules that dictate the activation and the exit of the validators. Fixing T_0 as the time at which block B_0 was published, and T_n as the time at which B_L and B_R were published, then a_i and e_i can be controlled by the difference between these two times.

If in every epoch new validators can be activated/deactivated up until a constant limit of stake, then

$$a_i \leq k_1(T_n - T_0) \quad \text{and} \quad e_i \leq k_2(T_n - T_0), \quad \text{for} \quad i = R, L.$$

If in every epoch new validators can be activated/deactivated proportional to the validator stake, then

$$a_i \leq w(\mathcal{V})(1 + k_1)^{(T_n - T_0)} \quad \text{and} \quad e_i \leq w(\mathcal{V})(1 + k_2)^{(T_n - T_0)}, \quad \text{for} \quad i = R, L.$$

Obviously, if the difference between the two times becomes too big, the bound is not useful, like in the case of a set of validators completely different from the one where there was the byzantine.

Chapter 5

Practical Features

The practical implementation of the consensus layer differs from the theoretical explanation of the protocol. This last chapter aims at describing technicalities of Ethereum's network and at presenting changes that its developers needed to make to avoid problems.

At the moment, November 11th, 2022 at 10:50 pm, the market price for ether is of 1265.46 dollars, with a market capitalization of 154.8 billion dollars. Ether's total offer is of 122.4 million and it is the second cryptocurrency in the market by capitalization. Ethereum's current *Annual Percentage Rate (APR)* is 4.4%; it represents the annually obtained interest from the initial investment. Ether's price decreased by 4.82% in the last 24 hours - values taken on November 11th, 2022 at 10:50 pm.

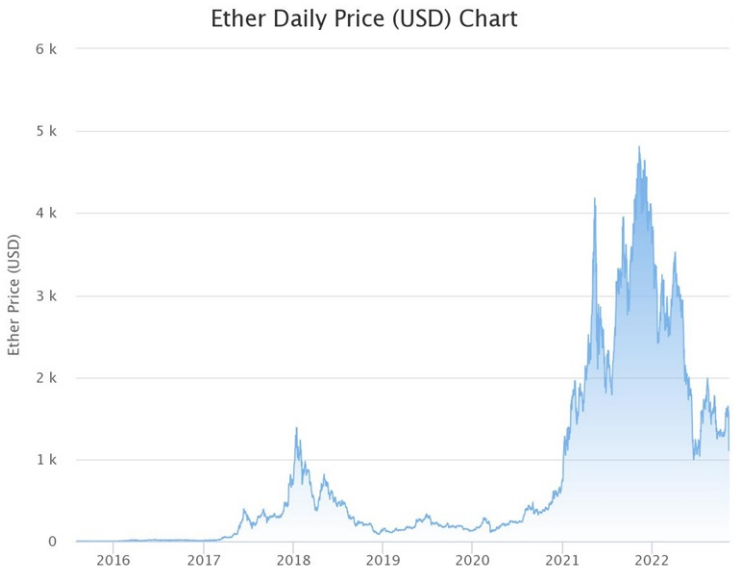


Figure 5.1. Ether's price from August 2015 to November 2022. Graph taken from Eherscan.

Ethereum's Beacon Chain, after the Merge, counts a total of more than 464 thousands validators with only almost 20 thousands nodes.

Figure 5.2. Number of validators on the Beacon Chain from December 1st, 2020 to November 11th, 2022. Chart taken from beaconcha.in

With the transition to PoS, Ethereum said goodbye to uncle blocks. This is shown by the following graph representing the trend of daily ommer blocks produced from December 1st, 2020 to November 11th, 2022.

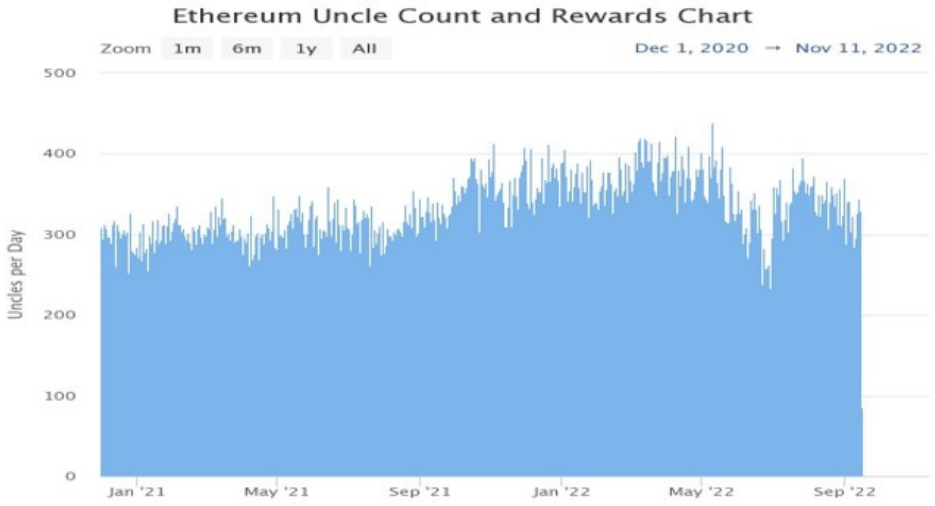


Figure 5.3. Chart taken from Etherscan.

5.1 How to become a validator

There exist different ways users could become validators, based on how much they are willing to stake. It is already been said that to activate a validator software 32 ETH are needed. User can stake the whole sum or a portion of it and still take part to the PoS protocol.

A user can stake the whole sum of 32 ETH and activate a validator individually, in the so-called *Solo home staking*. Besides the ETH stake, an hardware connected to the internet is needed; it must run both an execution client and a consensus client. This kind of staking ensures the whole reward going to the user, improves decentralization of the network and does not require any trust in a third party. There are risks that results from solo home staking, such as the fact that the whole sum of 32 ETH is at stake, the necessity of being almost always online in order to not being punished for being offline and the possibility of having a large amount of ETH slashed due to malicious behaviour. Particular abilities are not necessary because there exist some tools that simplify the process.

A user can stake the whole 32 ETH sum, but delegate an external provider to run the hardware, in the *Staking as a service* process. The user gets the whole reward, minus a monthly provider fee; the risks are the same as solo staking, with the addition of having to trust a third entity. There are only two requirements: staking the 32 ETH and safely storing the keys needed to access the provider.

If a user does not have or does not want to stake the entirety of 32 ETH, there are solutions that allow the user not to stake the whole quota, but only a part, and to receive rewards proportional to this quota. These are the *staking pools*. The biggest staking pools of ETH are *Lido, Coinbase, Kraken* and *Binance*.

- Lido is the biggest staking pool and owns almost 130 thousands validators, 27.79% of the total number of validators. Joining Lido, users receive a token that represents their ether on the Beacon Chain. A *token* is an asset that lives on its own blockchain, which can be used to make economic exchanges and investments. The system applies a 10% fee, split between those who control the node and Lido DAO, which manages the system's funds. Independent validators, those who stake the entirety of 32 ETH and run the validator software alone, are almost 110 thousands and make up 23.44% of the total number.

- The second biggest staking pool is Coinbase, with about 60 thousands validators, 12.73% of the total validators. Coinbase works as a standard staking pool, taking the stakes of different users and rewarding them with a proportional share. Coinbase does not establish a minimum amount of ETH that one can stake, but there is a fixed maximum amount, that changes over time, to not clog the network.

- Kraken staking pool is the third bigger one and counts little more than 35 thousands validators, 7.72% of the total. Kraken weekly rewards users with a variable rate, while the annual reward rate is fixed between 4% and 7%.

- Binance is the fourth largest staking pool and has circa 27 thousand validators, for a percentage of 5.83% of the total validators. After staking ETH on Binance, the user receives a BETH token. An user will be able to obtain its ETH only after 6-12 months after the Merge.

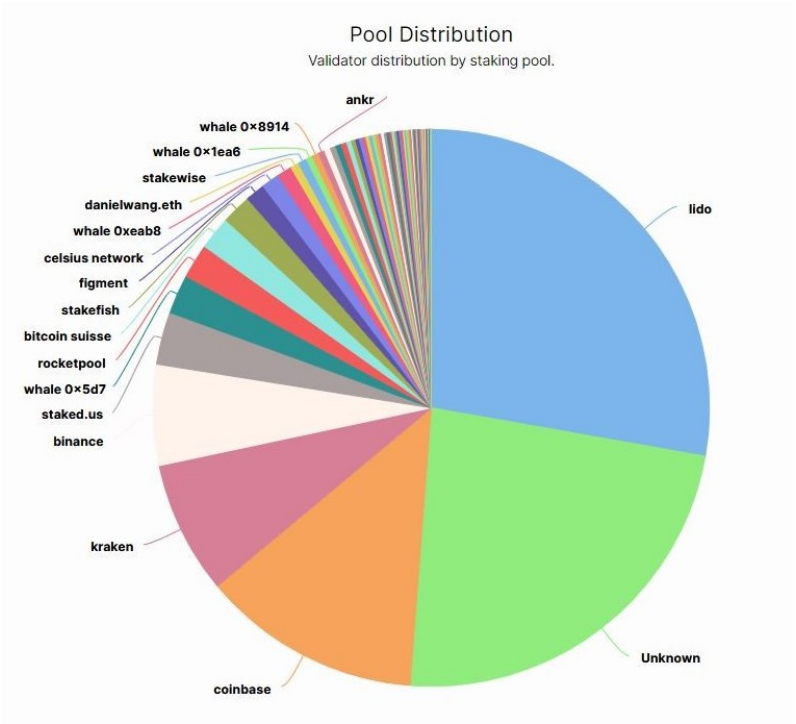


Figure 5.4. Staking pools distribution. Chart taken from beaconcha.in

5.2 Additional keys for validators

With the passage to a Proof of Stake consensus protocol, Ethereum needed another type of keys for users to stake ETH and run validators. This was made to improve scalability, in order to make the network even faster in reaching consensus. In fact, these new keys use the Boneh-Lynn-Schacham digital signature scheme, which allows to generate $m-\text{of}-n$ threshold signatures. Therefore, there is no longer the necessity of reaching an absolute totality of agreement.
The BLS scheme is based on a pairing friendly curve. However, the usual keys are generated on the $Secp256k1$ elliptic curve, defined on the field $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. This curve is not suitable for pairings so, to use the new keys, one could think it would be necessary to change the curve. This would cause many problems with the 'old' keys, problems that can be solved using a Smart Contract. There is no need to change the curve, but the network asks a specific Smart Contract to verify these

signatures built on the pairing friendly curve.
Before the Merge, users only needed a key to get access to their wallets; after the Merge, users that wanted to participate in the protocol activating a validator software needed two additional keys: the *validator key* and the *withdrawal key*. The former consists in a *validator private key* used to sign proposal messages and attestations, and a *validator public key* used to stake ETH to activate a validator software. This public key is also used by the network to identify the validator itself. The withdrawal key is not used yet; it will be used when the Shanghai upgrade hits, to move the validator funds. It also consists in a private and a public key. If this key is lost, the validator looses access to its balance.
Considering two keys for every 32 ETH staked, the problem of managing the keys goes out of control, especially if a user runs multiple validators. To avoid this problem, multiple validator keys are made to be derived from a single seed, known to many. Different keys are identified with different paths, so there is no ambiguity.

5.3 Differences between Gasper and implementations

While describing the Gasper protocol, the blockchain's network was modeled as a graph and validators were thinking entities with infinite computational power. In the practical implementation, validators do not see the network as a graph, but each one runs a software to analyze the structure and return information. This software saves the *'store'*, which is a representation of the view, and updates it whenever new blocks or attestations are received.

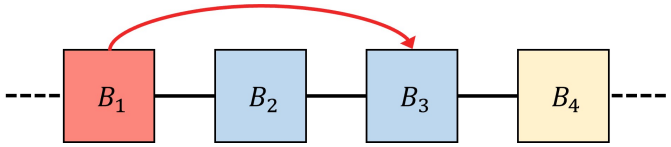
In the Gasper protocol, the validator chosen to propose a block must include all the attestations in its view; furthermore, a validator that attests to a block must run the HLMD GHOST fork choice rule using as input the view containing all the attestations. Developers included some restriction to prevent some type of attacks.

In practice, a validator that proposes a block does so including all attestation that were made at least a fixed number of slots before. This number is the *attestation inclusion delay*. This needs to be done to prevent centralization of the network: for example, if the duration of a slot is slim, well-connected nodes of the network might be able to see and publish attestations way faster than common nodes, centralizing the system. The developers decided to calibrate this attestation inclusion delay on information coming from the real world: a small delay improves the transaction processing time, raising network's scalability, a large delay enhances the decentralization of the system.
Developers decided to put limits to the attestations that a validator can consider while running he HLMD GHOST function. Indeed, a validator that has to attest to a block belonging to slot N must consider only attestation made up until the previous slot, which is slot $N-1$. This is done to prevent byzantine taking control of the chain. Indeed, the Gasper protocol establishes that validators can attest to a block of slot N only after half of the slot has already passed. This means that, if a slot is composed of s seconds, validators that have to attest to a block belonging to slot N can start doing so only at

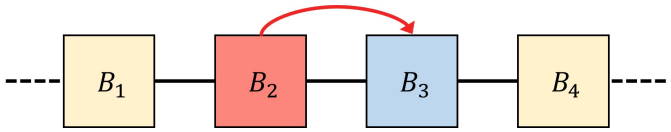
the time $s(N + 1/2)$. If a byzantine does not wait this time and starts making attestations before everyone else, he has more chances of gaining control of the network. This could be done because attesting before the right time is not a slashable action, and also malicious validators could easily fake timestamps to make the others believe they signed their attestation at the rightful time.

Another difference from the theoretical Gasper protocol is in the definition of a finalized pair. Definition 4.2.3 defines a finalized pair considering k subsequent justified pairs. The practical implementation is made to consider only the last 4 epochs. For example, if B_1, B_2, B_3 and B_4 are the last four epoch boundary blocs of four consecutive epochs, with B_4 being the last, then:

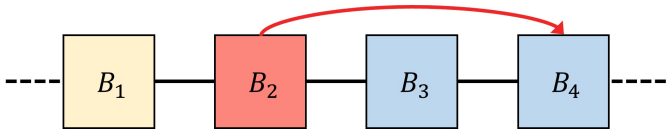
1. if B_1, B_2, B_3 are justified blocks and the attestations that justify B_3 have as the last justified block B_1, then B_1 is finalized



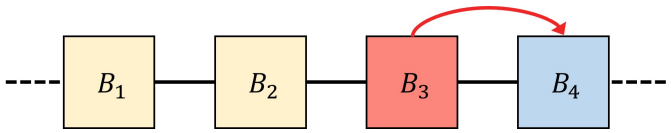
2. if B_2, B_3 are justified blocks and the attestations that justify B_3 have as the last justified block B_2, then B_2 is finalized



3. if B_2, B_3, B_4 are justified blocks and the attestations that justify B_4 have as the last justified block B_2, then B_2 is finalized



4. if B_3, B_4 are justified blocks and the attestations that justify B_4 have as the last justified block B_3, then B_3 is finalized



5.4 Weak subjectivity

Consensus protocols can be categorized into two main groups, based on the degree of social information needed.

Definition 5.4.1 (Objective) *A consensus protocol is objective if a node that joins the protocol, knowing only the protocol, the set of all blocks and other important published information about it, can reach the same conclusion on the network current state as every other node already belonging to the system.*

Definition 5.4.2 (Subjective) *A consensus protocol is said to be subjective if different nodes reach different conclusions on the network current state, when joining the system.*

Proof of Work is an objective protocol: the main chain is unique, so the network current state is obvious to every node, even to those who join the network later. There cannot be mistakes, because the longest chain is always recognizable.
Blockchains that use a social system as consensus set are subjective. A new node, when joining the network, must trust another node to give reliable information about the network current state. Different nodes can give different information, so the main chain is not unique and there can be various valid chains.

Proof of Stake seems to be subjective, but that is not quite the case. Vitalik Buterin introduced the concept of *weak subjectivity* to represent a requirement for PoS based blockchains.

Definition 5.4.3 (Weakly subjective) *A protocol is weakly subjective if a node, entering the system knowing only the protocol, the set of all blocks, other important published information about it and a provably valid state of the network, lacking at most the latest N nodes, reaches the same conclusion about the network current state as all the other nodes.*

This model perfectly describes the Proof of Stake: the protocol prevents nodes from modifying a block that is more than N positions behind. Thus, if a state is proved valid and becomes an ancestor of more than N valid states, then a subsequent state that is not a descendant of the first one can be valid.

This process is carried out on Ethereum by defining *weak subjectivity checkpoints*: blocks that provably belong to the main chain. These serve as *revert limits*, because blocks older than a weak subjectivity checkpoint cannot be altered. This property weakens long-range attack, by simply rejecting long-range forks. Assuring a smaller interval between two consecutive weak subjectivity checkpoints than the withdrawal period of a validator assures that a byzantine cannot act maliciously forking the chain and then leave without being slashed.

The difference between weak subjectivity checkpoints and finalized blocks stands in the way those are treated by the protocol. If a node receives two conflicting finalized blocks, there is no way to choose the one certainly belonging to the canonical chain, because both

are valid; therefore, consensus is not reached. Contrariwise, if a node receives a block conflicting with a weak subjectivity checkpoint, the block is immediately rejected, because the checkpoint represents the absolute truth of the state of the network that cannot be changed.

The probability of establishing an incorrect weak subjectivity checkpoint is extremely low, because their validity can be verified through comparison with an external source, like a block explorer or other nodes.

Weakly subjective protocols work quite well because:

- nodes that are always online are not affected by weak subjectivity, but for them it is equivalent to objectivity: they will always recognize the network current state and will not need external information about it;
- nodes that are offline, but log-in in intervals of N nodes will not be affected because they will always be able to obtain an updated version of the network current state;
- nodes that join the network at a later time or that are offline for a long period rely heavily on weak subjectivity: they will have to retrieve from a node, an external provider or a block explorer, the hash of a recent block and insert it into their view as a weak subjectivity checkpoint.

Conclusion

Blockchain is a new technology that could be useful in many fields, indeed it is gaining more and more importance in the society.
In 2009, the first blockchain, Bitcoin, was developed by an anonymous inventor, Satoshi Nakamoto. Bitcoin is a shared, permissionless blockchain, based on the homonymous cryptocurrency, whose structure and integrity are based on a Proof of Work (PoW) protocol. Proof of Work implies that miners have to prove they have done a certain amount of work by hashing a chosen variation of a given value and finding a quantity that must be lower than a fixed target. Proof of Work leads to a certain centralization of the network, due to the ever-increasing difficulty of the task: users group into mining pools in order to combine all their computational power in a single miner. Also, because of the great amount of computational power needed to solve it, the Proof of Work has a great negative environmental impact. Because of these problems, another type of protocol was created: Proof of Stake. This protocol does not consist in miners solving a very difficult task, but its actuators must stake a portion of their cryptocurrency to activate a validator software. These validators both propose and attest to blocks to choose which one might join the canonical chain of the blockchain. This is the protocol that Ethereum adopted on September 15th, 2022. Proof of Stake is based on the idea that all those who take part in the protocol have a decisional power proportional to the amount of ether staked. The PoS protocol used by Ethereum is Gasper, obtained combining Casper and GHOST. Casper is a finalization gadget used to mark certain blocks as finalized, so that even a user with only partial information can be sure of the validity of the chain previous to that specific block. GHOST is based on the idea that ommer blocks, valid blocks that share a parent with a block belonging to the main chain, contribute to the heaviness of a branch. The chosen chain in a fork will no longer be the longest one, as in Bitcoin, but the heaviest one. The Gasper protocol combines these two elements in a complete Proof of Stake mechanism. Validators are split into committees in each slot of each epoch. In each slot a validator is randomly chosen to propose a block, while the others must attest to it. If the block receives at least two thirds of attestations it is finalized and added to the chain.

For a very long time it has been discussed whether Proof of Stake is really better than Proof of Work. The objection that has been made, among others, is that validators must stake their money and lock it up for a certain amount of time. It is true that freezing an amount of capital could be inefficient for users, but keeping it locked will increase the cryptocurrency value in the market. This is due to the fact that a cryptocurrency value

increases when its availability in the market decreases. Also, PoS can establish a level of security way higher than the one assured by PoW by slashing byzantine validators, while security on a Proof of Work protocol is only based on the reward system.
In conclusion, Proof of Stake algorithms can be built to be extremely safe, with weak subjectivity being a necessary and sufficient condition to solve the nothing-at-stake problem; also, it is plausible to believe that Proof of Stake protocols are way more economically efficient that Proof of Work ones.

Ethereum's future is based on the introduction of the shard chains: secondary chains that will be introduced, probably in 2023, to verify specific transactions alongside the Beacon Chain. Their main goal is to make transaction processing time faster, thus increase the scalability of the network.

Bibliography

[1] Danilo Bazzanella and Andrea Gangemi. *Lecture notes on Blockchain and Cryptoeconomy*. Politecnico di Torino, Slides, 2021.

[2] Evangelos Benos, Rodney Garratt, and Pedro Gurrola-Perez. "The Economics of Distributed Ledger Technology for Securities Settlement". In: *Bank of England Working Paper No. 670* (2017).

[3] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2014.

[4] Vitalik Buterin. *Proof of Stake: How I Learned to Love Weak Subjectivity*. https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity. 2014.

[5] Vitalik Buterin and Virgil Griffith. *Casper the Friendly Finality Gadget*. 2019.

[6] Vitalik Buterin et al. *Combining GHOST and Casper*. 2020. DOI: 10.48550/ARXIV.2003.03052. URL: https://arxiv.org/abs/2003.03052.

[7] *Ethereum*. https://ethereum.org.

[8] *Ethereum 2 Matters - A Journey Through Time*. 2019. URL: https://www.bitcoinsuisse.com/news/ethereum-2-0-matters-a-journey-through-time.

[9] *Fork Choice Rule*. https://eth2.incessant.ink/book/03__eth1/06__fork-choice-rule.html.

[10] Diego Geroni. *Hybrid Blockchain: The Best Of Both Worlds*. 101 Blockchains. 2021. URL: https://101blockchains.com/hybrid-blockchain/.

[11] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems, vol. 4, n. 3* (1982), pp. 382–401.

[13] Wahome Macharia. *Cryptographic Hash Functions*. ResearchGate. 2021. URL: https://www.researchgate.net/publication/351837904_Cryptographic_Hash_Functions.

[14] Vitor Mesk. *Weak Subjectivity*. https://academy.binance.com/en/glossary/weak-subjectivity.

[15] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426.

[16] Everett Muzzy. *What is Proof of Stake?* ConsenSys Blog. 2020. URL: https://consensys.net/blog/blockchain-explained/what-is-proof-of-stake/.

[17] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[18] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22 (1979), pp. 612–613. URL: https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf.

[19] Yonatan Sompolinsky and Aviv Zohar. "Secure High-Rate Transaction Processing in Bitcoin". In: *Financial Cryptography. Lecture Notes in Computer Science* 8975 (2015), pp. 507–527.

[20] Alexandra Tran. *An introduction to byzantine fault tolerance and alternative consensus*. 2017. URL: https://ancapalex.medium.com/a-cursory-introduction-to-byzantine-fault-tolerance-and-alternative-consensus-1155a5594f18.

[21] Sue Troy and Mary K. Pratt. "Distributed ledger technology (DLT)". In: *TechTarget* (2021). URL: https://www.techtarget.com/searchcio/definition/distributed-ledger.

[22] Jianing Wu. *Ethereum's History: From Zero to 2.0*. 2021. URL: https://www.etftrends.com/model-portfolio-channel/ethereums-history-from-zero-to-2-0/.

[23] Dylan Yaga et al. "Blockchain Technology overview". In: *NISTIR 8202* (2018).