Università degli Studi di Genova

Doctor of Philosophy Thesis in

COMPUTER SCIENCE AND SYSTEMS ENGINEERING

A new consensus mechanism for permissionless Blockchains

Luca Andreoli

March 2026

Supervisor: Prof. Massimo Maresca

Summary

1. Introduction

1.1 Distributed systems and consensus in untrusted environments

The topic of **distributed systems** (1) **and consensus in untrusted environments** constitutes one of the longest-standing research lines in computer science and distributed systems. To address it correctly, it is necessary to clarify what "consensus" means and what makes an environment "untrusted."

In the context of distributed systems, consensus is a formal property concerning the ability of a set of independent nodes to produce a common decision despite the absence of a global view of the system. Each node observes only part of the overall state, receives messages with non-deterministic delays, and cannot reliably distinguish whether another node is slow, faulty, or malicious. A consensus protocol must guarantee that all correct nodes converge on the same value, that the decided value is consistent with the system inputs, and that a decision is reached. These properties describe the objective that a consensus protocol must satisfy. It should be emphasized that consensus is a global property of the system, not a local behavior of a single node.

An environment is defined as "untrusted" when it is not possible to assume that participants follow the protocol or act in good faith. In such a context, a node may arbitrarily deviate from the expected behavior, send contradictory messages to different recipients, collude with other nodes, or attempt to prevent system progress. In an untrusted environment, the system must remain secure even in the worst case, assuming strategic adversaries.

The impossibility of distinguishing between network delays and malicious behavior, combined with the possibility that nodes act arbitrarily, leads to impossibility results such as the FLP theorem (2), which shows that deterministic consensus cannot be guaranteed in a completely asynchronous system with even a single faulty node. The FLP theorem states that **no deterministic algorithm** can guarantee consensus in a **fully asynchronous distributed system if even a single node may fail** (even by crashing).

Any practically usable consensus protocol must explicitly state which assumptions it introduces to circumvent these limits. Some protocols assume partial synchrony, others introduce randomization, and others still limit the adversary's power by imposing economic costs or scarce resources. In all cases, the problem of consensus in untrusted environments does not admit a universal solution.

1.2 The Byzantine Generals Problem as a theoretical model

The **Byzantine Generals Problem** is a fundamental theoretical model in the discipline of distributed systems that formalizes the difficulty of achieving **reliable consensus** in the presence of nodes that may behave arbitrarily and maliciously. It was introduced by Leslie Lamport, Robert Shostak, and Marshall Pease in their famous 1982 paper (3), in which they transferred the consensus problem from synchronous systems with simple crash faults to situations in which components may send inconsistent or misleading information to other participants.

The original formulation uses a military metaphor: a set of generals of the Byzantine army surrounds an enemy city and must decide whether to attack or retreat. Each general commands part of the army, but there is no central coordination mechanism. The generals can communicate only through messengers, and some of them may be traitors intending to sabotage coordination by sending false or inconsistent messages. The objective is that the loyal generals reach a common decision, for example to attack, even

in the presence of such malicious messages. This theory shows that, in the classical model without digital signatures, to tolerate up to f Byzantine nodes it is necessary that the total number of nodes be at least $n > 3f$. This result formalizes that **below a certain threshold of reliable nodes, correct consensus is impossible**, regardless of the algorithm used, because correct nodes cannot reliably distinguish between false and legitimate messages when there are too many sources of misinformation.

Classical BFT protocols (such as PBFT, Tendermint, HotStuff (4; 5; 6)) are based on several fundamental assumptions:

- **Known and finite number of participants**: The set of participating nodes is closed and known to all. This implies that each node can have complete information about the identities of the other nodes and verify signatures or certificates.
- **Persistent and verifiable identities**: Nodes have a unique and persistent identity, usually verified through digital signatures. This allows tracking who sent what, preventing impersonation or double participation.
- **Maximum number of Byzantine nodes**: Defined as f, the maximum number of malicious nodes that can be tolerated. The total number of nodes n must satisfy $n \geq 3f + 1$ without digital signatures, or $n \geq f + 1$ with digital signatures.
- **Synchronization model**: Most BFT protocols assume partial synchrony: complete synchronization is not required, but there exists an upper bound on message delay that will hold "eventually."
- **Reliable and authenticated communication**: Messages sent between correct nodes arrive intact and verifiable. This is essential for constructing quorums and voting procedures.

1.3 Limits of classical BFT solutions

Classical BFT solutions, while theoretically elegant and capable of guaranteeing immediate finality and deterministic safety, present significant structural limitations that affect their practical applicability.

The first limitation is **scalability**: communication complexity grows rapidly with the number of nodes ($O(n^2)$ or more), making it difficult to extend these protocols to very large networks without sophisticated optimizations such as rotating leaders or commit pipelines.

A second limitation derives from the **assumption of a known and closed set of participants** with persistent and verifiable identities; this condition makes BFT protocols poorly suited to permissionless contexts or open environments, such as public blockchains, where nodes may freely enter and leave.

Third, classical BFT solutions assume a model of **partial synchrony** (7; 8) in which messages arrive within a known maximum time "eventually"; if the network behaves in a fully asynchronous manner for prolonged periods, the protocol may stall, giving up termination.

Finally, tolerance to Byzantine nodes is limited by theoretical constraints: in the absence of digital signatures, it is necessary that $n \geq 3f + 1$ to tolerate f malicious nodes; if this limit is exceeded, consensus may be compromised (9).

These constraints imply that, although classical BFT solutions are suitable for closed and controlled distributed systems, their application in very large or dynamic networks requires trade-offs, optimized variants, or additional mechanisms to preserve security, availability, and scalability.

1.4 Bitcoin as a turning point

Bitcoin (10) is the first practical implementation of secure **distributed consensus in an open system** in which no central identities exist.

Bitcoin is a **peer-to-peer** distributed system designed to enable the exchange of digital value without the need for a central authority or trusted intermediaries. Formally, it can be viewed as a **public blockchain**, that is, a shared digital ledger replicated among all network participants, in which transactions are stored in **blocks** linked in chronological order. Each block contains a set of transactions, a timestamp, and the hash of the previous block, forming an immutable chained structure that guarantees **integrity and historical consistency** of information (11).

The true innovation of Bitcoin lies in its consensus protocol, which introduces a probabilistic **leader election mechanism based on Proof-of-Work (PoW)**. In this scheme, the entity that "wins" the right to propose a new block is not selected through explicit coordination but emerges implicitly through a computational race: each node may attempt to solve a cryptographic puzzle, and whoever solves it first obtains the privilege of adding the next block.

This approach brings a series of revolutionary consequences. First, consensus is not deterministic: two nodes may temporarily have different views of the blockchain, and blocks may temporarily conflict. However, thanks to the **probability of success being linked to miners' computational power** and to block propagation in the network, the chain converges over time toward **eventual consistency** (12). This means that, as new blocks are added, the probability that a confirmed transaction will be reversed tends to zero, providing a stable and emergent agreement without the need for trusted identities or coordinated voting.

Secondly, Bitcoin addresses the problem of consensus in an **asynchronous** environment in which messages between nodes may experience variable and unpredictable delays. There is no guarantee of immediate data delivery nor global synchronization among participants (13). In Bitcoin, the network tolerates these conditions thanks to the Proof-of-Work mechanism: blocks are added probabilistically, and eventual consistency of the blockchain emerges as new confirmations propagate, ensuring that the main chain prevails over time even in the presence of high latency.

Bitcoin operates in an **open and permissionless environment**, where participants may freely enter and leave the network and where no certified identities or predefined committees exist. This implies that there is no predetermined limit on the number of potentially malicious nodes, nor defined network boundaries (14). The security of consensus in this context is guaranteed by the combination of PoW and economic incentives: honest participation is rewarded with cryptocurrency, while malicious behavior becomes costly due to the computational resources required to compete, discouraging large-scale attacks.

The use of PoW also enables a completely decentralized form of **leader election without any coordination**: each node participates independently, and success is proportional to the resources employed, eliminating the need for a committee or explicit voting. This innovative choice transformed the perception of what "consensus" means in distributed systems: a protocol determined by probability, incentives, and information propagation (15).

1.5 The role of Proof-of-Work

Proof-of-Work represents the conceptual core of the Bitcoin protocol. PoW does not arise as "work for its own sake," but rather as a fundamental mechanism to introduce **unpredictable randomness and resistance to Sybil attacks** in an open distributed system.

In the context of distributed consensus in open systems, **unpredictable randomness** is an essential property to prevent an adversary from anticipating or manipulating the evolution of the protocol. In Bitcoin, this randomness emerges from the use of cryptographic hash functions: given a candidate block, the only way to discover whether it satisfies the difficulty condition is to compute the hash. There are no computational shortcuts or deterministic strategies to predict the outcome. This implies that the moment at which a node succeeds in producing a valid block is a random variable, approximately exponentially distributed and independent of previous attempts. Leader election is therefore not schedulable: the leader emerges as the result of a global stochastic process whose unpredictability is guaranteed by cryptography rather than by trust in participants.

In distributed environments, one of the main attack vectors is represented by **Sybil attacks**, originally introduced by Douceur (16). A Sybil attack occurs when a single adversary creates and controls an arbitrary number of logical identities, presenting them to the system as distinct and independent nodes in order to influence overall behavior (17). The underlying assumption of the attack is that the system assigns participation rights to individual identities without having an effective mechanism to verify their uniqueness. Under such conditions, multiplying identities allows the attacker to artificially amplify influence without a corresponding increase in real resources (18).

Resistance to Sybil attacks is another key property enabled by Proof-of-Work. In an open and permissionless system, there is no central authority capable of limiting identity creation. Without countermeasures, an adversary could generate an arbitrary number of nodes and influence consensus simply by multiplying instances. PoW eliminates this vulnerability by **linking decision power** not to identity but **to a physical resource** external to the system: **computational capacity**. Creating new identities does not increase influence unless accompanied by a proportional increase in resources.

Formally, PoW transforms the Sybil problem from an **identity-control** problem into a **resource-allocation** problem. The probability that an actor controls the consensus process depends not on the number of nodes it owns, but on the fraction of total computational power it controls. This makes Sybil attacks economically measurable and progressively more costly, introducing a natural barrier to malicious participation.

Bitcoin's Proof-of-Work is a consensus mechanism that combines **simple cryptographic verification** with **expensive computation**, with the objective of selecting in a probabilistic and unpredictable manner the node entitled to extend the blockchain.

From a structural point of view, each miner constructs a **candidate block** containing: the hash of the previous block, a set of valid transactions, a timestamp, the current difficulty parameter, and a variable field called a **nonce**. The entire block content is processed by the SHA-256 (19) hash function. The output is a 256-bit numerical value.

The Proof-of-Work rule is **simple to verify** but **difficult to satisfy**: the block hash must be **less than a global threshold** called the ***target***. This threshold determines the problem difficulty: the lower the target, the lower the probability that a random hash satisfies it. Since SHA-256 is modeled as a pseudo-random function, the only way to find a valid hash is to repeat the computation many times, varying the nonce. Each attempt is independent and has a constant probability of success (20).

This process induces a well-defined stochastic distribution. The number of attempts required before finding a valid block follows a geometric distribution, and the time to discover the next block is approximately exponential.

When a miner finds a valid hash, it broadcasts the block to the network. Other nodes deterministically and immediately verify that all transactions are valid, the block correctly references the previous block, and the block hash satisfies the difficulty constraint. If the block is valid, it is accepted and added to the local blockchain. Since multiple miners may find blocks almost simultaneously, the network may temporarily diverge. Bitcoin resolves this situation by adopting the "longest chain" rule. Over time, one chain tends to prevail because it accumulates more Proof-of-Work, and alternative blocks are discarded.

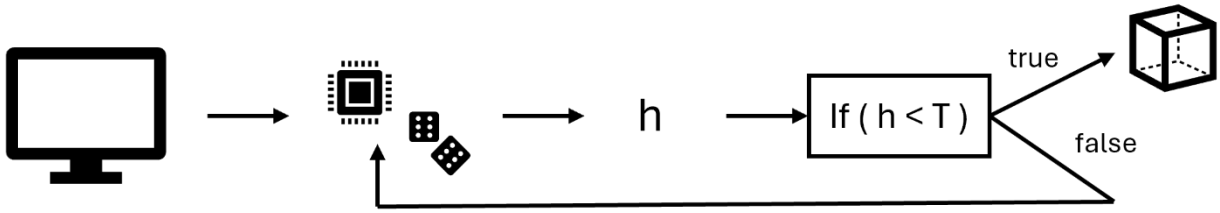

Figure 1: Repeated hash extraction computed by a miner

A fundamental structural element of Bitcoin's Proof-of-Work is the **dynamic difficulty adjustment mechanism**, whose role is to guarantee temporal stability of the rate of the publication of blocks (10 minutes) in an open and time-varying context, explained by Figure 2. Without such a mechanism, the rate of block production would be directly proportional to the total computational power of the network, making the system extremely sensitive to the entry or exit of miners and therefore unstable both operationally and economically.

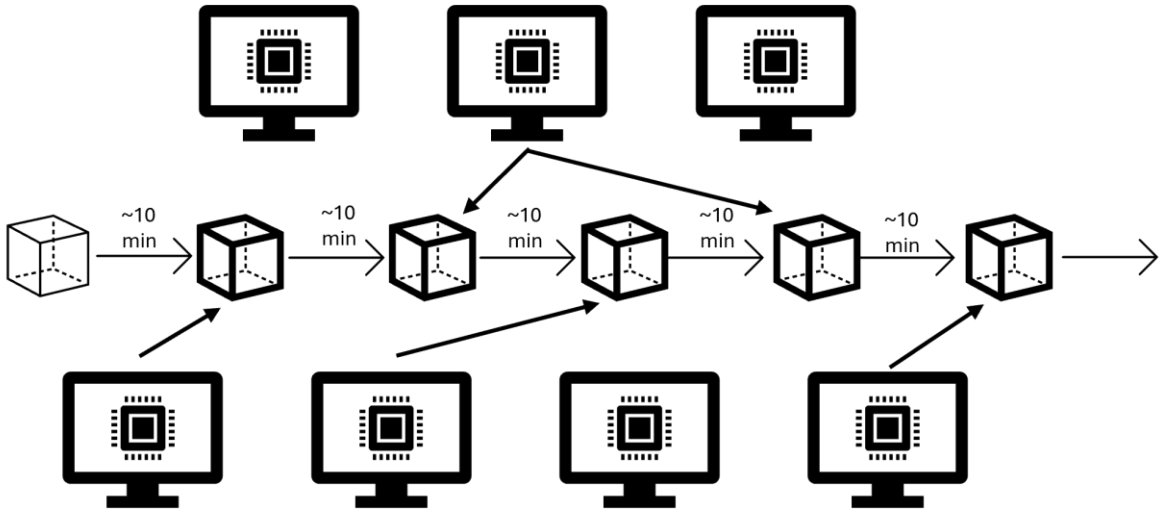

Figure 2: Construction of the Blockchain by Bitcoin miners' community

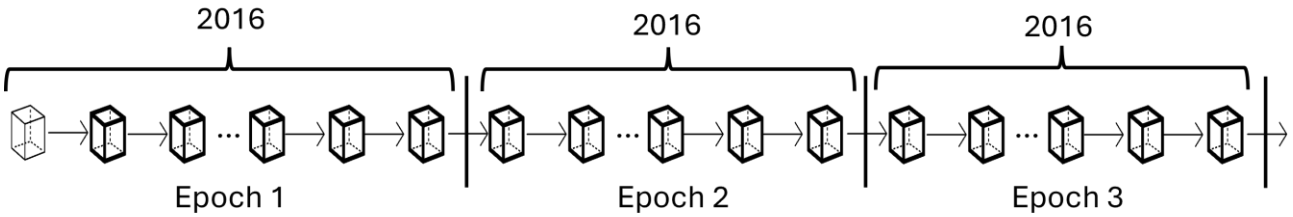

Figure 3: Subdivision of the blockchain into Epochs

In the Bitcoin protocol, difficulty is updated every **2016 blocks**, an interval that under ideal conditions corresponds to approximately two weeks, as illustrated in Figure 3. This interval is defined as an ***epoch***. At the end of each epoch, the protocol compares the **actual time** required to produce the last 2016 blocks with the **expected time**, equal to 2016 × 10 minutes. If blocks were produced faster than expected, this indicates an increase in global computational power; consequently, the protocol increases the difficulty, reducing the probability that a single hash attempt produces a valid block. Conversely, if production was slower than expected, the difficulty is reduced, compensating for a decrease in available computing power (21).

PoW implements a form of **probabilistic leader election without identities**. Each mining attempt is equivalent to a "lottery ticket," and the probability of winning is proportional to the fraction of computational power owned. There is no voting, coordination, or predefined membership: the leader emerges implicitly from the computation process.

1.6 Limits of Proof-of-Work and sustainability

In the Bitcoin model, protocol security is directly proportional to the amount of computational work invested by the network. As the economic value protected by the blockchain increases, the incentive to invest in computational power grows, triggering competition that translates into ever-greater energy consumption.

In a Proof-of-Work system such as Bitcoin, the computational work performed by miners does not contribute to an objective external to consensus itself: the computation is "useless" from an application standpoint, because its function is exclusively to make participation in the validation process costly. This implies that an increase in security cannot be achieved through algorithmic improvements, but only through an increase in the consumption of physical resources, primarily electrical energy (22).

Bitcoin's Proof-of-Work requires **repeated computation** of the SHA-256 function applied to the block header, as shown in Figure 1. This operation is simple, highly parallelizable as depicted in Figure 4, and free of sequential dependencies. From a hardware perspective, this means the computation can be implemented in extremely optimized pipelines, for example using ASICs (*Application-Specific Integrated Circuits*), integrated circuits designed to perform a **single specific function** as efficiently as possible. Energy consumption therefore arises from the massive and continuous execution of these elementary operations.

The probability that a single hash satisfies the difficulty condition is extremely low (for example 2^{-70} or lower). Consequently, each miner must perform on average a number of attempts inversely proportional to the difficulty target before achieving success.

It is estimated that the Bitcoin network currently consumes at least 2.55 gigawatts of electricity and potentially 7.67 gigawatts in the future, making it comparable to countries such as Ireland (3.1 gigawatts) and Austria (8.2 gigawatts) (23).

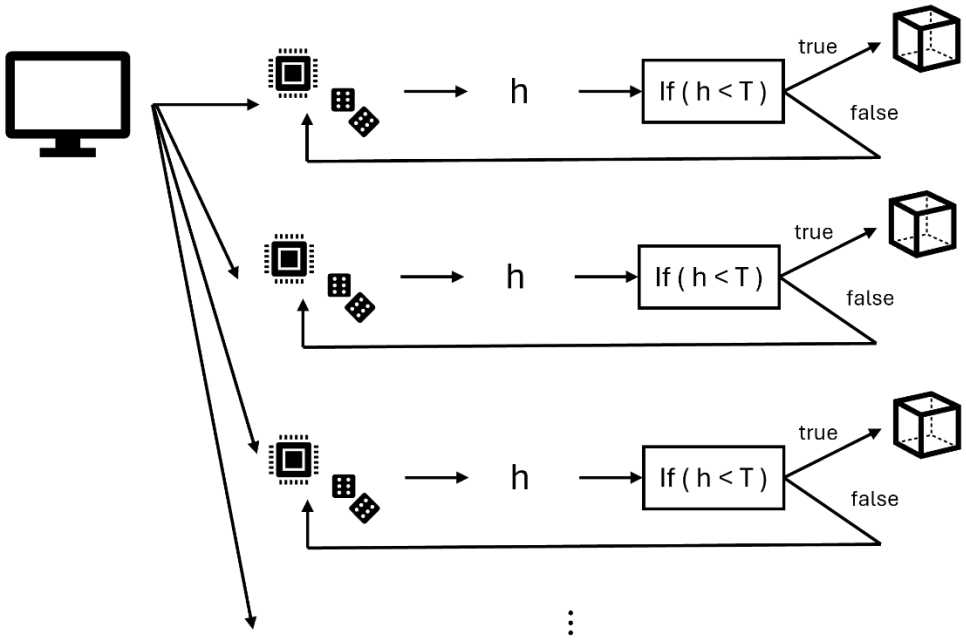

Figure 4: Parallelization of hashing procedure

Another determining element is the **difficulty adjustment mechanism**. Since Bitcoin maintains a constant average time between blocks, any increase in global computational power is compensated by an increase in difficulty. This implies that hardware efficiency does not reduce the total number of hashes required per block but simply shifts the system toward a regime in which more total hashes are executed per unit of time. In other words, the protocol forces the system to consume all available computational power until the predefined temporal target (one block every 10 minutes) is reached (24).

Each hash attempt is completely independent of previous ones and **does not contribute in any way to reducing the future search space**. This is a security requirement: if work were accumulable, an adversary could precompute advantages or reuse past work. However, from an energy standpoint, this means that almost all of the energy spent is dissipated without leaving any trace in the system.

1.7 Critical analysis of alternative blockchain consensus mechanisms

In the years following the introduction of Bitcoin, numerous consensus algorithms have been proposed with the objective of improving efficiency, latency, and throughput of blockchains. Many of these approaches arise as a direct response to the practical limits of Proof-of-Work, particularly its high energy consumption and low transaction capacity. However, in pursuing these objectives, such systems substantially modify the underlying consensus model, introducing assumptions that place them in a conceptually distinct category compared to Bitcoin.

Proof-of-Stake protocols (25) (26) (27) (28) (29) represent the most direct attempt to replace Proof-of-Work while maintaining, at least in appearance, system openness. Formally, anyone can become a validator, provided they possess the required capital. However, from a consensus standpoint, the

system is no longer neutral with respect to identities. Stake functions as a persistent and weighted identity: participation alone no longer matters, but rather the amount of economic resources locked. This introduces an implicit and relatively stable membership, because validators must remain online, identifiable, and economically bound in order to participate continuously. Consensus is almost always achieved through voting and quorum among selected validators, according to schemes clearly traceable to classical Byzantine consensus. In this sense, PoS does not eliminate the membership problem, but shifts it from the plane of identities to the plane of capital.

A second line of work consists of committee-based protocols (5) (6) (30) (31) often derived from classical Byzantine consensus algorithms such as PBFT. In these systems, consensus is reached by a limited subset of nodes forming a committee, which changes over time but remains relatively small compared to the total set of participants. Within the committee, nodes execute structured communication phases, elect a leader, and vote on proposed blocks.

There are also hybrid protocols that combine elements of Proof-of-Work, Proof-of-Stake, and Byzantine consensus. In these cases, PoW or VRFs are often used as lottery mechanisms to select a committee, while the actual consensus takes place through voting within that group. Even in this scenario, the role of the probabilistic mechanism is not to directly produce consensus, but to restrict the problem to a sufficiently small set of nodes to make traditional BFT protocols applicable (32).

The common trait of these approaches is that, while operating in a blockchain context, they **implicitly reintroduce assumptions typical of the classical Byzantine Generals Problem**. In particular, consensus is achieved by a finite and relatively restricted set of nodes, endowed with persistent identities and involved in an explicit coordination process. This represents a significant change compared to the model introduced by Bitcoin, in which no privileged set of participants exists, and the leader emerges unpredictably as the result of a distributed and uncoordinated process.

1.8 Open research problem

Bitcoin's consensus introduced a radically new property in open distributed systems: the possibility of reaching agreement in a permissionless environment, without a predefined set of validators, without a fixed leader, and without relying on a trusted third party. This architecture produces emergent properties of great relevance, in particular the unpredictable election of the block proposer, resistance to Sybil attacks, and the absence of committees, which do not derive from a single explicit mechanism but from the overall dynamics of Proof-of-Work.

However, these properties are obtained at the cost of massive and increasing use of computational and energy resources. In Proof-of-Work, protocol security and timing are intrinsically linked to a repetitive and competitive computation process in which the vast majority of work performed does not directly contribute to the production of new blocks. This link between security and energy consumption is not an accidental side effect but a structural consequence of the mechanism itself.

In the years following Bitcoin's proposal, numerous alternative consensus protocols have sought to overcome this limitation by introducing models based on Proof-of-Stake, elected committees, predefined leaders, or forms of explicit coordination (33). However, most of these approaches do not introduce a fundamentally new principle for achieving consensus in permissionless environments, but rather rely on technologies and mechanisms already existing, such as elected committees, predefined leaders, staking systems, or explicit coordination among nodes.

These solutions, although more energy-efficient, tend to replace computational work with additional assumptions about participant behavior or network structure, shifting the problem rather than solving it. In particular, the election of the block proposer often becomes an explicit, partially predictable

process or one constrained to restricted sets of nodes, reducing the openness and unpredictability that characterize Bitcoin's consensus.

In this sense, such protocols do not provide a conceptual alternative to Proof-of-Work but circumvent its costs by relying on mechanisms already known in the distributed systems literature.

1.9 Key Idea of the Thesis

The consensus introduced by Bitcoin can be interpreted as a mechanism that governs **time and the ordering of node actions** in an open system, rather than as a mechanism founded on the consumption of computational work. In Proof of Work, the repeated computation of hashes is the means through which the protocol achieves the generation of the blockchain over time: blocks are published at unpredictable instants, distributed at nearly regular time intervals, without explicit coordination among participants.

This observation suggests that the fundamental problem of permissionless consensus is **how to determine, in an unpredictable and verifiable way, when a node may act**, preventing anyone from arbitrarily manipulating the moment of publication. In Bitcoin, this temporal constraint emerges from the hashing process; however, nothing implies that this process must be the only way to obtain it.

The thesis proposes to make **explicit** the characteristics that in Proof of Work are fused into a single mechanism: the generation of randomness, the determination of a waiting time, and the public verification of compliance with such waiting. By focusing on these aspects, it becomes possible to replace wasteful computational work with more targeted primitives.

The first element introduced is a source of **unpredictable and verifiable randomness**, necessary to prevent nodes from strategically choosing when to publish a block. For this purpose, a **Verifiable Random Function** (VRF) (34) is introduced, allowing each node to extract a pseudo-random value that is deterministic with respect to its private key and a protocol-defined element available on the chain, but unpredictable to anyone else until the moment of publication. The VRF replaces the implicit role played by the winning hash in Proof of Work: providing a random result that cannot be anticipated nor manipulated after the fact.

The second element is the **translation** of **such randomness** into a **waiting time**. In Proof of Work, the time that elapses before the discovery of a valid block follows an exponential distribution; in the proposed protocol, this behavior is reproduced by extracting a random variable with a **geometric distribution**, representing the number of abstract time units that a node must wait before being allowed to publish. In this way, the waiting time is determined explicitly in probabilistic form, preserving the same statistical dynamics observed in Proof of Work blockchains such as Bitcoin. However, just as in Bitcoin, the probability of success cannot be considered static. In a permissionless blockchain, the total computational power of the network varies over time, making it necessary to introduce a **difficulty adjustment mechanism** in order to maintain a stable average rate of block production. In the proposed system, difficulty is updated at discrete intervals (epochs), analogously to what happens in Bitcoin, and each difficulty value determines a different probability of success per abstract time unit. This observation introduces a substantial difference compared with the classical geometric distribution: when difficulty changes, the probability of success also changes, and therefore the waiting process can no longer be modeled as a single geometric distribution with a constant parameter. To address this problem, the protocol adopts a **Geometric Distribution with Variable Probability** (GD-VP) (35), which extends the geometric distribution to the case in which the probability of success remains constant within each epoch but may vary between successive epochs.

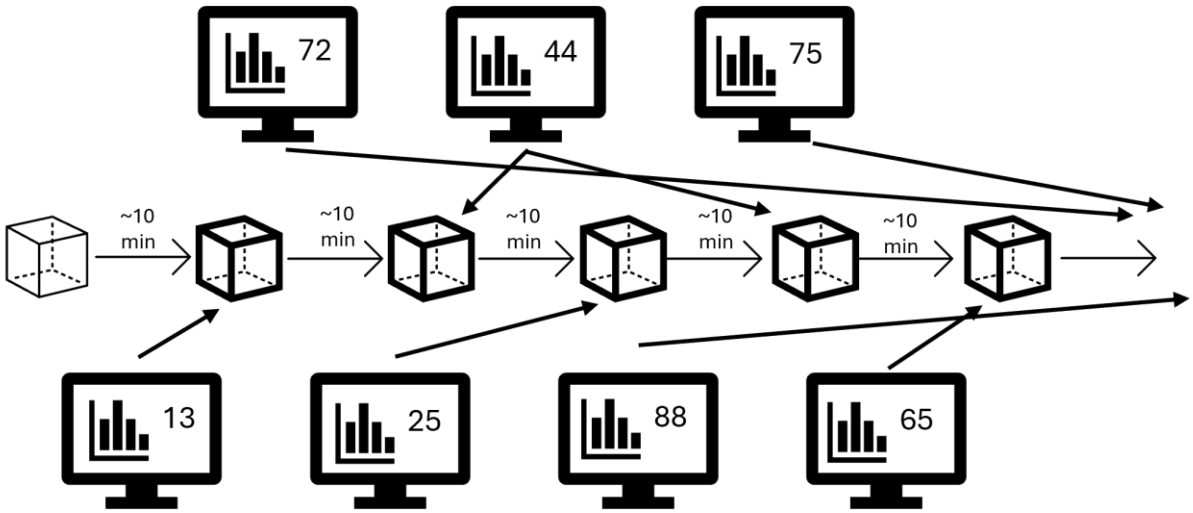

Figure 5: Block publication deadlines prediction instead of PoW

The third element, crucially, is to make this **waiting real** and **non-bypassable**. Knowing that one must wait a certain time is not sufficient if the node can simulate the waiting or compute it in advance. For this reason, the protocol introduces a **Verifiable Delay Function** (VDF) (36; 37), a function that requires a sequential number of operations to be evaluated and that cannot be accelerated through parallelism or more powerful hardware. The VDF transforms the extracted waiting time into a physical constraint: the node must actually wait before being able to publish, and it can demonstrate this efficiently to other participants.

By combining these elements, the publication of a block becomes the outcome of the following process: each node extracts a verifiable random value, maps it to a probabilistic waiting time, and proves that it has respected such waiting through a proof of time. Consensus thus emerges not from the consumption of resources, but from the **statistical convergence of independent waiting times**, analogously to what occurs in Proof of Work, but without requiring competitive and continuously repeated computation. Figure 5 highlights the parallelism between PoW consensus mechanism and our proposed consensus mechanism deadline extraction.

From this perspective, the proposed protocol does not aim to replace Bitcoin with a completely different mechanism, but rather to **recapture the fundamental principle of Nakamoto's invention**—the spontaneous ordering of actions in an open system—separating it from the specific technical method used to realize it. The following sections formalize this idea, describing in detail how randomness, time, and verification are implemented and combined in a complete consensus protocol.

1.10 Contributions and Structure of the Thesis

This thesis addresses the problem of designing a distributed consensus protocol that maintains properties of fairness, verifiability, and robustness in open and untrusted environments, while reducing the energy inefficiencies typical of Proof-of-Work-based mechanisms.

The main contribution of this work consists in the design and analysis of a consensus protocol that integrates mechanisms for verifiable randomness generation, verifiable delay functions, and a probabilistic model of block publication time, with the objective of obtaining predictable, verifiable behavior resistant to local manipulation. The protocol has also been implemented in a simulation prototype that made it possible to analyze its behavior in controlled scenarios, varying parameters such as network latency, node computational power, and global difficulty.

In addition to proposing the protocol, the thesis provides:

- a formalization of the probabilistic model describing block publication time
- an analysis of the role of verifiable randomness in distributed protocols
- an experimental validation based on simulation

The structure of the thesis is organized as follows.

Chapter 2 introduces the probabilistic model of block publication time. The behavior of a miner is described, the distribution used to model the publication process is presented, and the transformation method employed to obtain block publication deadlines is explained. Furthermore, the GD-VP model is presented, which constitutes one of the fundamental elements of the protocol.

Chapter 3 presents the Verifiable Random Function, discussing its formal properties, the motivations for its use, and the methods of integration into the protocol, with particular attention to the mapping between pseudo-random output and publication time.

Chapter 4 introduces the Verifiable Delay Function, explaining why verifiable randomness alone is not sufficient and illustrating the role of VDFs in guaranteeing sequentiality and resistance to temporal manipulation. The main security aspects are also analyzed.

Chapter 5 describes the proposed consensus protocol. After an overview of the components, the complete operational flow is presented, and collision and fork cases are analyzed, highlighting the system's behavior in the presence of concurrent events.

Chapter 6 presents the prototype developed for experimental validation. The objectives, the simulator architecture, the main implementation choices, and the experiments conducted to evaluate protocol behavior under varying network and computational parameters are described.

Chapter 7 addresses the problem of resistance to Sybil attacks. The intrinsic vulnerability of open systems is analyzed and several countermeasures are discussed, with particular attention to participation costs, computational costs linked to identity, and a reflection about permissionless and registration-less features.

Finally, Chapter 8 summarizes the results obtained and presents possible future developments.

2. Probabilistic Model of Publication Time

This chapter presents the probabilistic model used to describe the block publication process. After defining the mining context and the behavior of a miner, it recalls the fundamentals of the geometric distribution and the inverse transform method for random variable generation.

The difficulty adjustment mechanism is then discussed, namely using the same methodology adopted by Bitcoin, and finally the GD-VP distribution is introduced, which extends the model to the case in which the probability of success varies over time discretized into epochs, making it possible to represent the process in the presence of difficulty updates.

2.1 Context

In the context of Proof of Work, the validation of a block is a computationally expensive operation. It requires searching for a block whose hash is lower than a predefined threshold, determined by the current difficulty of the protocol. The only known way to obtain such a result consists in repeatedly executing a hash function, varying at each attempt a specific field of the block (the nonce) in order to generate distinct inputs. Each attempt constitutes an independent trial, whose positive outcome occurs with extremely low probability.

From a probabilistic point of view, the number of attempts required to obtain the first success in a sequence of independent and identically distributed Bernoulli trials follows a **geometric distribution**. This observation is not an arbitrary abstraction, but a direct consequence of the functioning of Proof of Work: each hash is independent of the previous ones and provides no useful information to increase the probability of success of the next attempt (38).

In other words, the geometric distribution exactly models the random variable “number of hashes required before finding a valid block,” which is the fundamental object of the mining process in Bitcoin.

A fundamental aspect of permissionless blockchains is that the number of miners and their total computational power are not known a priori and may vary significantly over time. Consequently, it is not possible to statically configure system parameters in order to guarantee a predetermined blockchain progress rate (Blockchain Progress Rate, BPR). For this reason, Proof of Work protocols introduce a dynamic adjustment mechanism based on a parameter called difficulty, which is periodically increased or decreased as a function of the block publication rate, with the objective of keeping the BPR approximately constant.

Difficulty Adjustment

As already mentioned, maintaining a constant average **Blockchain Progress Rate** (BPR), which can be expressed as the inverse of the **Inter Block Time** (IBT), requires compensating for variations in the **Global Hash Rate** (GHR), i.e., the total computational power made available by miners.

The only mechanism available to achieve such compensation is **difficulty adjustment**, illustrated in Figure 6.

The problem can be formalized as a classical control problem. There exists a target value, the **Target Inter Block Time** (TIBT), which in Bitcoin is set to 10 minutes, and an observed value, the **Measured Inter Block Time** (MIBT), which depends unpredictably on the GHR. The control variable is the difficulty D, which is periodically updated as a function of the deviation between MIBT and TIBT in order to bring the system back toward the desired regime.

It is important to emphasize that the regulator is not an external or centralized entity: the adjustment logic is **replicated locally in every miner**, which autonomously computes the new difficulty based on publicly available information on the blockchain.

Since the MIBT is affected by high statistical variability, it is not used directly but estimated as an average over a window of consecutive blocks. In Bitcoin, this window is equal to 2016 blocks, corresponding to approximately two weeks of real time.

At the beginning of each new window, miners update the difficulty by rescaling the previous one according to the relation:

$$D_{i+1} = D_i \cdot \frac{\text{TIBT}}{\text{MIBT}_i}$$

During each window, instead, difficulty remains **constant**, and the block production process proceeds with an unchanged probability of success.

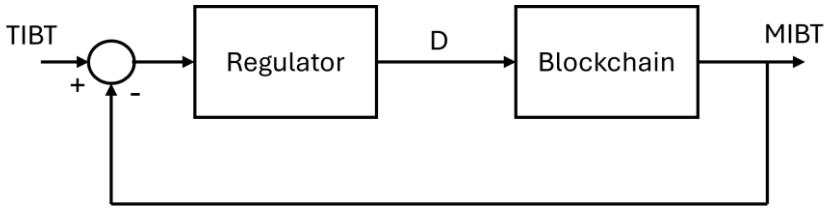

Figure 6: Difficulty Regulation scheme

Adaptation of Our Protocol

It follows that the evolution of the blockchain can be seen as a **succession of sequences of blocks**, each characterized by a constant difficulty, separated by discrete instants in which the difficulty is updated.

This observation has a fundamental consequence: when difficulty evolves over time, it is **no longer possible to describe the waiting time for the next block by means of a standard geometric distribution**, since the latter assumes a constant probability of success. In particular, if a success event falls in a future sequence relative to the one in which the process was initiated, the classical geometric distribution is no longer applicable.

GD-VP represents an extension of geometric distribution to contexts in which the probability of success **changes over time**. Its application to blockchain simulation demonstrates how it is possible to update block publication deadlines in response to difficulty adjustments without invalidating the progress already made. In summary, each miner initially generates the deadline of its next publication by extracting a random variable according to a geometric distribution; subsequently, whenever the difficulty changes, such deadline is updated using the GD-VP formulation.

2.2 Model of a Miner

In the Proof of Work protocol, the publication of a valid block is conditional on satisfying a cryptographic condition: the hash of the entire block structure must be lower than a predefined threshold T. To verify

this condition, a miner executes a potentially infinite loop in which, at each iteration, it constructs a new block, computes its hash, and checks whether the obtained value satisfies the imposed constraint. If the condition is satisfied, the block is declared valid and immediately published to the network.

The **nonce** field ensures that each attempt produces a different input to the hash function, even in the absence of modifications to the block content. The execution loop is conceptually non-terminating: as long as a miner does not find a valid hash, it continues to generate and verify new attempts. Since hash computation is by far the most computationally expensive operation, miners can be considered continuously engaged in hashing operations. This continuous process is what is commonly referred to as **mining**.

Assuming that the cryptographic hash function produces outputs uniformly distributed over the entire space of possible values, each hashing attempt can be modeled as an independent random trial with a binary outcome: success, if the hash is lower than threshold T; failure, otherwise. In this context, the publication of a valid block by a miner occurs at the k^{th} attempt, after $k-1$ consecutive failures followed by a single success.

2.3 Geometric Distribution

This behavior corresponds exactly to the definition of geometric distribution. If we denote by p the probability of success of a single trial and by $q = 1 - p$ the probability of failure, the probability that the first success occurs exactly at the k^{th} attempt is described by the probability mass function of the geometric distribution:

$$f(k) = p\, q^{k-1}$$

while the cumulative distribution function is given by:

$$F(k) = 1 - q^k.$$

In the context of Proof of Work, the probability of success of a single hash is equal to the ratio between threshold T and the total range of values producible by the hash function, denoted by R. Introducing the parameter Difficulty $D = R/T$, we obtain:

$$p = \frac{1}{D}, \qquad q = 1 - \frac{1}{D}.$$

This formalizes the fact that, as difficulty increases, the probability that a single hashing attempt produces a valid block decreases in inverse proportion.

Consequently, the number of hash operations that a miner must execute before successfully publishing a valid block can be modeled as a geometrically distributed random variable. Knowing the computational power of the miner, expressed as the average number of hashes computable per unit of time, it is possible to convert this discrete quantity into an estimated waiting time for block publication.

2.4 Inverse Transform Method

To generate samples of a random variable with geometric distribution, it is possible to use the inverse transform method. Given a random number u extracted uniformly in the interval $[0,1]$, the corresponding value k distributed according to a geometric distribution can be obtained by applying the inverse of the cumulative distribution function:

$$k = F^{-1}(u)$$

In the specific case of a geometric distribution with probability of success $p = 1/D$, this transformation takes the form:

$$k = \frac{\ln(1-u)}{\ln\left(1-\frac{1}{D}\right)}$$

The value k thus obtained represents the number of hash operations that the miner must execute before finding a valid block. Dividing this quantity by the miner's hash rate and adding the result to the current time yields an estimate of the next block publication time.

This modeling approach makes it possible to accurately capture the stochastic behavior of mining in Bitcoin, highlighting how the block publication process naturally emerges from a sequence of independent and random attempts.

2.5 GD-VP

Consider a sequence of Bernoulli trials, each producing a binary outcome: **success** or **failure**. Unlike the classical geometric distribution, in which the probability of success is constant over time, we assume here that this probability may vary during execution.

Formally, the sequence of trials is organized into an ordered set of subsequences S_i, each associated with:

- a constant probability of success p_i,
- a probability of failure $q_i = 1 - p_i$,
- a finite number of events n_i.

In the blockchain context, each subsequence S_i represents an **epoch of constant difficulty**, while each trial corresponds to a mining attempt (that is, conceptually, to a hash operation).

We focus on the distribution of the random variable k, which represents the **number of failed events preceding the first successful event**, assuming that this success occurs within subsequence S_m.

Probability Density Function (PDF)

The probability that the first success occurs exactly at step k, with $k \in S_m$, is given by:

$$f(k) = P(t = k, k \in S_m) = \prod_{i=0}^{m-1} q_i^{n_i} \cdot q_m^{k-1-\sum_{i=0}^{m-1} n_i} \cdot (1 - q_m)$$

Cumulative Distribution Function (CDF)

The probability that the success occurs by step k, again with $k \in S_m$, is:

$$F(k) = P(t \leq k, k \in S_m) = 1 - \prod_{i=0}^{m-1} q_i^{n_i} \cdot q_m^{k-\sum_{i=0}^{m-1} n_i}$$

Sampling via the Inverse Transform Method

To sample a random variable k distributed according to GD-VP, the **Inverse Transform Method** is applied.

Given a value u uniformly distributed in $[0,1]$, we impose:

$$u = F(k)$$

and solve with respect to k. In the blockchain context, the probability of success of a single attempt is determined by the difficulty D_i of epoch i, according to:

$$p_i = \frac{1}{D_i}, q_i = 1 - \frac{1}{D_i}$$

Substituting into the CDF and inverting, the final formula is obtained:

$$k = \sum_{i=0}^{m-1} n_i + \frac{1}{\ln\left(1-\frac{1}{D_m}\right)}\left(\ln(1-u) - \sum_{i=0}^{m-1} n_i \ln\left(1-\frac{1}{D_i}\right)\right)$$

In our model, the value k represents **the number of hash operations that a miner would need to execute before obtaining a valid hash**, that is, before being authorized to publish a valid block.

A fundamental property of this formulation is that the computation of k can be performed **incrementally**, maintaining **two** accumulators:

- $\sum_{i=0}^{m-1} n_i$, representing the total number of attempts already "consumed" in previous epochs;
- $\sum_{i=0}^{m-1} n_i \ln\left(1-\frac{1}{D_i}\right)$, incorporating the cumulative effect of past difficulties.

When difficulty changes (that is, at the beginning of a new epoch), these accumulators are updated in constant time $O(1)$, without the need to re-sample the random value u nor to "reset" the process. This behavior faithfully replicates real Proof of Work, in which work already performed is never invalidated by a change in difficulty.

Proof

Consider a sequence of independent Bernoulli trials, each producing a binary outcome: success or failure. The random variable of interest is k, which represents the number of iterations required to obtain the **first success**, preceded by $k-1$ failed attempts.

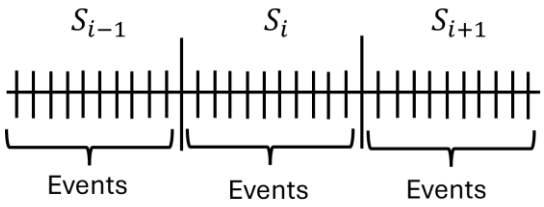

Figure 7: Subsequences of events

Unlike the classical geometric distribution, we assume that the probability of success is not constant over time. The trials are organized into an ordered sequence of subsequences, as shown in Figure 7.

$$S_0, S_1, \dots, S_{m-1},$$

Within each subsequence the probability remains unchanged, while it may change when moving from one subsequence to the next. The objective is to compute the **PDF** and the **CDF** of k in the case where success occurs within subsequence S_m. The result is obtained by **induction**, starting from the base case.

Assume initially that the probability of success is $p = p_0$ (and therefore $q_0 = 1 - p_0$). For the first time instants $t \in S_0$, the probability that the first success occurs exactly at time t is:

$$P(t \le 1) = P(t = 1) = 1 - q_0$$
$$P(t = 2) = [1 - P(t \le 1)]p_0 = q_0 - q_0^2$$
$$P(t \le 2) = 1 - q_0^2$$
$$P(t = 3) = [1 - P(t \le 2)]p_0 = q_0^2 - q_0^3$$
$$P(t \le 3) = 1 - q_0^3$$
$$P(t = 4) = [1 - P(t \le 3)]p_0 = q_0^3 - q_0^4$$
$$P(t \le 4) = 1 - q_0^4$$

This behavior clearly shows that, as long as the probability remains constant, the distribution of k coincides with a classical geometric distribution.

Without loss of generality, assume now that at time $t = 5$ the probability of success is updated from p_0 to p_1, entering subsequence S_1.

The probability of success at time $t = 5$ is:

$$P(t = 5) = [1 - P(t \le 4)]p_1 = q_0^4 - q_0^4 q_1$$
$$P(t \le 5) = 1 - q_0^4 q_1$$

Similarly, for subsequent times:

$$P(t = 6) = [1 - P(t \le 5)]p_1 = q_0^4 q_1 - q_0^4 q_1^2$$
$$P(t \le 6) = 1 - q_0^4 q_1^2$$

$P(t=7) = [1 - P(t \leq 6)]p_1 = q_0^4 q_1^2 - q_0^4 q_1^3$
$P(t \leq 7) = 1 - q_0^4 q_1^3$

It can be observed that the contribution of previous epochs appears as a **cumulative multiplicative factor.**

Assume now that at time $t = 8$ the probability is updated to p_2, entering subsequence S_2. We obtain:

$P(t=8) = [1 - P(t \leq 7)]p_2 = q_0^4 q_1^3 - q_0^4 q_1^3 q_2$
$P(t \leq 8) = 1 - q_0^4 q_1^3 q_2$
$P(t=9) = [1 - P(t \leq 8)]p_2 = q_0^4 q_1^3 q_2 - q_0^4 q_1^3 q_2^2$
$P(t \leq 9) = 1 - q_0^4 q_1^3 q_2^2$

By induction, in the general case where time k belongs to subsequence S_i, the relations hold:

$P(t=k) = [1 - P(t \leq k-1)]p_i$
$P(t \leq k) = P(t \leq k-1) + P(t=k)$

These expressions constitute the basis for deriving the closed form of the **Geometric Distribution with Variable Probability (GD-VP)**, from which the general PDF and CDF presented in the previous sections are obtained.

The probability that the first success occurs exactly at step k, with $k \in S_m$, is therefore:

$$f(k) = P(t=k, k \in S_m) = \prod_{i=0}^{m-1} q_i^{n_i} \cdot q_m^{k-1-\sum_{i=0}^{m-1} n_i} \cdot (1 - q_m)$$

The probability that the success occurs by step k, again with $k \in S_m$, is:

$$F(k) = P(t \leq k, k \in S_m) = 1 - \prod_{i=0}^{m-1} q_i^{n_i} \cdot q_m^{k-\sum_{i=0}^{m-1} n_i}$$

Applying the Inverse Transform Method, we set:

$$u = F(k) = 1 - \prod_{i=0}^{m-1} q_i^{n_i} \; q_m^{k-\sum_{i=0}^{m-1} n_i}$$

and solve with respect to k. Rearranging the terms, we obtain:

$$1 - u = \prod_{i=0}^{m-1} q_i^{n_i} \; q_m^{k-\sum_{i=0}^{m-1} n_i}$$

Applying the natural logarithm to both sides:

$$\ln(1-u) = \sum_{i=0}^{m-1} n_i \ln q_i + \left(k - \sum_{i=0}^{m-1} n_i\right) \ln q_m$$

From which k is explicitly obtained:

$$k = \sum_{i=0}^{m-1} n_i + \frac{1}{\ln q_m} \left(\ln(1-u) - \sum_{i=0}^{m-1} n_i \ln q_i \right)$$

3. Verifiable Random Function

This chapter introduces Verifiable Random Functions (VRFs) and their role within the protocol. After describing the fundamental properties of this cryptographic primitive and the limitations of an approach based on signature and hash, it provides a formal definition of VRFs and illustrates their use for generating verifiable pseudo-random values. Subsequently, the chapter describes the mechanism by which these values are transformed into waiting times through the geometric distribution, and finally presents an analysis of the main security implications, with particular attention to grinding attacks and Sybil attacks.

3.1 The Need for Verifiable Randomness

Verifiable Random Functions (VRFs) were introduced as a cryptographic primitive to solve a well-defined problem: generating an apparently random value that is **deterministic for the holder** of a secret key, yet **publicly verifiable** without revealing any information about the key itself or about possible alternative outputs. In other words, a VRF allows an entity to produce a pseudo-random value bound to a public input, together with a cryptographic proof that guarantees to anyone that the value has been generated correctly and uniquely from a specific private key.

This structure implies a set of fundamental characteristics:

- **Determinism**
- **Pseudo-randomness**
- **Public verifiability**
- **Non-malleability**

First, **conditional determinism**: for a given private key and a given input, the VRF output is always the same. This removes ambiguity and prevents the output generator from retroactively choosing among multiple possible results.

Second, **pseudo-randomness**: for anyone who does not possess the private key, the output is indistinguishable from a random value, even when the input and public key are known.

Third, **public verifiability**: anyone can verify the correctness of the output without trusting the generator.

Finally, a frequently underestimated but crucial property is **non-malleability**: it is not possible to derive the VRF output for a different input, nor to manipulate a valid output to make it valid for another message.

From the perspective of inputs and outputs, a VRF receives an arbitrary but well-defined input and produces an output that is a fixed-length bit string. This output is then used as a source of “controlled” randomness, for example to decide leader election, validator selection, or the execution order of events in a distributed protocol. The proof associated with the output does not serve a randomness function, but rather a guarantee function: it exists solely to enable public verification of the correctness of the value.

The use of VRFs is particularly relevant in distributed systems and consensus protocols. In blockchain and distributed ledger contexts, VRFs are used for pseudo-random and verifiable validator selection (as in Algorand (31)), or to generate manipulation-resistant randomness beacons. In more general settings,

they apply wherever it is necessary to balance unpredictability, verifiability, and the absence of trust in a single actor.

3.2 First approach

In our initial project, the objective was to obtain a pseudo-random value cryptographically bound to a client's identity, without introducing external trust mechanisms. To achieve this goal, a multi-step procedure was designed.

First, a **public and deterministic message** was defined, identical for all protocol participants. This message could represent, for example, the identifier of a block in the existing chain, i.e., data known in advance and not manipulable by an individual client. The purpose of this message was to serve as a common input, ensuring that the procedure was repeatable and verifiable by any observer.

Next, each client used its **private key** to cryptographically sign this public message, applying a standard digital signature scheme. The result was a valid signature demonstrating unambiguously possession of the private key corresponding to the client's public key. At this stage, anyone holding the public key could verify the correctness of the signature and thus the authenticity of the signer.

Once obtained, the signature was not used directly as the protocol output. Instead, it was treated as an intermediate value and passed through a cryptographic hash function, with the goal of transforming it into a fixed-length bit string that was statistically uniform and apparently random. The output of the hash function constituted the final pseudo-random value produced by the client.

In summary, the overall process followed this conceptual chain:

- selection of a public and shared message
- signing of the message using the client's private key
- public verification of the signature using the public key
- application of a hash function to the signature to derive a pseudo-random value
- use of that value as a randomness source within the protocol

This mechanism had the advantage of being simple to implement and based exclusively on well-established cryptographic primitives. Moreover, it ensured that the pseudo-random value was strictly bound to the client's cryptographic identity and to a common public input, while providing a form of indirect verifiability through signature checking.

At first glance, this approach appears to satisfy certain requirements: the signature is publicly verifiable, it is tied to a private key, and the hash produces an output that appears random. However, this construction exhibits structural weaknesses that emerge as soon as one formally analyzes the properties required of a VRF.

The main issue lies in the absence of constrained **uniqueness of the output**. In many signature schemes, multiple valid signatures exist for the same message and private key, especially if the scheme is probabilistic or includes internal randomness. This means the client can generate multiple different signatures over the same public message, hash them, and selectively choose the one that produces the "most convenient" output. In other words, the generator can influence the pseudo-random output, violating the requirement of unpredictability and fairness. Even in deterministic signature schemes, the absence of a specific proof of pseudo-randomness makes the output potentially distinguishable in more complex contexts.

A second critical aspect is the **lack of a separate and structured cryptographic proof** that uniquely binds input, output, and public key. Signature verification guarantees only that the message was signed by the holder of the private key, not that the hashed output is the only correct and non-manipulable value. The verification is therefore indirect and incomplete compared to the guarantees provided by a formally defined VRF.

VRFs solve these issues at the root, as they are specifically designed to prevent opportunistic output selection and to provide formal guarantees of correctness, uniqueness, and verifiable pseudo-randomness. Adopting a VRF is therefore not merely an implementation-level substitution of signature plus hash, but a paradigm shift: from "derived randomness" to cryptographically bound and verifiable randomness, which becomes a reliable component upon which complex and robust protocols can be built.

3.3 Formal Definition of VRF

Formally, a VRF is defined by three main algorithms: $KeyGen$, $Eval$, and $Verify$.

The first generates a key pair, consisting of a private key and a public key.

The second takes as input the private key and a message (typically public or otherwise known to all parties) and produces two outputs: a VRF value, which constitutes the pseudo-random output, and a cryptographic proof.

The third algorithm allows anyone, using the public key, the input message, the VRF output, and the proof, to verify that the output was indeed generated by the holder of the corresponding private key and that it is the only possible one for that key–input pair.

More formally, a VRF consists of three algorithms:

$(pk, sk) = KeyGen(\lambda)$
Algorithm that, given a security parameter λ, generates a key pair:

- a private key sk,
- a public key pk.

$(y, \pi) = Eval(sk, x)$
Deterministic algorithm that, given an arbitrary input x and the private key sk, produces:

- a value y, which is the pseudo-random output of the VRF,
- a proof π, attesting that y has been correctly computed from sk and x.

$true/false = Verify(pk, x, y, \pi)$
Deterministic algorithm that, given the public key pk, the input x, the output y, and the proof π, returns:

- $true$ if y is the unique valid VRF output for (pk, x),
- $false$ otherwise.

3.4 Use in the Protocol

In our protocol, the VRF is used to generate pseudo-random values in a verifiable way, ensuring that each node produces a unique output for each round.

The VRF input is a public value known to all participants, identified as the output y of the VDF of the previous block produced by the same miner, ensuring that the randomness is **bound** to the **current state of the chain** and cannot be manipulated retroactively.

From the execution of the VRF, each node obtains two distinct outputs:

- a pseudo-random value;
- a cryptographic proof that allows any observer to verify that the pseudo-random value was correctly generated by the node's private key, without revealing it.

Thanks to this mechanism, the protocol guarantees uniqueness per node and per round, preventing a single node from producing multiple valid outputs for the same input. In this way, both unpredictability and public verifiability are ensured—fundamental properties for the security and correctness of the distributed system.

3.5 VRF → Time T Mapping

Within the protocol, the pseudo-random value y generated by the VRF plays a fundamental role as a **mechanism for transforming randomness into a deterministic and verifiable waiting time**. The objective is to assign each node a specific time interval during which it may publish a block, without the possibility of manipulation or anticipation.

The process assumes that the selection defined by the protocol must satisfy two fundamental constraints: **unpredictability** of the block sequence and **fairness** among nodes. The VRF provides a pseudo-random number uniquely bound to the node's key and to the VDF output y of the previous block, ensuring that each output is publicly verifiable and unique per node.

To transform this output into real time, y is interpreted as a uniformly distributed number in the interval $[0,1]$ and used as input to the geometric distribution, which models the number of hash attempts in Bitcoin required to achieve success. In other words, geometric **distribution** allows simulation—deterministic yet pseudo-random—of the **moment when a node succeeds in generating a block**, taking into account its computational power and the round difficulty.

The conversion follows the **inverse transform method:** starting from y, the number of hash operations k required to reach success is computed as:

$$k = \frac{\ln(1-y)}{\ln(1-p)}, \text{ dove } p = \frac{1}{D}$$

Here, D represents the difficulty parameter of the round, and p the probability of success for a single attempt.

This transformation guarantees **verifiable determinism**: anyone who knows the node's public key and the hash of the previous block can reconstruct exactly the time T associated with the node, without dispute. At the same time, the randomness of the VRF prevents selective outcome choice: the node cannot influence either y or T, ensuring impartiality and security within the protocol.

3.6 Security Analysis

The adoption of the VRF for generating pseudo-random values and mapping them to waiting times introduces significant security advantages compared to traditional Proof-of-Work (PoW) mechanisms but still requires thorough analysis of potential threats.

Grinding attack

A grinding attack consists of a malicious node attempting to repeatedly compute the VRF over different or manipulable inputs in order to obtain a favorable output. Thanks to the deterministic and verifiable nature of the VRF, each output is unique per node and round and depends on already established public inputs (e.g., the VDF output y of the previous block). This drastically limits selective outcome choice: the node cannot “repeat” the computation over arbitrary inputs to obtain more favorable waiting times, reducing the effectiveness of grinding attacks compared to traditional PoW-based systems.

Sybil attacks

The VRF preserves pseudo-random and verifiable randomness even in the presence of multiple identities controlled by a single actor. However, as in any distributed system, a large number of false identities could theoretically increase an attacker’s probability of success in certain rounds. The combined use of unique public keys and verification of values generated by each node helps mitigate the risk, but complete protection against Sybil attacks also requires additional mechanisms, such as economic requirements or identity-bound participation limits.

4. Verifiable Delay Function

This chapter introduces Verifiable Delay Functions (VDFs) and their role within the proposed consensus protocol. After discussing the limitations of VRFs when used alone in the presence of real temporal constraints, the chapter defines the properties that make them suitable for imposing verifiable sequential delays. It illustrates the choice of an RSA-group-based VDF and its concrete use within the protocol, particularly in the block publication process. The chapter concludes with an analysis of the main security considerations, highlighting the differences between the role of VDFs and that of traditional Proof-of-Work mechanisms.

4.1 Why the VRF Is Not Sufficient

Verifiable Random Functions (VRFs) are extremely powerful tools in distributed protocols: they allow the generation of **deterministic, verifiable, and unique pseudo-random values** per node and per round, transforming public inputs into unpredictable outputs that can be checked by anyone who knows the node's public key. However, when their use is analyzed in temporal contexts, significant limitations emerge that compromise their effectiveness when used alone.

The main limitation of VRFs concerns the **temporal sequentiality** of actions. In a distributed protocol, it is not sufficient to know that a node "should" act at a certain moment: it is necessary that the action occurs after a **real and measurable delay**, in order to guarantee correctness and prevent manipulation.

The VRF allows computation of a waiting time T derived from its pseudo-random output, but this T is **virtual**. Nothing prevents a node from:

- **Precomputing multiple outputs over possible future inputs**, for example hashes of subsequent blocks, selecting the most advantageous ones
- **Ignoring the assigned time** and immediately participating in the next publication, publishing a block before the "simulated" time

This phenomenon is known as **simulated waiting**. Even if VRF values are verifiable and unique, the node is not physically bound to respect the derived times. In other words, the VRF establishes **when an action should occur**, but it cannot prevent its premature execution.

To understand the scope of this limitation, consider the parallel with Bitcoin. In the Bitcoin protocol, mining difficulty is calibrated so that the average time between the discovery of new blocks is approximately 10 minutes; this behavior is not a logical constraint, but a constraint arising from the nature of Proof of Work. To find a valid hash, each miner must perform an **enormous number of sequential hashing operations** that cannot be bypassed through parallelization, and only after doing so can it present a valid block. This mechanism induces a real, physical, and non-bypassable delay, distributing block publication over a stable and predictable time interval.

In our protocol, the objective is to **reproduce this controlled block timing behavior**, but without the energy cost and centralization typical of PoW. This is where VDFs come into play. A VDF is designed to impose a **real and sequential delay**, transforming a pseudo-random input into an effective waiting time that cannot be accelerated either through greater computational power or parallelization.

A VDF requires a sequential number of operations to be computed, meaning it **cannot be accelerated** either with more powerful hardware or with parallel computation. This property guarantees that the

execution time is physically constrained, transforming a pseudo-random value into a **measurable and effective delay** that cannot be manipulated by opportunistic nodes.

The VRF establishes who has the **right to act** and with what probability, while the VDF transforms this probability into a real and unavoidable delay, distributing node actions over a controlled interval. In this way, the protocol maintains a stable and **predictable rhythm**, guarantees fairness among nodes with different computational power, and prevents opportunistic strategies, without relying on enormous energy consumption. In other words, the "simulated" time of the VRF becomes **actual real time** thanks to the VDF.

4.2 Definition and Properties of VDFs

Verifiable Delay Functions (VDFs) are cryptographic functions designed to ensure that a computation requires a real delay, while still producing an output that can be efficiently verified. The main objective of VDFs is to transform waiting-time values into effective and unavoidable delays, allowing control over the temporal rhythm of node actions in a distributed protocol.

Three fundamental properties define VDFs and guarantee their security and effectiveness:

Sequentiality

The most important property of a VDF is that computing its output requires a minimum number of strictly sequential operations. This means the computation cannot be accelerated either by using more powerful hardware or by parallelizing operations. Sequentiality guarantees that execution time is physically constrained, preventing nodes from anticipating results and bypassing the intended delay.

Uniqueness

Each input of the VDF produces a single valid output. This property prevents the possibility of generating multiple results and choosing the most favorable one, ensuring fairness among nodes. Uniqueness is fundamental to maintaining protocol predictability and ensuring that each node has waited the predetermined time without the possibility of manipulation.

Efficient verification

Although computing the VDF requires real sequential time, its output can be quickly and publicly verified by any protocol participant. This feature allows all nodes to confirm that the delay has been respected without having to repeat the entire computation. Efficient verification is essential for system scalability and auditability, as it enables fairness and correctness to be checked without introducing significant computational costs.

4.3 Choice of the VDF

The concept of VDFs emerged in the years following the spread of cryptocurrencies such as Bitcoin, when researchers began asking whether it was possible to **separate time control from the energy consumption of Proof of Work**.

In Bitcoin, the delay between block discoveries naturally derives from PoW difficulty: each miner must perform an enormous number of sequential hashing operations, guaranteeing an average interval of about 10 minutes between blocks. However, this approach entails extremely high energy consumption and strong centralization toward nodes equipped with powerful hardware.

VDFs were conceived to replicate the main effect of PoW without the energy expenditure. The goal was to create a cryptographic function that produces a unique and verifiable pseudo-random output, while requiring a **minimum sequential time to be computed**, independently of hardware power or parallelization strategies.

RSA group–based VDFs

These functions exploit the difficulty of computing modular roots in an RSA group without knowing the secret factorization of the modulus. They are among the most studied and theoretically secure VDF constructions and guarantee strong sequentiality and uniqueness.

RSA group–based VDFs rely on the difficulty of computing modular roots without knowing the secret factorization of the modulus. In practice, an RSA modulus $N = pq$ is chosen, the product of two large prime numbers, and the VDF is defined as the sequential computation of modular exponentiations.

Given an input x, the VDF requires computing:

$y = x^{2^T} \bmod N$

where T is the number of sequential iterations. Each step depends on the previous one, making parallel execution impossible.

Technical definition

Let $N = pq$ be a securely generated RSA modulus, where p and q are large and secret primes, and let T be a parameter determining the number of sequential iterations. The VDF is defined as:

$$y = \mathrm{VDF}(x, N, T) = x^{2^T} mod\ N$$

where:

- $x \in \mathbb{Z}_N^*$ is the function input
- $T \in \mathbb{N}$ is the number of sequential iterations, determining the delay
- N is the shared RSA modulus, generated during a trusted setup phase
- y is the final output of the VDF iterations

The **proof** π allows any network node to quickly verify that y is correct without executing all 2^T sequential iterations.

The choice of parameter T determines the delay duration: larger values increase sequential computation time, making it impossible to bypass the temporal constraint.

The RSA modulus N must be securely generated during the initial trusted setup and shared among all network nodes.

4.4 Use of the VDF in the Protocol

In the proposed protocol, the **RSA VDF** plays a central role in translating the pseudo-random value generated by the VRF into a **real sequential delay**, ensuring that each node respects physical and verifiable temporal constraints before being able to publish a block.

The VDF parameter T determines the number of sequential iterations required to compute the output:

$$y = x^{2^T} mod\ N$$

Each iteration depends on the result of the previous one, making it impossible to reduce computation time through parallelization or more powerful hardware. In this way, T becomes the d**irect control of the real temporal delay**: the larger T is, the greater the sequential waiting time the node must respect.

The VDF as Proof of Actual Waiting

The sequential computation of the VDF produces, in addition to the output y, a cryptographic proof π demonstrating to anyone that the node has completed the required iterations and therefore has "spent" the prescribed time.

This proof is essential to prevent opportunistic strategies: a node cannot simply skip the computation, anticipate the result, and publish the block before others.

In other words, the VDF transforms the **"simulated" time of the VRF into a real, measurable, and unavoidable physical delay**, preserving protocol fairness and security.

Attaching the Proof to the Block

During the publication process, each node attaches to the block not only the VRF output, but also the **VDF output and the corresponding proof π**.

The complete sequence is therefore:

1. The node generates a pseudo-random value x through the VRF, verifiable by anyone.
2. This value becomes the input of the RSA VDF:

   $y = \text{VDF}_{RSA}(x, N, T)$.
3. The node computes the proof π demonstrating sequential execution.
4. The block is published attaching y and π.

Observing nodes can quickly verify the correctness of the delay **without** re-executing all iterations.

This mechanism guarantees that the rhythm of block publication is controlled, predictable, and unavoidable, emulating Bitcoin's temporal block distribution behavior without the energy expenditure of Proof of Work.

4.5 Example

Choose:

- an RSA modulus $N = pq$
- an input x
- a difficulty parameter T

Define:

$$y = x^{2^T} \text{mod}\ N$$

This operation is slow because it requires T sequential squarings:

$$x \to x^2 \to x^4 \to x^8 \to \cdots \to x^{2^T}$$

The verifier cannot recompute 2^{T} squarings. A proof π is needed such that:

- the proof is small
- verification is fast

Compute:

$$\ell = H(x \parallel y)$$

where H is a 256-bit hash.

Then define:

$$2^T = q\ell + r$$

with:

$$0 \leq r < \ell$$

Thus:

$q = \left\lfloor \frac{2^T}{\ell} \right\rfloor, r = 2^T \bmod N$

Define the proof:

$$\pi = x^q \bmod N = x^{\left\lfloor \frac{2^T}{\ell} \right\rfloor} mod\ N$$

The verifier checks:

$$\pi^\ell \cdot x^r \equiv y \quad (mod\ N)$$

Proof:

$$y = x^{2^T} \bmod N =$$

$$x^{q\ell+r}\ mod\ N =$$

$$x^{q\ell} * x^r\ mod\ N =$$

$$\pi^\ell \cdot x^r\ mod\ N$$

Parameters:

$$N = 77$$
$$x = 5$$
$$T = 4$$

Step 1: Compute y

$$y = 5^{2^4} = 5^{16} \bmod 77$$

Compute:

$$5^2 = 25$$
$$5^4 = 25^2 = 625 \, mod \, 77 = 9$$
$$5^8 = 9^2 = 81 \, mod \, 77 = 4$$
$$5^{16} = 4^2 = 16$$

$$y = 16$$

Step 2: Hash

Assume:

$$\ell = 7$$

Step 3: Division

$$2^T = 16$$
$$16 = 7 \cdot 2 + 2$$

Thus:

$$q = 2, r = 2$$

Step 4: Proof

$$\pi = x^q = 5^2 = 25 \bmod 77$$

Verification

The verifier checks:

$$\pi^\ell \cdot x^r \bmod 77$$

Compute:

$$25^2 = 625 \, mod \, 77 = 9$$
$$25^4 = 9^2 = 81 \, mod \, 77 = 4$$
$$25^7 = 25^4 \cdot 25^2 \cdot 25$$
$$= 4 \cdot 9 \cdot 25$$
$$= 900 \, mod \, 77 = 53$$

Now:

$$x^r = 5^2 = 25$$

Multiply:

$$53 \cdot 25 = 1325 \bmod 77 = 16$$

Result:

$$16 = y$$

Verification successful.

4.6 Security and Attacks

The introduction of a Verifiable Delay Function within the protocol does not primarily aim to replace the computational security of Proof of Work, but rather to impose a real, verifiable, and non-bypassable

temporal constraint. However, adopting a VDF introduces specific security considerations, particularly regarding specialized hardware and comparison with classical PoW-based consensus mechanisms.

Hardware Parallelism

One of the most relevant attacks in distributed protocols is the possibility that nodes equipped with more powerful hardware obtain a systematic advantage over other participants.

In traditional Proof of Work, this phenomenon is intrinsic to the model: the probability of success is directly proportional to available computational power, and hardware evolution has led to strong mining centralization.

VDFs, particularly those based on **groups of unknown order (RSA),** are explicitly designed to mitigate this risk. Evaluating the function requires a fixed number of **strictly sequential operations**, where each iteration depends on the previous result. This dependency eliminates effective parallelization: even a node equipped with extremely powerful hardware cannot significantly reduce execution time by distributing the computation across multiple units.

It is important to note that this does not imply perfect equality among all nodes. Marginal differences due to hardware or implementation optimizations still exist. However, such advantages are **linear and limited**, and do not scale exponentially as in PoW. The result is a system in which the delay imposed by the VDF remains substantially uniform and globally predictable.

From a protocol security perspective, the use of a VDF offers several advantages over traditional PoW.

First, the VDF **decouples temporal constraints from energy consumption**. In PoW, the average block production time is a direct consequence of the computational waste required to solve the cryptographic puzzle. In a VDF, time is an explicit protocol parameter, controlled by the number of sequential iterations T, without requiring continuous energy competition.

Second, the presence of a **cryptographic proof of waiting** allows any node to verify that the block producer has actually waited the prescribed time. In PoW, verification concerns only the validity of the final hash, not the temporal process that led to its discovery. With a VDF, the delay becomes an integral and verifiable part of the protocol state.

It is essential to distinguish the role played by a VDF from that of Proof of Work.

Proof of Work introduces a cost of an economic nature, in which participation and network attacks require a continuous and quantifiable investment in energy resources. This cost makes malicious behavior economically disincentivized and provides an entry barrier based on real physical resource consumption.

A VDF operates on a different level. Its purpose is not to impose a competitive cost, but to introduce **an objective, verifiable, and non-bypassable temporal constraint**. Through the intrinsic sequentiality of computation, the VDF guarantees that a given action can occur only after a real time interval has effectively elapsed, independently of computational power or available parallelism.

For this reason, it is naturally employed as a mechanism for timing and ordering actions, rather than as a tool of economic competition among participants.

In other terms, the VDF establishes when an event may legitimately occur, while determining who is authorized to perform a given action is entrusted to the Verifiable Random Function (VRF) + Geometric Distribution with Variable Probability at the moment the waiting time is extracted.

5. Proposed Consensus Protocol

This chapter presents the proposed consensus protocol, describing its architecture, operating principles, and operational rules. After an overview of the objectives and properties that the protocol aims to preserve with respect to traditional mechanisms, the role of its fundamental components is illustrated: VRF, variable-probability geometric distribution, and VDF, and the way they interact to verifiably determine the moment of block publication. The chapter then analyzes the complete operational flow of a miner, the block verification procedures, the management of difficulty adjustments and epochs, and finally the behavior of the system in the presence of collisions and forks.

5.1 Overview

Block publication in a traditional Proof of Work blockchain, such as Bitcoin, is driven by a mechanism of computational competition: miners repeatedly compute hashes until they find a value that satisfies a threshold determined by the difficulty, which in turn regulates the average frequency of block publication. This approach ensures that the creation of new blocks is unpredictable, randomly distributing the possibility of proposing blocks among miners, while at the same time preventing any entity from controlling the sequence of blocks without investing a proportional amount of computational power.

However, the main drawback of PoW lies in its **enormous energy consumption**. The security of the network is directly linked to the energy spent performing useless computations, namely hashes that do not produce valid blocks. Increasing computational power leads to a proportional increase in energy consumption to maintain a stable block publication frequency. This constraint not only generates a significant environmental impact but also limits the scalability and efficiency of the blockchain.

Our objective is to develop a consensus protocol that **preserves the fundamental properties of a blockchain**, such as:

- **Decentralization** – no central authority or trusted party can determine who publishes blocks.
- **Uniqueness and randomness of block proposal** – the selection of the miner that generates a block remains unpredictable.
- **Temporal distribution of blocks** – block publication occurs over controlled and predictable intervals, preventing spam or immediate publication by opportunistic miners.

At the same time, the new protocol aims to drastically **reduce energy consumption** by eliminating the need for repeated computationally useless calculations, while maintaining a timing mechanism **analogous to that of PoW**, in which the waiting time is random and distributed according to well-defined statistical properties.

To achieve this objective, the protocol integrates three main tools:

- **Verifiable Random Function (VRF)** – provides a cryptographically secure random number, unique per round, associated with the miner and verifiable by anyone.
- **Geometric Distribution (GD)** – maps the pseudo-random number generated by the VRF into a discrete number of iterations, representing the virtual time required before publishing a block.

- **Verifiable Delay Function (VDF)** – guarantees that the waiting time is physical and cannot be bypassed, by executing sequences of non-parallelizable operations and providing a verifiable proof of having waited.

This combination makes it possible to **replicate the temporal and random properties of PoW** without wasting energy, establishing a consensus mechanism that maintains security, fairness, and verifiability.

5.2 Components

Protocol startup: VRF input

A node participating in consensus begins by **determining the VRF input**, i.e., the source of randomness on which the rest of the process will depend.

- **Node in the first round:** if the node has never published a block, it uses the hash of the genesis block as input. This approach ensures that all nodes have a common, public, and non-manipulable initial base, avoiding any advantage derived from arbitrarily choosing the initial state.
- **Node with previous blocks:** if the node has already published at least one block, the VRF input becomes the VDF output y of the last block published by that node. This choice ensures that VRF extractions are sequential, unique, and non-reusable, establishing an internal node history that cannot be forged.

Formally, for each round, the node computes:

$$(y, \pi) \leftarrow \mathrm{VRF}_{sk}(x)$$

where x is the deterministic input (genesis or the VDF output y of the previous block), $u \in (0,1)$ is the pseudo-random uniform value, and π is the publicly verifiable proof. This extraction occurs **only once per round**, making u the only source of randomness for determining the exact publication time.

Figure 8: VRF scheme

In the proposed protocol, the VRF, illustrated in Figure 8, **replaces the continuous hash sampling of PoW** as a source of randomness. In Bitcoin, each hashing attempt implicitly generates a uniform random number; in our protocol, this number is generated **only once**, in a verifiable way.

The VRF output does not directly enable block publication but is used as a **cryptographic seed** to determine the waiting time (expressed as a number of VDF iterations) in a fair and non-manipulable manner.

We adopted a **simple, canonical, and verifiable rule**:

"Each miner extracts the VRF using as input the VDF output y of its last published block; if the miner has never published any block, it uses the hash of the genesis block."

This choice guarantees:

- absence of input grinding (the miner cannot choose arbitrarily)
- temporal continuity between one extraction and the next
- ex post verifiability by all nodes

The choice of using as VRF input the VDF output y of the previous block produced by the same miner satisfies several security and protocol-consistency requirements.

First, this value is **public and universally verifiable**, since it is an integral part of the block data and therefore stored on the blockchain. This guarantees that all participants have the same input and can deterministically verify the correctness of the VRF output, eliminating ambiguities.

Second, the output of a VDF has the property of being **unpredictable** before completion of the previous VDF evaluation. Since computing the VDF requires a minimum sequential, non-parallelizable time, no node can know the value y in advance before the block is produced and published. This prevents a miner from precomputing multiple possible VRF outcomes or strategically selecting among favorable inputs.

Another relevant aspect is the retroactive **non-manipulability** of the value used as input. Once the block containing the VDF output has been published and accepted in the chain, modifying its contents would require rewriting that block and all subsequent ones, an operation that, in the protocol's security model, is computationally prohibitive. Consequently, the VRF input is strongly bound to the historical state of the blockchain.

Translation of the VRF value into a number of iterations: variable-probability geometric distribution (GD-VP)

The next step is to determine when the node will be authorized to propose the block, translating the pseudo-random value u into a number of iterations k, as shown in Figure 9.

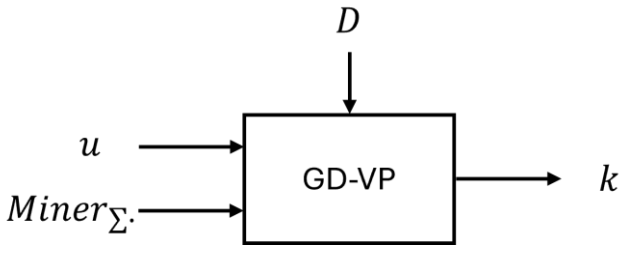

Figure 9: GD-VP scheme

In Proof of Work, the waiting time before publishing a block follows an **exponential distribution**, since each hashing attempt is independent and has a constant probability of success. The probability of success in a single attempt is inversely proportional to the difficulty D:

$$p = \frac{1}{D}$$

The number of attempts before obtaining a valid hash therefore follows a **geometric distribution**. This model ensures that:

- the average time between blocks is controllable via the difficulty;
- the sequence of hashes provides unpredictable randomness, protecting the network from attacks.

However, repeatedly computing hashes is costly and unnecessary if one can mathematically map the number of required attempts without executing useless hashes.

The main complication is that **difficulty changes over time**, both guaranteeing a constant average block interval and responding to variations in node participation.

To correctly model the behavior, we introduce the GD-VP. The network is divided into epochs $E_0, E_1, \dots, E_m$, each with constant difficulty D_i. For each epoch i, the node maintains the number of iterations already executed n_i.

The total number of iterations k required to publish a block, considering all previous epochs, is calculated with the formula already illustrated in the chapter dedicated to GD-VP:

$$k = \sum_{i=0}^{m-1} n_i + \frac{1}{\ln\left(1-\frac{1}{D_m}\right)}\left(\ln(1-u) - \sum_{i=0}^{m-1} n_i \ln\left(1-\frac{1}{D_i}\right)\right)$$

This formula guarantees continuity in waiting time, independence from the current epoch, and correct adaptation to difficulty changes.

In the protocol, difficulty is not expressed as a threshold on a hash, but as a parameter p that regulates the expected value of k:

- high difficulty → small p → large k → slower blocks
- low difficulty → large p → small k → more frequent blocks

This mechanism allows the publication rate to be adjusted without introducing wasted computational work.

In practice, **GD-VP translates a pseudo-random number into a relative waiting time, consistent with the history of the node and the network**, without requiring superfluous computations.

The VRF and the geometric distribution determine **how long a miner should wait**, but they do not enforce that this time is actually respected.

Proof of elapsed time: VDF

A node could theoretically compute k without waiting. To solve this problem, the protocol uses a VDF.

In practice, each iteration $1,2, \dots, T$ of GD-VP is **mapped onto a step of the VDF**, as depicted in Figure 10, creating a real and non-bypassable delay, replicating the waiting-time function of PoW.

Figure 10: VDF scheme

If difficulty changes during execution, the node can split the execution into batches, as illustrate the Figure 11, each with its own VDF proof, guaranteeing auditability and temporal continuity.

Upon completion of k iterations:

- The node constructs the block, containing:
  - Header and transactions
  - VRF value u and proof π
  - VDF proof
- The node proposes the block to the network.
- The nodes verify:
  - Correctness of the VRF value
  - Validity of the VDF proofs
  - Consistency with previous epochs and current difficulty

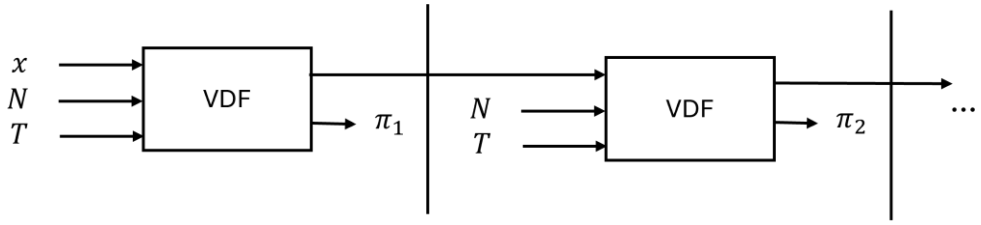

Figure 11: VDF verifiable chain scheme

If everything is correct, the block is inserted into the blockchain according to the **Longest Chain Rule**. In case of a fork, nodes update their state and continue executing the VDF starting from the new chain head.

5.3 Operational flow of the consensus protocol

This section describes in detail the complete life cycle of a miner, from selecting the initial input to the publication and verification of a block. The protocol is structured as a sequence of **waiting rounds**, each of which may potentially culminate in the production of a block.

In the context of the proposed protocol, a **round** is defined as the time interval between the extraction of a random value via VRF and the publication of the associated block, after executing the required VDF iterations.

A round is measured by the **number of sequential VDF iterations**. This notion replaces the concept of "hashing attempts" in traditional Proof of Work.

Each round is probabilistically independent of the previous one, but critically dependent on the previous one with respect to the VRF input.

Each miner locally maintains the following state elements:

- sk, pk: key pair for the VRF
- $last\_block$: the last block published by the miner (if any)
- $longest\_chain$: the chain currently considered longest (Longest Chain Rule)
- $epoch$: the current epoch, determined by the position in the blockchain
- D: current difficulty (or equivalently parameter p of the geometric distribution)

There is no global state or external coordination: each miner operates completely autonomously.

The choice of the VRF input is **deterministic, canonical, and verifiable**, and is defined as follows.

VRF extraction

If the miner has never published a block, the VRF input is:

$$x = hash(header(genesis))$$

If the miner has already published at least one block, the input is:

$$x = y\_VDF(last\_block)$$

Given input x, the miner computes:

$$(y, \pi) = VRF(sk, x)$$

The output y is interpreted as a uniform value in the space $[0,1)$ through deterministic normalization. This extraction occurs **only once** per round and is irrevocable.

Mapping to a geometric variable

The uniform value $u \in (0,1)$ obtained from the VRF is transformed into an integer k through the GD-VP formula:

$$k = \sum_{i=0}^{m-1} n_i + \frac{1}{\ln\left(1-\frac{1}{D_m}\right)}\left(\ln(1-u) - \sum_{i=0}^{m-1} n_i \ln\left(1-\frac{1}{D_i}\right)\right)$$

This value k constitutes the cryptographic iterations of the round.

Start and execution of the VDF

Once k has been determined, the miner starts executing the VDF. A single VDF iteration is considered atomic and sequential, similarly to a single hash in PoW.

$$y_0 = input\_vdf$$
$$y_i = f(y_{i-1})\ for\ i = 1..n$$

where f is the sequential function defined by the chosen VDF.

To allow incremental verifiability and epoch management, execution can be divided into batches of iterations, each accompanied by a proof.

The miner maintains:

- the number of completed iterations
- the current intermediate output
- the proofs associated with completed batches

Epoch management and variable difficulty

The protocol assumes that difficulty changes only at epoch boundaries.

When a miner receives a block that ends an epoch, it must:

1. **Interrupt the VDF**
2. Store the proof of the iterations already executed
3. Recalculate the number of remaining iterations using the variable-difficulty geometric distribution formula
4. Resume the VDF from the intermediate point

This behavior is essential to preserve the property that already completed work is never invalidated, analogously to PoW.

Full start procedure is summarized by Figure 12.

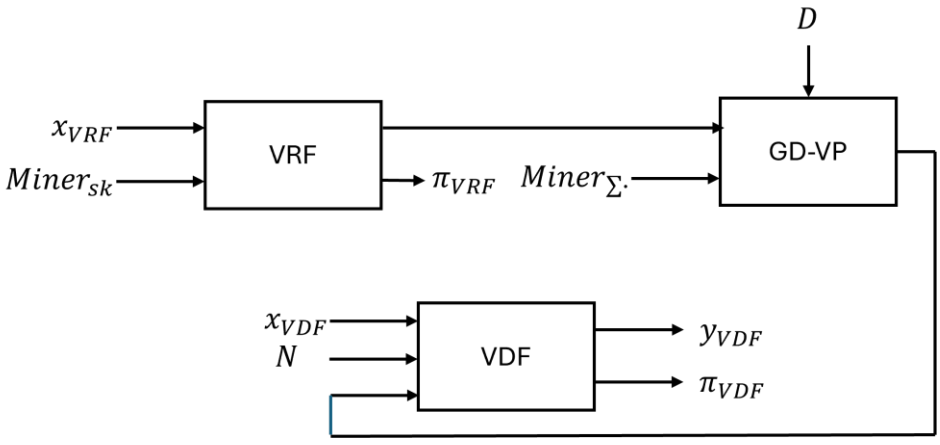

Figure 12: Complete miner round extraction

Block publication

In the proposed protocol, VDF execution and waiting-time calculation via geometric distribution are conceptually organized into publication rounds, each culminating, if successfully completed, in the publication of a single block. Proper delimitation of these rounds is essential to ensure that the temporal mechanism faithfully replicates the behavior of Proof of Work, preventing accumulation of undue "temporal credit" between successive publications.

The variable-difficulty geometric distribution uses two distinct summations: one over the number of iterations executed in each epoch, denoted n_i, and a weighted summation involving terms of the type $n_i \cdot \ln(\cdot)$, necessary to correctly incorporate difficulty variations across epochs. In this context, n_i represents exclusively the number of VDF iterations executed by the miner within a single publication round during epoch i.

A crucial aspect of the protocol is that these summations must not span distinct publication rounds. When a miner successfully publishes a valid block, the entire accumulated state related to the previous waiting-time calculation must be considered completed and permanently consumed. From that moment, the miner starts a completely independent new round, with a new VRF extraction and a new value of k, and the geometric-distribution summations must be reset.

Therefore, for a new extraction following block publication, the GD mapping formula becomes:

$$k = \frac{1}{\ln\left(1-\frac{1}{D_m}\right)} \ln(1-u)$$

The new round begins immediately after block publication and follows a procedure identical to the initial one: the miner extracts a new VRF output using as input the VDF output y of the block just published, maps that output through the geometric distribution to obtain a new value of k, and starts executing a new VDF independent of the previous one.

When the total number of iterations k has been completed, the miner can publish the block.

The block must contain:

- hash of the previous block or genesis
- VRF output y
- VRF proof π
- total number of iterations k
- final VDF output
- proofs of all executed VDF batches

Block verification

The node first verifies that the block is well formed, i.e., that it contains all mandatory fields required by the protocol, including at least:

- the VDF output of the declared previous block
- block header, including epoch, height, and consensus metadata

- VRF output y and proof π
- complete description of VDF batches

The block is immediately discarded if:

- one or more mandatory fields are missing
- formats do not comply with specifications
- internal references are inconsistent

VRF Verification

The node then proceeds to verify the VRF associated with the block. Let:

- x be the VRF input, which must correspond to the VDF output of the last block previously published by the same miner, or to the genesis if the miner has never published
- y be the VRF output included in the block
- π the cryptographic proof
- pk the miner's public key

The node verifies that:

$\text{Verify}(pk, x, y, \pi) = \text{true}$

Otherwise, the block is invalid and is permanently discarded.

Deterministic Recalculation of the Value k

Starting from the VRF output y, the node deterministically recomputes the value k, namely the total number of VDF iterations required for the block publication round.

The re-computation uses:

- the mapping function defined by the protocol
- the geometric distribution formula with variable difficulty
- the difficulty parameters valid in the epochs crossed by the round

The node verifies that:

- k is a positive integer
- $k \geq 1$
- the value implicitly declared by the miner through the VDF batches exactly matches the recomputed k

Any discrepancy results in block invalidation.

Verification of VDF Proofs for All Batches

The node then verifies the VDF execution. Since execution may span multiple epochs, the VDF is divided into **verifiable batches**, each associated with:

- a declared number of iterations
- an intermediate output
- a proof of correctness that can be efficiently verified

For each batch, the node verifies that:

- the output correctly derives from the input of the previous batch
- the associated VDF proof is valid
- the number of iterations in the batch is consistent with the difficulty of the corresponding epoch

The node also verifies that:

- the concatenation of batches produces a continuous and uninterrupted sequence
- the sum of iterations of all batches is exactly equal to k
- there are no overlapping, missing, or redundant batches

If even a single VDF proof fails, the block is invalid.

Consistency with Epochs and Difficulty

The node verifies that the block complies with the epoch transition rules:

- the number of blocks per epoch is respected
- VDF batches are correctly segmented at epoch boundaries
- the recalculation of the residual time is consistent with the variable-difficulty formula

In particular, the node verifies that the miner interrupted the VDF only at verifiable batch boundaries and correctly updated the residual-time computation at each epoch transition.

Insertion of the Block into the Local View

If all previous checks are successful, the node inserts the block into its local blockchain structure:

- the block is linked to its predecessor
- any forks are updated
- height and depth information are updated

The block is now considered **locally validated**, though not necessarily part of the main chain.

Application of the Longest Chain Rule (LCR)

Finally, the node applies the Longest Chain Rule:

- it compares the length (in terms of number of blocks) of all known valid chains
- it selects as the main chain the longest one

If the arrival of the new block modifies the main chain:

- the node updates its head
- blocks previously considered main become part of side forks
- local miners currently executing a VDF do not interrupt or restart their round, in accordance with the established rules

Epoch Transition

In the proposed protocol, network difficulty is not static but evolves discretely over time, according to a temporal subdivision into epochs analogous to that adopted by Bitcoin. An epoch is defined as a fixed interval of 2016 blocks on the chain considered longest (Longest Chain Rule, LCR). Once this threshold is exceeded, the network determines a new global difficulty, which will apply to all blocks belonging to the subsequent epoch.

The epoch transition is a particularly significant event, as it directly affects the geometric distribution used by miners to determine the number of VDF iterations to execute before publishing a block.

Unlike traditional Proof of Work, in which difficulty influences the probability of success of each independent attempt, in the proposed protocol difficulty modifies the mapping between the VRF output and the total waiting time, expressed as the number of VDF iterations.

At the beginning of a round, a miner has already sampled a VRF and determined an initial value k, representing the number of VDF iterations required before publishing the block. This value is computed using the geometric distribution formula parameterized by the difficulty in force at the time of sampling. However, since the real time required to complete k iterations may span one or more epoch transitions, the protocol must rigorously specify how to handle difficulty variation while a VDF is in execution.

When a miner observes that the chain it considers LCR has reached an epoch boundary — i.e., a block whose height is a multiple of 2016 has been published — the protocol requires the miner to interrupt the VDF at the first verifiable point. Since the VDF cannot be interrupted arbitrarily at the level of individual atomic iterations, but only at the level of verifiable batches, the interruption occurs upon completion of the current batch. At that point, the miner has an intermediate VDF state consisting of the output y and proof π, as well as the total number of iterations T completed so far.

This intermediate state does not represent block publication, but it is conceptually binding: the work performed up to that point is neither lost nor reset and must be accounted for in the continuation of execution. The miner does not resample the VRF and does not restart from zero; instead, it updates the residual waiting-time computation using the geometric distribution formula with variable difficulty, as depicted in the Figure 13.

Specifically, the new difficulty of the subsequent epoch modifies the parameter of the geometric distribution and thus the relationship between the sampled random value and the expected number of iterations. The miner then recomputes a new effective value of k, taking into account both the updated difficulty and the iterations already performed. Formally, the new residual time is not obtained through

a new random sampling, but through a deterministic transformation incorporating the cumulative iterations, thereby preserving the fundamental Proof of Work property that previously performed work progressively increases the probability of success.

This mechanism may repeat multiple times: if the overall value of k is large enough to span multiple epochs, the miner must interrupt the VDF and update the residual-time computation at each subsequent epoch transition. At each step, the miner accumulates an ordered sequence of VDF batches, each associated with a specific epoch and difficulty.

From the perspective of verifiers, this behavior is fully verifiable ex post. When the miner finally publishes its block, it must include all information necessary to reconstruct the entire history of VDF execution. In particular, for each batch, the block must contain the output y, the proof π, the number of iterations performed, and the reference to the epoch (and therefore the difficulty) under which that batch was computed. A verifier, knowing the historical sequence of blockchain difficulties, can independently and deterministically verify each batch and confirm that the total number of executed iterations is consistent with the value of k derived from the VRF and the geometric distribution with variable difficulty.

It is important to emphasize that all these operations are performed exclusively with reference to the chain the miner considers longest according to the LCR. In the presence of forks, the miner completely ignores the difficulties and epoch transitions of alternative chains and recalculates residual time solely based on the chain it considers canonical. This choice is consistent with traditional Proof of Work behavior, where work is always oriented toward a single local view of the chain, and global convergence emerges only ex post through the longest-chain rule.

Finally, the protocol mandates that interrupting the VDF at an epoch transition is not an optional or strategic choice for the miner, but a normative requirement. The miner cannot choose to ignore the epoch transition or continue computing under obsolete difficulty parameters, as this would render the final block unverifiable and therefore invalid. In this way, the protocol ensures that difficulty adjustment produces an immediate and global effect on the block publication rate, without introducing artificial resets, without wasting computational work, and without creating advantages for opportunistic behavior.

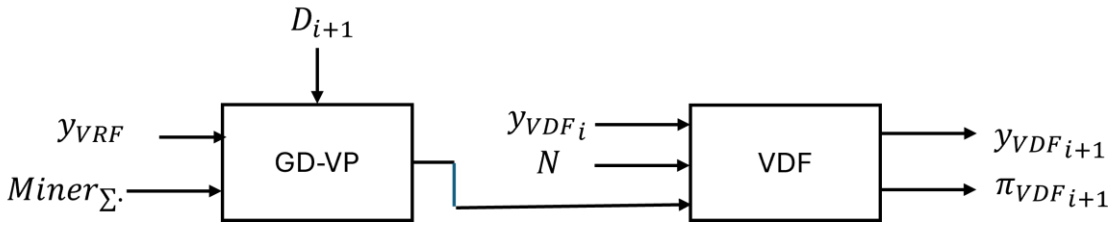

Figure 13: End-epoch miner recalculation

5.4 Collisions and Forks

In the real operation of the protocol, block publication does not occur in a serial and orderly manner, but in an asynchronous distributed environment where network latency, miner concurrency, and the absence of global coordination make temporal **collisions and blockchain forks inevitable**. The protocol does not attempt to eliminate them, as they are a natural consequence of any permissionless decentralized system; rather, it explicitly incorporates them into its security and convergence model.

A collision occurs when two or more miners complete their sequence of VDF iterations within a sufficiently close time interval such that they have not yet received the other's block. In this scenario, each miner publishes a valid block referencing the same parent, giving rise to competing branches of the blockchain. From the protocol's perspective, such blocks are indistinguishable in terms of cryptographic validity: both satisfy the constraints imposed by the VRF, the geometric distribution, and the VDF, and must therefore be accepted by nodes that receive them first.

Each node locally maintains all valid chains it is aware of but selects one as the operational reference by applying the Longest Chain Rule (LCR). The longest chain, defined as the one containing the greatest number of valid blocks, is considered the canonical chain.

If a new block makes an alternative branch longer than the chain on which a miner was working, behavior remains coherent and deterministic. The miner updates its view of the blockchain but does not **resample the VRF and does not modify the already determined value of k**. As established by the protocol, VRF sampling is completely independent of the chain head and is never repeated. Consequently, the miner continues executing the VDF associated with the current round, simply realigning verification and anchoring of the future block to the new canonical chain.

6. Prototype and Experimental Validation

This chapter presents the prototype developed for the experimental validation of the protocol and for studying the dynamics of consensus systems based on probabilistic competition. After clarifying the objectives and assumptions of the model, it describes the architecture of the simulator, with particular attention to the representation of virtual time, event management, and the local behavior of miners. The chapter then illustrates the main implementation choices and the mechanisms used to reproducibly model phenomena such as block publication, network propagation, fork formation, and difficulty adjustment, showing how the prototype enables experimental observation and analysis of consensus behavior.

6.1 Prototype Objectives

The objective of the developed prototype is to provide an experimental tool for analyzing and understanding the dynamic behavior of consensus protocols, with particular focus on the temporal and probabilistic aspects that emerge from the concurrent interaction of multiple nodes in a distributed environment.

Unlike a real blockchain implementation, the prototype is not designed to validate the cryptographic security of the protocol nor to operate in the presence of malicious nodes or Byzantine behavior. Its primary purpose is instead to study how consensus rules, competition among miners, and difficulty adjustment mechanisms influence the global evolution of the blockchain over time. In this sense, the prototype assumes an honest participation model in which all nodes correctly follow the specified protocol.

The simulator is conceived as a discrete-event (39) execution environment capable of explicitly representing time, randomness, and node interaction. This approach makes it possible to observe phenomena that, although fundamental to system operation, are difficult to capture through static or purely mathematical descriptions. These include competition for block publication, the formation of temporary forks due to network latency, and the role of difficulty adjustment as a feedback mechanism stabilizing block production rate.

The Blockchain Execution Environment (BEE) is therefore designed as a flexible and observable tool, suitable both for experimentation and educational purposes. It enables exploration of scenarios that are difficult to reproduce in a real blockchain, such as sudden variations in global computational power, changes in network latency, or modifications to difficulty parameters, allowing their impact on consensus behavior to be analyzed.

The BEE architecture operates under clearly defined assumptions. All miners are considered honest and follow the protocol. Attacks such as selfish mining or Sybil attacks are not modeled. No cryptographic verification mechanisms are implemented, and no VDF is used. These choices are deliberate: since the BEE is a simulated environment, verifying correct node behavior is unnecessary, and removing such components significantly simplifies the simulation, focusing on consensus dynamics.

In summary, the prototype is not intended to replace a real blockchain, but to provide an executable model that makes the fundamental protocol dynamics visible and understandable, supporting both experimental analysis and learning of the designed consensus mechanism.

6.2 Architecture

The BEE architecture is designed to capture the fundamental dynamics of Proof of Work consensus: miner competition, probabilistic block publication, network latency, forks, and difficulty adjustment—while avoiding the computational cost and complexity of real hashing-based simulation.

Event Queue

The BEE adopts a **Discrete-Event Simulation** model in which system evolution is governed by an ordered sequence of events, each associated with a virtual timestamp. All events are stored in a global **Event Queue** implemented as a Calendar Queue (40), ensuring efficient extraction of the event with the minimum deadline. Global virtual time is updated iteratively by assuming, at each step, the deadline for the next event.

Architecturally, the BEE uses only two event types, which are sufficient to model the full lifecycle of a PoW blockchain:

- **Block publication events**, representing completion of mining by a miner
- **Block reception events**, representing arrival of a block at another miner after network delay

This separation clearly distinguishes block creation from block observation by other nodes, making network latency effects on fork formation and consensus convergence explicit.

Each event includes:

- A deadline *t* (virtual timestamp)
- A type
- A reference to the destination miner
- Associated data (for reception events)

The Event Queue imposes a total temporal order. In case of equal deadlines, FIFO ordering is applied, ensuring deterministic behavior given the same random seed.

A miner may publish a block at the same virtual instant it receives a competing block. If publication is processed first, the published block may later become part of a fork, faithfully reproducing real blockchain competition dynamics.

Event Processing Engine

The system is orchestrated by an Event Processing Engine (EPE), which continuously extracts and handles events. The EPE implements no consensus logic; its sole role is to advance virtual time, identify the destination miner, and delegate processing. This reflects the decentralized nature of PoW consensus.

The core architectural principle of the BEE is **separation between logical time and physical computation**. In real systems, blockchain progress depends on billions of hash operations; in the BEE, time is explicitly modeled as an abstract variable—Virtual Time (*t*)—which advances only at significant events. This preserves PoW statistical properties while enabling fast, controllable simulation.

The EPE:

- Extracts the event with minimum deadline
- Updates global virtual time
- Delegates processing to the associated Miner Execution Engine

Miner Execution Engines

Node behavior is modeled through a collection of **Miner Execution Engines (MEE)**. Each MEE represents an independent miner, equipped with a complete local state and a partial view of the blockchain. Miners operate asynchronously, reacting to events that concern them and autonomously updating their internal state.

Each MEE maintains a local state that includes the entire block tree observed up to that point.

The miner maintains:

- its local view of the blockchain
- the branch considered best
- the block from which it started computing the iterations to be performed
- the current difficulty
- its computational power
- the two accumulators required for generating publication deadlines via GD and GD-VP
- the set of orphan blocks, if necessary

The miner stores a reference to the root of the blockchain, to the branch considered best (implemented as the longest branch), and to the set of orphan blocks. This representation makes it possible to model multiple forks and chain reorganizations without imposing a global order a priori.

In the BEE, each miner can have **only one active publication deadline at a time**, representing the expected time for discovering the next valid block. This reflects the fact that, in real Proof of Work, a miner continuously works on a single mining attempt until success or until an external event (e.g., the reception of a new block) changes the context.

Latency Matrix

The BEE network model is deliberately simple yet expressive. Latency between two miners is modeled as potentially **asymmetric** through a global matrix, and latencies may **vary dynamically during the simulation**. In particular, latency is periodically sampled every configurable number of published blocks. This makes it possible to simulate time-varying network conditions such as congestion, infrastructure improvements, or temporary degradation events.

A relevant architectural aspect is that latencies **may change dynamically** during the simulation. At configurable intervals (expressed as number of published blocks), the latency matrix is updated according to predefined functions. This enables the simulation of non-stationary network conditions such as congestion, infrastructure improvements, or transient events.

When a miner publishes a block, the BEE generates a reception event for each other miner, whose deadline is computed as:

$$t_{\text{recv}} = t_{\text{pub}} + L_{i,j}$$

where $L_{i,j}$ is the sampled latency value for the considered pair of miners, as shown in Figure 14.

In this way, block propagation occurs asynchronously and non-deterministically, reproducing the real conditions of a peer-to-peer network.

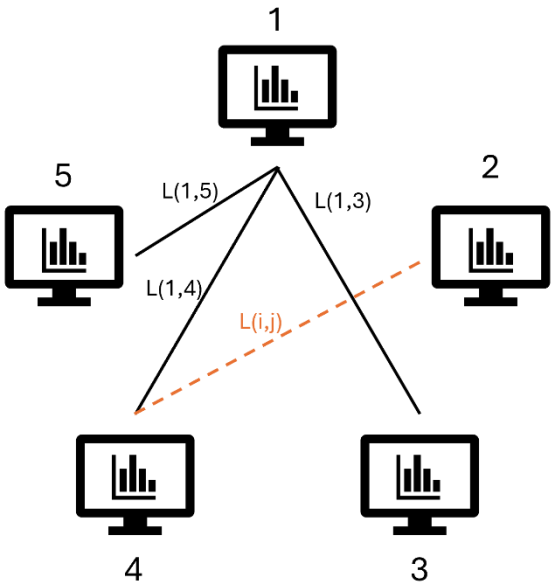

Figure 14: Latency graph

Blockchain Representation

Each miner maintains a **local view of the blockchain** represented as a tree of blocks rooted at the genesis block. Blocks are modeled as nodes of an explicit tree, as Figure 15 shows.

Each block stores both structural information (parent–children relationships) and contextual information (difficulty, computational power of the miner that produced it, estimated global hash rate). These fields allow the simulator to:

- reconstruct fork topology
- compute the best branch according to the longest chain rule
- associate each block with the difficulty context in which it was generated

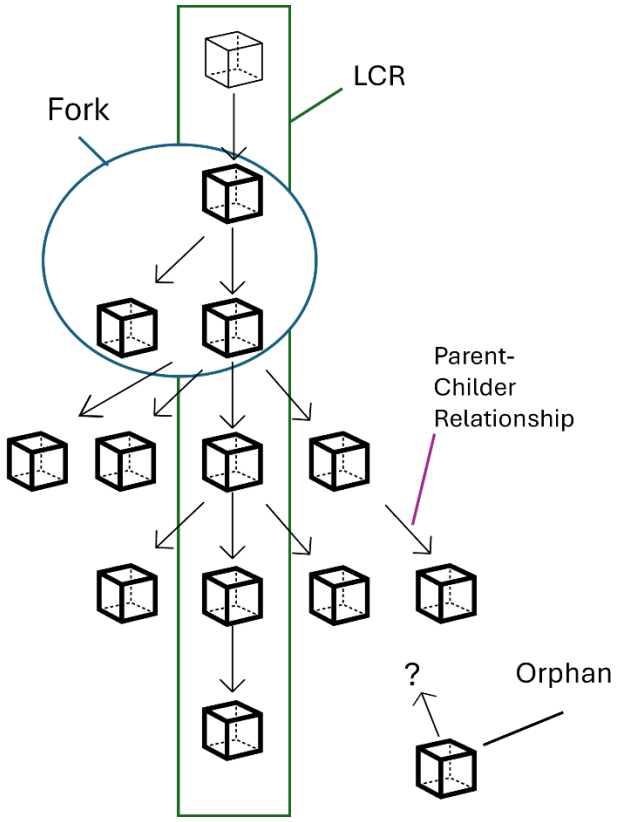

Figure 15: Model nomenclature

Blocks that cannot be immediately connected to the main tree (e.g., because the parent is not yet known) are temporarily stored as **orphans**. This structure makes it possible to correctly handle out-of-order block reception, a common phenomenon in the presence of network latency.

GD-VP and Difficulty Adjustment

The probabilistic core of the system is the generation of publication deadlines. Under constant difficulty conditions, the waiting time before discovering a block follows an exponential distribution; in the BEE, this behavior is simulated by sampling a random variable from the **geometric distribution**, representing the number of attempts required before success. The deadline is then obtained by converting that number into virtual time through the miner's hash rate.

However, difficulty is not constant over time. As in Bitcoin, the BEE implements a **dynamic difficulty adjustment mechanism** based on comparing the observed average MIBT. Difficulty is updated every configurable number of blocks according to the formula:

$$D_{i+1} = D_i \cdot \frac{TIBT}{MIBT_i}$$

Difficulty is global and unique: two miners on the same branch and in the same epoch cannot operate with different difficulty levels. However, due to network latency, miners may learn about a difficulty change at slightly different times. In the BEE, each miner adapts to the new difficulty when it receives the **first block of the subsequent epoch**, consistently with real-world behavior.

This aspect introduces a fundamental complexity in deadline generation: a miner may start mining under a certain difficulty and need to continue under a different difficulty before publishing. To correctly handle this scenario, the BEE uses **GD-VP**.

Each miner maintains two local GD-VP accumulators representing work already performed under previous difficulties. When a difficulty change occurs, the miner:

- updates the accumulators according to the elapsed virtual time
- reschedules the publication deadline using the recursive GD-VP formula

This mechanism guarantees that previously performed work is not invalidated by a difficulty change, faithfully replicating real Proof of Work behavior.

It is important to emphasize that, after a publication, the accumulators are reset and generation of the next deadline again uses the standard geometric distribution. GD-VP comes into play exclusively when difficulty changes before publication.

Additional Model Assumptions

A further objective of the prototype is to conceptually separate consensus dynamics from cryptographic enforcement aspects. In the real protocol, primitives such as VRF and VDF play an essential role in ensuring unpredictability, fairness, and verifiability. However, in a controlled simulation environment, such properties—particularly those of the VDF—are not necessary because:

- miners cannot behave maliciously
- the event-driven structure of the simulator is not compatible with the operational mode of a VDF

The VRF, instead, is partially integrated into the simulator: the relevant functionality is the generation of a uniformly distributed random number. On the other hand, VRF verification is omitted, since in a simulated environment it is considered unnecessary to verify the validity of the extraction.

This choice preserves the statistical and temporal properties of Proof of Work without introducing the computational cost and implementation complexity of cryptographic primitives.

Finally, the architecture supports dynamic variations of miners' hash rates. The computational power of each node can be modified during the simulation according to input parameters, inducing variations in global hash rate and consequently triggering the difficulty adjustment mechanism. This makes the BEE particularly suitable for studying and teaching the interactions among hash power, difficulty, latency, and forks.

6.3 Implementation

The implementation of the BEE faithfully reflects the architecture described in the previous section, translating each conceptual component into well-defined data structures and operational procedures. The simulator is developed in C, a choice that allows explicit control over memory management, execution model, and event ordering—fundamental aspects for ensuring reproducibility of simulations.

The entire implementation is organized around three central elements:

- Management of virtual time and events

- Local miner behavior
- probabilistic generation of publication deadlines and the difficulty adjusting mechanism

Virtual Time Management and Event Processing Engine

Virtual time is represented as a global counter that, at each instant, assumes the value of the deadline of the current event. There is no continuous time progression: virtual time evolves exclusively when events extracted from the Event Queue are processed.

The Event Queue is implemented as a list ordered by increasing deadline. Each insertion preserves temporal ordering, ensuring that the event at the head always corresponds to the minimum deadline. In the case of events with identical deadlines, FIFO ordering preserves the internal causality of the simulation.

The main simulation loop:

- extracts the event at the head of the queue
- implicitly updates virtual time to the deadline value of the event
- invokes the event processing procedure

This logic realizes the EPE, which acts solely as an orchestrator and contains no consensus or mining logic.

Event Types and Execution Flow

The implementation includes only two types of events:

- EVENT_TYPE_PUBLISH, associated with the publication of a new block by a miner
- EVENT_TYPE_RECEIVE, associated with the reception of a block published by another miner

Event processing is centralized in a single dispatch function that distinguishes behavior based on event type. This design choice makes the execution flow explicit and easily traceable, facilitating both debugging and experimental analysis.

System Initialization

At the start of the simulation, the following are created:

- a set of miners, each with a local copy of the genesis block
- an initially empty Event Queue
- an initial publication event for each miner

The initial difficulty is computed as a function of the number of miners, the initial computational power, and the target inter-block time. This choice ensures that, from the very first iterations, the system operates in a regime consistent with the target inter-block time.

Each miner generates its first publication deadline using the classical geometric distribution, since no work has yet been accumulated under previous difficulties.

Implementation of the Miner Execution Engine

Each miner is represented by a data structure that fully encapsulates its local state. In particular, the miner maintains:

- a local blockchain view in the form of a tree;
- a reference to the branch considered best;
- the current difficulty;
- computational power;
- the state required for deadline generation via GD and GD-VP.

A fundamental invariant of the implementation is that **each miner has at most one active publication event in the Event Queue**. This constraint is strictly enforced: whenever a deadline is updated or invalidated, the corresponding event is removed and reinserted with the new timestamp.

Generation of Publication Deadlines

The generation of publication deadlines constitutes the probabilistic core of the simulator. It is based on the GD-VP formula.

When a miner (re)starts the mining process, a uniform random value $u \in [0,1]$ is sampled. This value remains associated with the miner until block publication or a difficulty change. The number of hashing attempts required for success is then computed by transforming u according to the appropriate distribution.

Under constant difficulty conditions, the formula reduces to classical geometric distribution. The publication time is obtained by scaling the number of attempts by the miner's computational power, thus converting the number of hashes into virtual time.

```
restart_miner(miner, actual_time, vrf_input):
    miner.epoch_product_accumulator = 1
    miner.epoch_sum_accumulator = 0
    miner.restart_hash_time = actual_time
    miner.epoch_start_vt = actual_time


    vrf_pi = vrf_prove(miner.vrf_sk, vrf_input)

    vrf_y = vrf_output(miner.vrf_sk, vrf_input)

    miner.random = vrf_y

    new_deadline = actual_time + flat_to_geom(miner.random, miner.difficulty,
miner.epoch_sum_accumulator, miner.epoch_product_accumulator)

Note: flat_to_geom corresponds to GD_VP formula.
```

Implementation of GD-VP and Work Accumulation

When difficulty changes before a miner has published a block, the implementation uses the GD-VP. To this end, each miner maintains two accumulators:

- an additive accumulator, representing the number of attempts already performed
- a multiplicative accumulator, incorporating the cumulative effect of previous difficulties

At the moment of a difficulty change:

1. the miner computes the number of attempts performed since the last update as a function of the elapsed virtual time
2. updates the accumulators
3. recalculates the publication deadline using the closed-form GD-VP formula

If the new deadline were earlier than the current virtual time, it is forced to an immediately subsequent value, ensuring the temporal causality of the simulation.

```
update_difficulty(miner, actual_time):
    miner.epoch_product_accumulator *= powl(miner.current_q, actual_time -
miner.epoch_start_vt)
    miner.epoch_sum_accumulator += actual_time - miner.epoch_start_vt

    new_deadline = miner.restart_hash_time + flat_to_geom(miner.random, miner.difficulty,
miner.epoch_sum_accumulator, miner.epoch_product_accumulator)

    miner.epoch_start_vt = actual_time

Note: flat_to_geom corresponds to GD_VP formula.
```

After a publication, both accumulators are reset and generation of the new deadline once again occurs through the standard geometric distribution. This distinction faithfully reflects the theoretical model: GD-VP is required exclusively when a single mining process spans multiple difficulty epochs.

Block Publication

Processing a publication event represents the moment when a miner successfully completes its mining process and makes a new block public. From the simulation perspective, this event marks the conclusion of a probabilistic extraction based on the geometric distribution (or GD-VP) and the beginning of a new competitive phase among miners.

When the Event Processing Engine extracts a **PUBLISH**-type event, the miner associated with the event executes a well-defined sequence of operations reflecting the behavior of a miner in a real Proof of Work blockchain.

1. Update of Global State and Computation of Global Hash Rate

As a first operation, the simulator increments the global counter of published blocks. This value does not play a direct role in consensus but is used to:

- track the temporal evolution of the blockchain
- support the difficulty adjustment mechanism
- enable controlled variations in miners' computational power

2. Creation of the New Block

The miner then creates a new block, which is linked as a direct child of the block belonging to the branch currently considered best (longest branch) in its local blockchain view. This choice implements the longest-chain rule typical of Proof of Work systems.

The created block includes, in addition to structural references (identifier, parent block, timestamp), a set of metadata relevant for experimental analysis:

- the identifier of the miner that produced the block
- the current difficulty
- the miner's computational power
- the global hash rate at the time of publication
- the total number of blocks published up to that moment
- the length of the branch on which the block is inserted

In this way, each block captures both the local and global state of the system at the moment of its creation.

3. Update of the Best Branch

After block creation, the miner updates its blockchain view and verifies whether the new block extends the best branch. Since the block is always constructed on the locally longest branch, the update generally succeeds. However, the procedure is implemented generically and correctly handles more complex cases, for example when the local tree structure includes preexisting forks.

Updating the best branch entails incrementing the branch length and possibly updating the internal data structures that track the miner's local blockchain.

4. Block Propagation and Network Latency Modeling

Once published, the block is propagated to all other miners through the creation of **RECEIVE** events. For each recipient miner, the simulator:

- determines the network latency associated with the pair of miners
- schedules a reception event with a deadline equal to the current virtual time plus the latency

Latency may vary dynamically during the simulation according to configurable parameters. This choice enables realistic simulation of asynchronous block propagation and the possibility of forks caused by communication delays.

It is important to note that block publication does not imply any form of global synchronization: each miner will receive the block at a potentially different time, and the order of reception may influence the formation of temporary forks.

5. Dynamic Variation of Computational Power

At fixed intervals, expressed as a fraction of the difficulty adjustment period, the simulator modifies miners' computational power. This mechanism is triggered in correspondence with certain publication events and introduces controlled variations of the global hash rate.

From a modeling perspective, this choice makes it possible to study the effect of new computational power entering the network and to observe the response of the difficulty adjustment mechanism.

6. Restart of the Mining Process and Generation of the New Deadline

After publishing a block, the miner must start a new mining process. To this end:

- the accumulators used for GD-VP are reset
- a new uniform random value is sampled
- a new publication deadline is computed using the classical geometric distribution

This behavior faithfully reflects the theoretical model: after publication, previous work no longer has any effect, and the mining process restarts from zero under the current difficulty.

7. Difficulty Update and Deadline Rescheduling

If publication of the block causes the best branch to reach a height that is a multiple of the adjustment interval, the miner activates the difficulty update procedure. The new difficulty is computed by comparing the observed average inter-block time with the target time.

If difficulty is updated immediately after publication, the newly computed deadline is recalculated using GD-VP, preserving work already performed within the new difficulty epoch.

8. Insertion of the New Publication Event

Finally, if the length of the best branch has not reached the maximum limit specified by the simulation, the miner inserts a new publication event into the Event Queue at the computed deadline. This ensures that the miner immediately participates in the next block production competition.

Processing of Reception Events

Processing a **RECEIVE** event models asynchronous block propagation within the network and represents the primary interaction point among miners. Unlike publication events, which depend exclusively on the miner's local state, reception events introduce external information and are the primary cause of temporary forks and blockchain reorganizations.

When a miner receives a block, the simulator executes a sequence of operations faithfully reflecting the behavior of an honest Proof of Work node.

1. Parent Block Lookup and Orphan Management

The first operation consists of attempting to locate the parent block within the miner's local tree. To optimize the most frequent case, the simulator initially checks whether the parent of the received block

coincides with the block currently at the head of the local best branch. Otherwise, a search is performed throughout the entire tree starting from the root.

If the parent block is not present in the miner's local view, the received block cannot be immediately attached to the blockchain. In this case, the block is inserted into the **orphan** structure along with the identifier of the missing parent. This choice reflects the reality of blockchain systems, where blocks may arrive out of order due to network latency.

Whenever a new block is added to the local tree, the miner checks whether there exist orphan blocks that can now be attached. This recursive check allows automatic resolution of out-of-order reception sequences and correct reconstruction of complete blockchain branches as soon as all necessary blocks become available.

2. Insertion of the Block into the Local Tree

If the parent block is known, the miner creates a local copy of the received block and links it as a child of the identified parent. The block is constructed by faithfully replicating the metadata of the original block, including:

- identifier of the miner that produced it
- publication timestamp
- current difficulty
- computational power of the producing miner
- global hash rate at the time of publication
- branch height at the time of creation

This duplication is necessary because each miner maintains its own independent local blockchain data structure, as occurs in real systems.

3. Update of the Best Branch (Longest Chain Rule)

After block insertion, the miner recalculates the best branch by applying the longest chain rule. If the new block extends a branch that becomes longer than the previous best branch, the reference to the longest branch is updated and the branch length is incremented.

The code explicitly distinguishes the case in which updating the best branch is actually necessary from the case in which the received block does not modify the main branch selection. This distinction is essential to avoid redundant updates of difficulty and publication deadlines.

4. Detection of Difficulty Change

If the new best branch reaches a height that is a multiple of the adjustment interval, the miner verifies whether difficulty needs to be updated. In the simulator's adopted model, difficulty is global and shared: two miners cannot operate with different difficulties on the same branch and in the same epoch. Consequently, the miner synchronizes to the difficulty associated with the received block marking the beginning of the new epoch.

This behavior reflects an intentional simplification of the real protocol: the difficulty change is implicitly learned through reception of the first block of the new epoch, without introducing an explicit coordination mechanism.

5. Rescheduling the Publication Deadline

When difficulty is updated following block reception, the miner must reschedule its publication deadline. This step is crucial because the probability of success of mining attempts changes with difficulty.

The simulator uses GD-VP to recalculate the deadline while preserving work already performed within the previous epoch. To this end:

- the accumulators representing the number of attempts already "consumed" and the cumulative product of failure probabilities are updated
- a new deadline consistent with the new difficulty is computed
- the previously scheduled publication event is removed from the Event Queue and reinserted with the new deadline

This mechanism guarantees that a difficulty change does not invalidate work already performed by the miner, replicating a fundamental property of real Proof of Work.

6. Interaction Between Reception and Publication and Fork Generation

A particularly relevant aspect of the model concerns the order of event processing. The Event Queue is ordered by deadline, and, in the case of identical deadlines, a FIFO policy is applied. Consequently, it is possible for a miner to:

- process a publication event
- publish a new block
- immediately afterward process a reception event related to a competing block

In this scenario, the received block is attached to a different branch, generating a temporary fork. This behavior naturally emerges from the combination of network latency, asynchrony, and FIFO event ordering, without requiring any special logic.

The possibility that a miner publishes a block while temporarily "ignoring" a valid block already in transit is a fundamental characteristic of Proof of Work systems and constitutes the statistical basis of fork formation and miner competition.

```
process_event(event):
    if event.type == PUBLISH:
        parent = miner.longest_branch
        block = create new block and attach to parent
        update miner.longest_branch
        foreach m in miners:
            calculate_latency(miner, m)
            set RECEIVE event in latency for m
        ! miner.longest_branch.lenght % epoch_lenght:
            update difficulty
        deadline = restart_miner(miner)
        schedule new PUBLISH event at deadline time

    if event.type == RECEIVE:
        block = received block
        parent = block.parent
        if parent is in chain:
            attach block to blockchain
            check orphans set
            update longest_branch
            longest_branch.lenght % epoch_lenght:
                get block.difficulty
                update deadline
                update PUBLISH event at deadline time
        else:
            add block to orphans
```

Difficulty Adjustment

Difficulty is updated every configurable number of blocks according to a proportional control formula that compares the observed average inter-block time with the target time. The computation is performed using timestamps of blocks belonging to the best branch.

Difficulty is global and unique for all miners. However, due to network latency, adoption of the new difficulty occurs asynchronously: each miner updates its difficulty when it receives the first block belonging to the new epoch.

```
calculate_new_difficulty(miner, reference_interblock_time):
    end_block = miner.longest_branch
    start_block = miner.start_epoch

    measured_interblock_time = (end_block.t – start_block.t) / REGULATION_INTERVAL

    new_difficulty =  miner.difficulty * reference_interblock_time
/ measured_interblock_time
```

Variation of Computational Power

The implementation allows dynamic modification of miners' power during the simulation. At configurable intervals, each miner's power is increased or decreased by a random amount. This

mechanism makes it possible to simulate variations in global hash rate and to observe the system's response through the difficulty adjustment mechanism.

Implementation Assumptions

The implementation explicitly reflects the assumptions of the model:

- all miners are honest
- attacks or strategic behaviors are not modeled
- no cryptographic verification is performed
- the VDF is not used

These choices make it possible to isolate the temporal and probabilistic dynamics of Proof of Work, making the simulator suitable for both experimental analysis and educational use.

6.4 Experiments

The purpose of the experimental evaluation is to validate the probabilistic model introduced in the previous sections, with emphasis on the behavior of publication deadlines under both **constant** and **variable Difficulty**.

The experiments are designed to verify three core properties of the proposed framework:

1. that under constant Difficulty the inter-block time (IBT) follows the **expected exponential behavior**
2. that under variable global hash rate (GHR), Difficulty Regulation **stabilizes** the average **IBT**
3. that **deadline updates** occur immediately at **epoch transitions**, restoring the **IBT**

Rather than focusing on implementation details, this section concentrates on the observable macroscopic properties of the blockchain process and their consistency with the theoretical model.

A fundamental observable in any blockchain protocol is the **Inter-Block Time (IBT)**, defined as the elapsed time between two consecutive blocks appended to the longest branch.

The IBT is a primary validation metric because it reveals whether the probabilistic model correctly reproduces expected dynamics by looking at the **stochastic process** governing block production.

Under constant Difficulty and constant GHR, block production is modeled as a memoryless process. Therefore, IBT is expected to follow an exponential distribution with mean equal to the target block interval (600 seconds in the reference configuration).

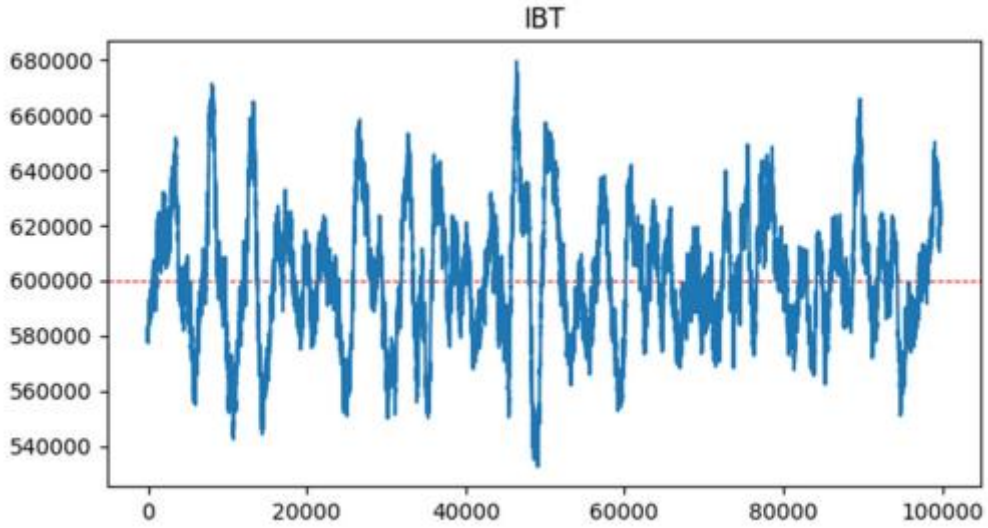

Figure 16: Measured IBT in 100000 blocks

The key observations are:

- IBTs fluctuate significantly around the nominal value
- “short” and “long” intervals both occur
- the average converges to approximately 600 seconds, as illustrated in Figure 16.

This variability is not an anomaly but a necessary consequence of the memoryless property of the geometric/exponential process. The absence of periodicity or deterministic spacing confirms that the probabilistic mechanism is correctly implemented.

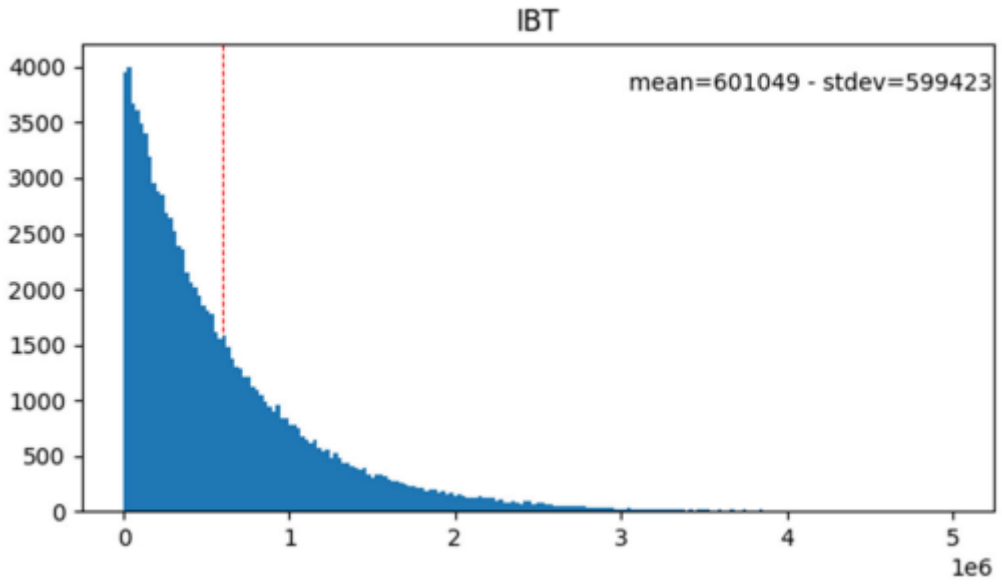

Figure 17: IBT Distribution

The histogram, Figure 17, exhibits the characteristic exponential decay:

- high frequency of short intervals
- progressively lower frequency of long intervals

This confirms that deadline computation based on GD produces the expected exponential inter-arrival distribution. The agreement between theoretical expectation and empirical histogram validates the correctness of the constant-Difficulty model.

The second experiment evaluates the behavior of the system under **evolving network power**.

In permissionless environments, the number of miners and the total GHR are not constant. An increase in GHR, if not compensated by **Difficulty adjustment**, would reduce the average IBT proportionally. This would destabilize the protocol by accelerating block production.

The experiment therefore simulates a progressive increase in the number of miners and, consequently, in total GHR.

The primary observable in this scenario is the joint evolution of:

- the **number of miners** (or GHR),
- the **Difficulty** parameter.

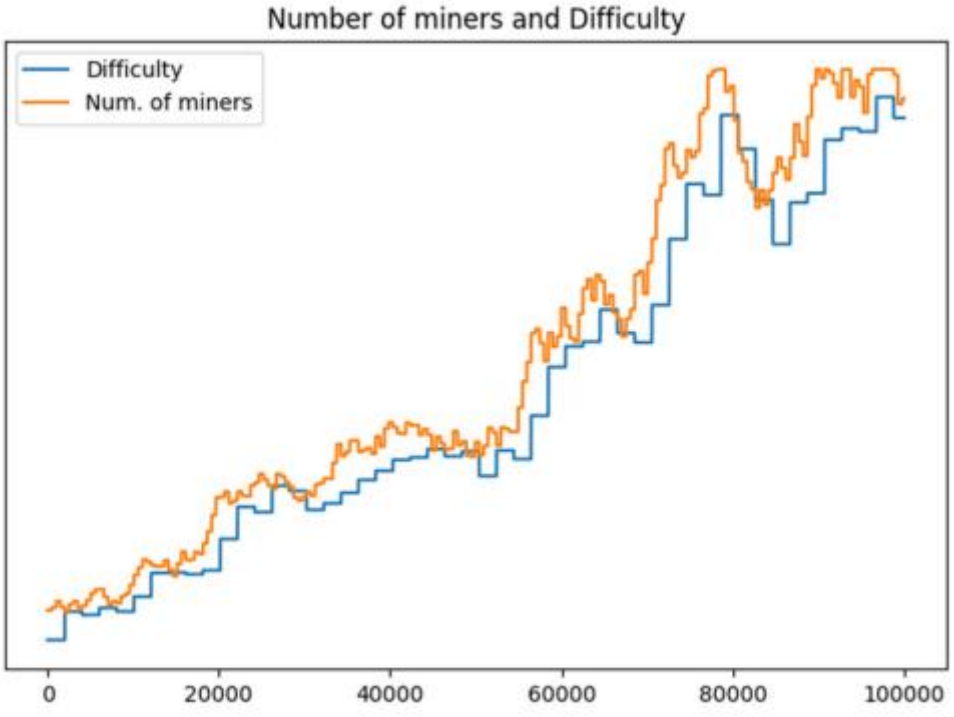

Figure 18: Comparison between # of miner and Difficulty (Resized)

The Figure 18 shows:

- a monotonic **increase** in the **number of miners** (or GHR);
- a stepwise **increase** in **Difficulty**.

The stepwise structure reflects epoch-based regulation: Difficulty is updated after fixed-length sequences of blocks. Each step corresponds to one epoch of constant Difficulty.

The crucial observation is that Difficulty reacts to changes in network power. As GHR increases, subsequent epochs exhibit higher Difficulty values.

This behavior validates the regulation mechanism:

- Difficulty **tracks long-term changes** in network power
- the system compensates for increased computational capacity
- block production rate remains **stable** over time.

The overlapping representation of miner growth and Difficulty evolution clearly illustrates the feedback mechanism: Difficulty is not static but dynamically adapts to network conditions.

The third experiment focuses on the **local behavior of IBT at epoch boundaries**, where Difficulty Regulation occurs.

While the previous experiment demonstrated global stability, this analysis investigates short-term dynamics.

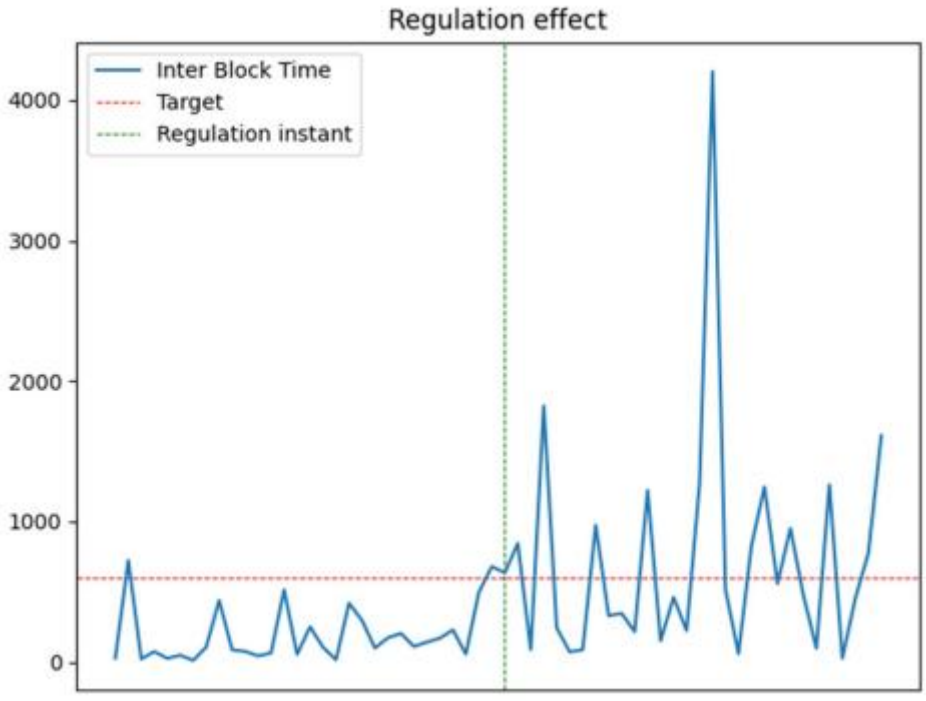

Figure 19: Regulation Effect in epoch change

Two distinct phases are visible in Figure 19:

1. **Pre-regulation phase**

   As GHR increases within an epoch, Difficulty remains constant until the epoch boundary. Consequently, the average IBT progressively decreases below the nominal 600 seconds.

2. **Post-regulation phase**

   At the epoch boundary, Difficulty is updated. Immediately after the update, IBT values return to approximately around 600 seconds.

This behavior is particularly significant. The fact that IBT is temporarily below target before regulation is expected: **increased GHR accelerates block production.**

The fact that IBT is immediately restored after regulation demonstrates that deadline recalculation based on GD-VP is effective.

There is no prolonged transient phase. The **rescheduling mechanism adjusts publication deadlines instantly**, restoring the nominal rate without oscillations or overshoot.

This observation validates two essential properties:

- **Correctness of Difficulty Regulation**, which ensures long-term rate stability
- **Efficiency of deadline update computation**, which ensures immediate short-term stabilization

The experimental results confirm that the proposed probabilistic framework satisfies the fundamental requirements of a blockchain consensus mechanism:

- Under constant Difficulty, IBT follows an exponential distribution with the correct mean
- Under increasing network power, Difficulty adapts stepwise to maintain a stable average block interval
- At epoch transitions, IBT is immediately restored to its nominal value
- The distributional properties of IBT remain consistent with the theoretical model

The analysis of IBT is central because it encapsulates the combined effects of:

- probabilistic block selection
- global hash rate evolution
- Difficulty Regulation
- deadline rescheduling

By observing IBT and its distribution, we validate both the statistical foundation and the dynamic stability of the protocol.

7. Resistance to Sybil Attacks

This chapter presents a discussion of Sybil attacks: in open distributed systems, where access is not regulated by a central authority and cryptographic key generation is free, identity does not constitute a scarce resource. In the proposed protocol, this issue takes a specific form: each registered identity can start an autonomous VDF instance and independently participate in the block selection process. The following discussion examines the problem in its entirety and motivates the introduction of mitigation mechanisms based on economic costs.

7.1 The Identity Problem

In open distributed systems, where participation is not constrained by a central authority, an **actor** can generate an **arbitrary number of cryptographic identities**. This type of behavior is known as a **Sybil attack** and represents one of the main threats to permissionless consensus protocols.

In the absence of mitigation mechanisms, a participant could artificially increase its influence by registering many identities and participating in the protocol with each of them. In protocols where the probability of selection depends on the number of participating nodes, this leads to a significant distortion of probabilities and compromises system fairness.

In our system, each identity can start an independent VDF instance. Since different executions can be performed in parallel on distinct hardware, an actor possessing multiple computational units can execute multiple VDFs simultaneously.

This implies that:

- the VRF prevents a single identity from producing multiple valid outputs for the same round
- the VDF prevents acceleration of a single computation
- however, it does not prevent an actor from increasing the number of attempts by executing multiple VDFs in parallel through registration of multiple key pairs

Consequently, the probability of success of a participant is proportional to the number of VDF instances it can execute simultaneously.

It is important to observe that this behavior does not depend on the number of identities per se, but on the available hardware resources. Creating identities without computational resources confers no advantage.

Let:

- n be the total number of active VDF instances in the system
- k be the number of instances controlled by an actor

If block producer selection is uniform with respect to valid outputs, the probability that the actor produces the next block is approximately:

$$P = \frac{k}{n}$$

This relationship shows that the determining parameter is not the number of identities, but the number of sequential computations effectively sustained.

The model therefore coincides with that of resource-based protocols: influence is proportional to the share of owned resources.

The problem of Sybil attacks, in the context of the proposed protocol, cannot be addressed by eliminating the possibility of creating multiple identities, since the system is designed to operate in an open and permissionless environment.

The real problem is that **the protocol is unable to distinguish between genuine plurality of participants and artificial plurality of identities controlled by a single actor**. This indistinguishability does not strictly compromise the security of the selection mechanism, but it may significantly affect system properties such as fairness, decentralization, and resilience.

This property has an important consequence: **the apparent distribution of participants may not reflect the actual distribution of control.** A system that appears populated by many nodes could be dominated by a few actors, each controlling many identities.

The problem becomes particularly relevant when considering properties that implicitly depend on participant independence. Many distributed protocols assume that nodes act in a statistically independent manner or that the probability of collusion among nodes is limited. If multiple identities belong to the same actor, these assumptions are no longer valid.

From a fair perspective, an actor controlling many identities may increase its overall probability of being selected more rapidly than a single participant with equivalent resources organized differently. Even if each identity must execute an independent VDF, fragmentation of resources across multiple identities may influence dynamics such as block propagation, latency management, or the probability of winning close competitions.

From a resilience perspective, the presence of many identities controlled by a single actor may make the system more vulnerable to coordinated behavior. For example, an actor could influence network topology, manipulate information dissemination, or amplify certain fork effects simply by exploiting the number of nodes under its control.

It is important to emphasize that this does not necessarily imply a violation of consensus security: if the protocol is correctly designed, an actor cannot produce valid blocks without respecting the rules; however, concentration of control may reduce system robustness.

In light of the preceding considerations, the protocol proposes a combined approach to mitigate the risk of Sybil attacks. The underlying idea is that creation and maintenance of an active identity must entail a real cost, in economic terms.

7.2 Economic Costs of Participation

In order to mitigate the effects of Sybil attacks, the protocol introduces economic costs associated with the **creation** and **maintenance** of identities. The objective of this mechanism is to ensure that influence in the protocol reflects a **proportional investment** of resources.

In Proof-of-Work systems such as **Bitcoin**, this resource is computational power. Influence in the consensus process is proportional to the fraction of global hash rate controlled by a participant. The

acquisition and operation of mining hardware require a substantial **economic investment**, which naturally limits identity multiplication.

The proposed protocol follows the same fundamental principle: **influence must be proportional to invested resources**. However, instead of relying on continuous computational expenditure, the protocol introduces explicit economic costs defined by the consensus rules.

This approach makes the cost structure transparent and controllable, while preserving the essential property that participation requires a non-negligible commitment of resources.

Each active identity corresponds to a single VDF instance executed during the epoch.

The introduction of economic costs modifies this behavior.

Let

- C_r be the **registration cost**,
- C_e be the **participation cost per epoch**,
- T the **number of epochs** during which the identity remains active.

The total cost associated with a single identity is:

$$C_{total} = C_r + TC_e.$$

For an actor controlling k identities, the total cost becomes:

$$C_{actor} = k(C_r + TC_e).$$

This linear relationship ensures that influence in the protocol cannot grow without a proportional increase in sustained costs.

The objective of this model is not to enforce a specific economic equilibrium, but rather to guarantee a structural property: increasing influence in the system must require increasing resource expenditure.

This property transforms the Sybil attack from an identity problem into a resource allocation problem.

An adversary may still control many identities, but only by sustaining a proportional cost. Therefore, large-scale Sybil attacks become economically constrained rather than technically trivial.

In Proof-of-Work systems, costs arise indirectly from hardware acquisition and energy consumption. These costs depend on external technological and economic conditions and are only partially controlled by the protocol. In our protocol, participation costs are defined by the consensus rules themselves. This allows the system to maintain a stable cost structure independently of external factors such as hardware efficiency or energy prices.

Another important aspect concerns the temporal distribution of costs. Proof-of-Work systems require continuous resource consumption. Mining hardware must operate constantly in order to remain competitive. This produces a continuous flow of energy expenditure.

The proposed protocol instead separates costs into two components:

- a fixed cost associated with identity creation
- a variable cost associated with continued participation

This separation reflects two distinct roles.

- The registration cost limits the creation of identities
- The operational cost limits their persistence over time

Together, these mechanisms ensure that inactive or strategically created identities cannot remain in the system indefinitely without sustained investment.

From a protocol design perspective, this structure provides a direct analogue to the cost model observed in Proof-of-Work systems.

Proof-of-Work miners must sustain:

- an initial investment in hardware
- a continuous investment in electrical energy

This analogy allows the protocol to reproduce one of the fundamental security properties of resource-based consensus: influence requires sustained investment. It is important to emphasize that the objective of this costs is to ensure that participation reflects a real commitment of resources. Under these conditions, the protocol preserves the essential security property that control of the system cannot be obtained without proportional investment.

7.3 Monetary Model

In addition to the structural cost model introduced above, the protocol defines an internal currency used to regulate participation and maintain economic balance within the system.

Participation in the protocol requires the acquisition of protocol tokens. These tokens must be spent when a **miner registers a new identity** and when it participates in successive epochs. Importantly, the tokens used for registration are **permanently destroyed** by the protocol.

This mechanism introduces a controlled **token burning process** associated with the creation of identities.

Let:

- B_r be the amount of tokens burned during identity registration
- B_e be the amount of tokens burned per epoch of participation
- T be the number of epochs in which the identity participates

The total number of tokens removed from circulation by a single identity is therefore

$$B_{total} = B_r + TB_e$$

For an actor controlling k identities, the total number of burned tokens becomes

$$B_{actor} = k(B_r + TB_e)$$

This mechanism directly ties system participation to a **permanent reduction of circulating supply**.

Opposing this burning mechanism, the protocol introduces a controlled process of **token creation** associated with successful block publication.

When a miner publishes a valid block, the protocol generates a reward consisting of newly minted tokens.

Let:

- R be the reward associated with a block publication
- N_b be the number of blocks produced during a given time interval

The total number of tokens created in that interval is

$$M = N_b R$$

Token creation therefore occurs only as a consequence of **useful work performed within the protocol**, namely the production of blocks.

This design introduces an important economic property. Entering the system requires the **irreversible destruction of tokens**, which creates a direct cost for the creation of identities.

At the same time, block rewards **redistribute** newly minted tokens to active miners. This ensures that honest participants can recover the costs associated with participation through successful block production.

To summarize:

1. New miners must acquire tokens.
2. These tokens are **destroyed** when identities are created and maintained.
3. Tokens are **reintroduced** into circulation through block rewards.

This mechanism performs two functions simultaneously:

- it **enforces economic costs** on identity creation,
- it **maintains incentives** for continued participation.

7.4 Permissionless but not Registrationless

Participation models represent a fundamental design dimension of blockchain protocols. Distributed ledger systems are commonly classified as either *permissioned* or *permissionless*, depending on whether participation is controlled by a central authority. However, this distinction alone is not sufficient to accurately describe the protocol proposed in this thesis. A more precise characterization requires introducing a second conceptual distinction, namely the difference between **permissionless** and **registrationless** systems.

The protocol presented in this work is **permissionless**, since no authority controls access to the system, but it is not **registrationless**, because participation requires an explicit registration procedure. This distinction reflects structural requirements of the protocol and is essential for understanding its operation and its relationship with existing blockchain systems.

A system is **registrationless** when nodes can participate in the protocol immediately, **without performing any preliminary registration procedure**. In such systems there is no persistent state

describing the set of participants, and nodes are **not required to establish a long-term identity** before participating.

Proof-of-Work systems approximate this model. In Bitcoin, for example, a miner may generate a private key and **immediately begin mining blocks**. The protocol does not maintain a registry of active miners, and participation does not require any explicit enrollment procedure. Miners effectively appear in the system only when they successfully publish a valid block. If a miner stops participating, the protocol does not retain any information about that miner's existence. From the point of view of identities, the system is largely stateless.

The protocol proposed in this thesis cannot operate in a registrationless manner. This limitation does not arise from administrative constraints but from structural requirements of the protocol.

Each participant must maintain a persistent identity associated with:

- a cryptographic key pair
- VRF computations
- VDF executions
- reward attribution
- participation costs

These elements require a stable association between identities and protocol state. In particular, the protocol must be able to associate each block producer with a specific identity in order to:

- validate protocol behavior
- assign rewards
- apply economic mechanisms

Without an explicit registration mechanism, these operations would not be reliably implementable.

Permissioned systems require explicit approval before a participant is allowed to join. This model is typical of enterprise blockchain platforms such as Hyperledger Fabric (41), where nodes are admitted by a governing authority and identities are explicitly managed.

The protocol proposed in this thesis follows a fundamentally different approach. Although participation requires registration, the process is governed entirely by protocol rules and does not involve discretionary decisions.

In particular:

- registrations cannot be arbitrarily rejected
- no authority evaluates participants
- no centralized entity assigns permissions

Any participant that follows the protocol rules can register and participate. The system therefore remains permissionless despite requiring registration.

Registration is also required in Proof-of-Stake systems.

Platforms such as Ethereum (28) and Cardano (29) require validators to register their public keys and maintain persistent identities. Participation typically involves locking a certain amount of stake and maintaining a continuously active node.

In Proof-of-Stake systems, registration directly determines the influence of participants, since the probability of being selected depends on the amount of stake associated with each identity.

In the protocol proposed in this thesis, registration serves a different purpose. The influence of a participant is not determined by registration itself but by the computational resources devoted to VDF execution.

The choice of permissionless but non-registrationless architecture reflects a deliberate design trade-off.

The protocol maintains:

- open participation
- absence of centralized control
- decentralized operation

At the same time, it introduces:

- persistent identities
- structured participation
- economic mechanisms

This approach allows the protocol to preserve the essential properties of permissionless blockchain systems while enabling mechanisms that improve stability and resistance to attacks.

8. Conclusions and Future Perspectives

This thesis investigated the problem of designing a distributed consensus protocol capable of operating in open and permissionless environments while reducing the structural inefficiencies associated with Proof-of-Work systems. The work was motivated by the observation that Bitcoin introduced a fundamentally new paradigm in distributed systems: **the possibility of achieving agreement among mutually untrusted participants without relying on a predefined validator set**, explicit coordination mechanisms, or trusted authorities.

Bitcoin's consensus architecture demonstrated that a decentralized network can spontaneously converge toward a shared ordering of events through the statistical behavior of many independent participants. The protocol achieves this by combining cryptographic primitives, probabilistic processes, and economic incentives into a single mechanism. However, this mechanism relies on large-scale computational competition, in which miners repeatedly perform hash computations in order to discover a valid block. As a consequence, the security of the system becomes intrinsically linked to **continuous resource consumption**, producing an ever-growing demand for specialized hardware and electrical energy.

The central question addressed in this thesis was whether the fundamental properties that make permissionless consensus possible could be preserved while reducing the dependence on large-scale computational expenditure. In particular, the work explored the hypothesis that the essential contribution of Proof-of-Work lies not in the specific hashing process itself, but in the way it implicitly regulates the timing and ordering of actions in an open network.

From this perspective, Proof-of-Work can be interpreted as a probabilistic mechanism that determines **when nodes are allowed to publish blocks**, ensuring that such events occur unpredictably and without coordination among participants. Each miner repeatedly performs a random experiment whose outcome determines whether it may publish a block. When a miner succeeds, the resulting block becomes a candidate for extending the blockchain, and the network collectively converges toward a consistent history through the longest-chain rule.

The thesis therefore explored an alternative design in which these elements are separated and implemented through specialized cryptographic primitives. By explicitly **modeling randomness generation, waiting time extraction, and time verification**, it becomes possible to reproduce the stochastic dynamics that characterize Proof-of-Work systems without requiring miners to continuously compete through repeated hash computations.

The first component introduced in the proposed protocol is a mechanism for **generating unpredictable yet verifiable randomness**. In open distributed environments, randomness plays a crucial role because it prevents participants from deterministically predicting or manipulating their chances of producing a block. Without such unpredictability, nodes could strategically schedule their actions or concentrate influence in ways that would compromise fairness and decentralization.

To address this requirement, the protocol adopts Verifiable Random Functions. A VRF allows each participant to generate a pseudo-random value that is deterministic with respect to a private key and a publicly known input, while remaining unpredictable to all other participants until it is revealed. When a node publishes its VRF output together with the corresponding proof, other participants can efficiently verify that the value has been correctly generated and that it could not have been modified after the fact.

Within the protocol, the VRF output serves as the source of randomness used to determine the potential publication time of a block. In this way, the VRF replaces the implicit role played by the

winning hash in Proof-of-Work systems: it provides a random outcome that cannot be anticipated and whose correctness can be publicly verified.

The second component of the protocol concerns the **probabilistic modeling of the waiting time** that must elapse before a node is allowed to publish a block. In Proof-of-Work systems, the time between successive blocks follows an exponential distribution, which emerges naturally from the repeated execution of independent hash trials. Each hash attempt represents a Bernoulli experiment with a very small probability of success, and the time until the first success corresponds to the exponential distribution that characterizes block discovery.

The proposed protocol reproduces this statistical behavior by explicitly sampling a random variable that represents the number of abstract time units a node must wait before publishing its block. This waiting time is modeled through a geometric distribution, which represents the discrete counterpart of the exponential distribution and describes the number of trials required before the first success occurs.

However, unlike the classical geometric distribution, the probability of success cannot be assumed to remain constant over time in a real permissionless network. In systems such as Bitcoin, the global computational power of the network varies continuously as miners join or leave the system. In order to maintain a stable average block production rate, the protocol periodically adjusts the mining difficulty.

This dynamic introduces an important complication in the probabilistic model. When the difficulty parameter changes, the probability associated with each trial also changes, meaning that the waiting process can no longer be represented by a single geometric distribution with a fixed parameter. Instead, the probability of success becomes piecewise constant over time, remaining stable within each epoch but potentially changing between epochs.

To capture this behavior, the thesis introduced the Geometric Distribution with Variable Probability (GD-VP), a probabilistic model capable of representing waiting processes in which the success probability varies across discrete intervals. The GD-VP formalism allows the protocol to maintain the same statistical dynamics observed in Proof-of-Work systems while explicitly incorporating the effects of difficulty adjustment.

The third and final component of the protocol concerns the **enforcement of the waiting time** determined by the probabilistic model. Knowing that a node must wait a certain amount of time before publishing a block is not sufficient if the node can simulate this waiting or compute it instantaneously. Without a mechanism enforcing the sequential passage of time, participants could bypass the intended delay and publish blocks prematurely.

To address this issue, the protocol employs Verifiable Delay Functions. A VDF is a cryptographic primitive that requires a predetermined number of sequential operations to evaluate and that cannot be significantly accelerated through parallel computation or specialized hardware. Once the function has been evaluated, however, the correctness of the result can be verified efficiently by any participant.

In the proposed protocol, the waiting time extracted from the probabilistic model is translated into the number of sequential steps required to evaluate the VDF. The node must therefore perform the VDF computation before being able to publish its block. When the block is broadcast to the network, the accompanying proof allows other participants to verify that the required waiting time has been respected.

This mechanism transforms the probabilistic waiting time into a real and non-bypassable temporal constraint. Even if a node knows in advance how long it must wait, it cannot accelerate the evaluation

of the VDF beyond the limits imposed by sequential computation. As a result, the protocol preserves the fundamental property that block publication events occur unpredictably over time.

By combining these three components—verifiable randomness, probabilistic waiting time extraction, and verifiable delay enforcement—the proposed protocol reproduces the statistical structure that underlies Proof-of-Work consensus. Each node independently determines its potential publication time, performs the required waiting computation, and publishes a block when its deadline is reached. The global ordering of blocks therefore emerges from the interaction of many independent processes rather than from explicit coordination or leader election.

This architecture preserves several of the desirable properties observed in Proof-of-Work systems. In particular, block proposers are not predetermined, participation remains open to new nodes, and block publication events occur unpredictably. At the same time, the protocol reduces the reliance on continuous computational competition, replacing repeated hash trials with a more targeted form of sequential computation required only when a node has obtained the right to publish a block.

In order to evaluate the behavior of the proposed protocol, a **simulation prototype** was developed as part of this work. The simulator models a network of nodes interacting under configurable parameters such as network latency, node computational power, and difficulty adjustment intervals. Through simulation, it becomes possible to observe the evolution of the blockchain under different conditions and to analyze phenomena such as fork formation, block propagation delays, and the distribution of block production among participants.

The experimental results obtained from the simulator provide preliminary evidence that the proposed mechanism can reproduce the **expected statistical properties of block generation** while maintaining verifiability and resistance to local manipulation. In particular, the simulations show that the probabilistic model used to determine waiting times produces block publication intervals comparable to those observed in traditional Proof-of-Work systems when appropriate parameters are chosen.

Another important aspect analyzed in the thesis concerns the vulnerability of open distributed systems to Sybil attacks. Because permissionless networks allow participants to create identities freely, adversaries may attempt to increase their influence by controlling a large number of identities. Without appropriate countermeasures, this behavior could undermine the fairness and security of the consensus process.

The thesis therefore examined several approaches to introducing participation costs that ensure influence in the system remains proportional to the resources committed by each participant. By associating economic or computational costs with identity creation and continued participation, the protocol can transform Sybil attacks from a **trivial identity-generation problem into a resource allocation problem**. In such a scenario, controlling a large number of identities becomes economically expensive rather than technically trivial.

More broadly, the work presented in this thesis suggests a different perspective on the design of consensus mechanisms for decentralized systems. Rather than viewing Proof-of-Work as a mechanism whose essence lies in the consumption of computational power, it may be interpreted as a particular implementation of a more general principle: the probabilistic and verifiable ordering of actions in a distributed system composed of mutually untrusted participants.

By identifying and isolating the components that make this principle possible—randomness generation, probabilistic waiting, and verifiable temporal constraints—it becomes possible to explore

alternative architectures that preserve the essential properties of permissionless consensus while reducing reliance on wasteful resource competition.

In this sense, the protocol proposed in this thesis does not aim to replace the conceptual contribution of Bitcoin, but rather to reinterpret and generalize the mechanism that made it possible. Understanding the deeper principles underlying Nakamoto's consensus may open the way to new classes of distributed protocols capable of achieving similar security guarantees with improved efficiency and flexibility.

Future work

While the results presented in this thesis demonstrate the conceptual feasibility of the proposed consensus approach, several aspects require further investigation in order to evaluate its applicability in real-world decentralized systems. The current work focuses primarily on the design of the protocol and on the analysis of its probabilistic behavior through simulation. Future research should therefore aim to extend these results by progressively **moving from theoretical modeling toward more realistic implementations** and experimental evaluations.

A first natural extension concerns the development of a more complete prototype capable of reproducing the temporal dynamics of the protocol under real execution conditions. The simulator developed in this thesis models the probabilistic aspects of block publication and the behavior of nodes under different network conditions but abstracts away several factors that would emerge in a real deployment. Implementing a prototype in which the protocol operates with real execution times, real cryptographic primitives, and realistic network delays would make it possible to validate the model under conditions closer to those encountered in operational distributed systems. Such an implementation would also allow the measurement of practical performance indicators, including block propagation latency, computational overhead, and system throughput.

Closely related to this aspect is the integration of a communication layer among nodes. In the current simulation environment, message propagation is represented through simplified models that approximate network delays without explicitly implementing the communication protocol used by participants. A natural direction for future work is therefore the **introduction of a gossip-like communication protocol**, similar to those employed in existing blockchain networks. Gossip-based dissemination mechanisms allow blocks and protocol messages to propagate efficiently across large decentralized networks while maintaining robustness to node failures and network partitions. Integrating such a mechanism into the prototype would enable a more realistic analysis of block propagation dynamics, fork formation, and network convergence properties.

The introduction of a communication layer also enables the **integration of an application layer** on top of the consensus mechanism. In the present work, the simulator focuses primarily on the production and ordering of blocks, without modeling the inclusion of transactions or application-level data. In a real blockchain system, however, blocks serve as containers for transactions representing state transitions within the network. Extending the prototype to include a transaction layer would make it possible to analyze the interaction between consensus dynamics and application-level workloads. In particular, it would allow the evaluation of metrics such as transaction throughput, confirmation latency, and the effect of network congestion on block production and propagation.

Another important direction for future work concerns the practical deployment and evaluation of the cryptographic primitives used by the protocol. While the simulation framework abstracts the behavior of Verifiable Delay Functions, their actual implementation introduces several parameters that may significantly influence system performance. The choice of the delay parameter, the cost of sequential computation, and the efficiency of proof verification all affect the balance between security

guarantees and computational overhead. A real deployment would therefore allow a detailed **study of VDF parameter selection**, including the calibration of delay values required to reproduce the intended temporal behavior of the protocol under realistic hardware assumptions.

In addition to the cryptographic aspects, further research is needed to **analyze the economic model** associated with participation in the protocol. As discussed in the thesis, open distributed systems are inherently vulnerable to Sybil attacks, and the introduction of participation costs represents a potential mechanism to ensure that influence remains proportional to the resources committed by each participant. Future work should therefore investigate the economic implications of such mechanisms in greater depth. In particular, the introduction of a native token associated with the protocol could allow the implementation of cost structures based on token acquisition and destruction.

In such a model, a participant wishing to join the network would first need to acquire tokens, which would then be burned as part of the identity registration or participation process. This mechanism would introduce a direct economic cost associated with the creation and maintenance of identities. At the same time, the protocol could generate new tokens as rewards for successful block publication, thereby creating a balance between token destruction and token issuance. The interaction between these two mechanisms would determine the long-term monetary dynamics of the system and could influence participation incentives, security properties, and resistance to large-scale Sybil attacks.

A rigorous economic analysis would therefore be necessary to evaluate the stability of such a model. Questions related to token supply evolution, equilibrium participation levels, and the incentives faced by rational participants would need to be addressed through both theoretical modeling and simulation. Such analysis would provide valuable insights into how economic mechanisms interact with the probabilistic consensus model proposed in this thesis.

Overall, these research directions represent a natural continuation of the work presented here. While the thesis focused primarily on the conceptual and probabilistic foundations of the protocol, future developments should progressively address the practical, network-level, and economic aspects required for the deployment of a fully operational decentralized system. By combining theoretical analysis with experimental validation and economic modeling, it may become possible to further assess the viability of consensus mechanisms based on verifiable randomness, probabilistic waiting times, and verifiable delay functions as an alternative approach to resource-intensive Proof-of-Work systems.

References

1. **Tanenbaum, Van Steen.** *Distributed Systems: Principles and Paradigms*.

2. **Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985.** *Impossibility of distributed consensus with one faulty process. J. ACM 32, 2 (April 1985), 374–382*.

3. **Lamport, Shostak, Pease.** *The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 1982*.

4. **Castro, Liskov.** *Practical Byzantine Fault Tolerance. OSDI, 1999*.

5. **Buchman.** *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Master Thesis, 2016*.

6. **Yin et al.** *HotStuff: BFT Consensus with Linearity and Responsiveness. PODC, 2019*.

7. **Lamport, L.** *he Part-Time Parliament. ACM Transactions on Computer Systems*.

8. **Dwork, C., Lynch, N., & Stockmeyer, L.** *Consensus in the Presence of Partial Synchrony*.

9. **Pease, M., Shostak, R., & Lamport, L.** *Reaching Agreement in the Presence of Faults. Journal of the ACM, 27(2), 228–234*.

10. **Nakamoto., Satoshi.** *Bitcoin: A Peer-to-Peer Electronic Cash System*.

11. **Garay et al.** *The Bitcoin Backbone Protocol Against Quantum Adversaries, 2019*.

12. **Werner Vogels. 2009.** *Eventually consistent. Commun. ACM 52, 1 (January 2009), 40–44*.

13. **Arthur Gervais et al.** *On the Security and Performance of Proof of Work Blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 3–16*.

14. **Eyal, I., & Sirer, E. G. (2014).** *Majority is not Enough: Bitcoin Mining is Vulnerable. FC*.

15. **Bonneau et al.** *SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. IEEE S&P, 2015*.

16. **Douceur.** *The Sybil Attack. IPTPS, 2002*.

17. **Viswanath, B., et al. (2012).** *An Analysis of Social Network-Based Sybil Defenses. SIGCOMM*.

18. **Heilman, E., et al. (2015).** *Eclipse Attacks on Bitcoin's Peer-to-Peer Network. USENIX Security*.

19. **FIPS 180-4 – Secure Hash Standard.** *National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4, 2015*.

20. **Narayanan et al.** *Bitcoin and Cryptocurrency Technologies. Princeton University Press*.

21. **Taylor et al.** *Block arrivals in the Bitcoin blockchain, 2018*.

22. **PLATT, Moritz, et al.** *The energy footprint of blockchain consensus mechanisms beyond proof-of-work. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2021. p. 1135-1144*.

23. **de Vries.** *Bitcoin's Growing Energy Problem. Joule, 2018*.

24. **Sompolinsky, Y., & Zohar, A. (2015).** *Secure High-Rate Transaction Processing in Bitcoin. FC*.

25. **King, S., & Nadal, S. (2012).** *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*.

26. **Nxt Community.(2013).**

27. **BlackCoin Development Team. (2014).**

28. **Buterin et al.** *Ethereum Proof-of-Stake Design*.

29. **Kiayias et al.** *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. CRYPTO, 2017*.

30. **DFINITY Foundation. (2021).** *The Internet Computer Protocol*.

31. **Gilad et al.** *Algorand: Scaling Byzantine Agreements for Cryptocurrencies. SOSP, 2017*.

32. **Cachin, Vukolić.** *Blockchain Consensus Protocols in the Wild*.

33. **SALEH, Fahad.** *Blockchain without waste: Proof-of-stake. The Review of financial studies, 2021, 34.3: 1156-1190*.

34. **Micali, S., Rabin, M., & Vadhan, S. (1999).** *Verifiable Random Functions. FOCS*.

35. **Maresca, M., Andreoli, L. e Baglietto, P.** *Leveraging Geometric Distribution with Variable Probability to Pre-Calculate Block Publication Deadlines in a Blockchain Simulation. Blockchains 2025, 3, 9*.

36. **Boneh, D., Bonneau, J., Bünz, B., Fisch, B. (2018).** *Verifiable Delay Functions. In: Shacham, H., Boldyreva, A. (eds) Advances in Cryptology – CRYPTO 2018. CRYPTO 2018. Lecture Notes in Computer Science(), vol 10991. Springer, Cham*.

37. **Wesolowski, B. (2019).** *Efficient Verifiable Delay Functions. EUROCRYPT*.

38. **Ross.** *Introduction to Probability Models*.

39. **Banks, J., et al.** *Discrete-Event System Simulation, 4th ed.; Prentice Hall: Englewood Cliffs, NJ,USA,2005*.

40. **Brown.** *Calendar Queues: A Fast O(1) Priority Queue Implementation, CACM 1988*.

41. **Elli Androulaki et al.** *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains, Proceedings of the Thirteenth EuroSys Conference, 2018*.