Sprints: Intermittent Blockchain PoW Mining

Michael Mirkin, *Technion;* Lulu Zhou, *Yale University;*
Ittay Eyal, *Technion;* Fan Zhang, *Yale University*

https://www.usenix.org/conference/usenixsecurity24/presentation/mirkin

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Sprints: Intermittent Blockchain PoW Mining

Michael Mirkin
Technion

Lulu Zhou
Yale University

Ittay Eyal
Technion

Fan Zhang
Yale University

Abstract

Cryptocurrencies and decentralized platforms have been rapidly gaining traction since Nakamoto's discovery of Bitcoin's blockchain protocol. Prominent systems use Proof of Work (PoW) to achieve unprecedented security for digital assets. However, the significant carbon footprint due to the manufacturing and operation of PoW mining hardware is leading policymakers to consider stark measures against them and various systems to explore alternatives. But these alternatives imply stepping away from key security aspects of PoW.

We present Sprints, a blockchain protocol that achieves almost the same security guarantees as PoW blockchains, but with an order-of-magnitude lower carbon footprint while increasing the number of mining rigs by a factor 1.27x. Our conservative estimate of environmental footprint uses common metrics, taking into account both power and hardware. To achieve this reduction, Sprints forces miners to mine intermittently. It interleaves Proof of Delay (PoD, e.g., using a Verifiable Delay Function) and PoW, where only the latter bears a significant resource expenditure. We prove that in Sprints the attacker's success probability is the same as that of legacy PoW. To evaluate practical performance, we analyze the effect of shortened PoW duration, showing a minor reduction in resilience (49% instead of 50%). We confirm the results with a full implementation using 100 patched Bitcoin clients in an emulated network.

1 Introduction

Proof of Work (PoW) cryptocurrencies offer a decentralized form of money, with monetary policy dictated by code. Participants can acquire coins and perform transactions without needing permission from other parties or centralized exchanges. PoW cryptocurrencies, starting with Nakamoto's Bitcoin [1], have gained significant success with market capitalization in the hundreds of billions [2] and attract the attention of major financial institutions [3, 4]. But PoW cryptocurrencies consume significant resources, with Bitcoin's electricity consumption surpassing Argentina's in 2022 [5]. Environmental concerns are leading to policy changes, including bans [6, 7]. Nevertheless, the stable valuation of the PoW Bitcoin, despite the proliferation of non-PoW alternatives, demonstrates the demand for PoW guarantees. It implies the need for a protocol that provides such guarantees with lower carbon footprint.

Indeed, both theoretical work and operational systems address this issue. Previous work (§2), showed [8] that resource expenditure cannot be reduced by tuning the protocol parameters. Proof of Storage [9] requires participants to dedicate storage resources instead of computation but is about 50% cheaper to attack [10]. Proof of Stake (PoS) protocols (e.g., [11–16]) take a different approach: Instead of physical resource expenditure, PoS uses on-chain deposits, thus the likelihood of a miner being able to create a new block and add it to the blockchain is determined by the amount of cryptocurrency they have *staked*, i.e., held in their account. With this approach, no physical resource is expended. But PoS protocols require stronger assumptions, such as long-term connectivity and availability [17–19].

Sprints. We introduce *Sprints* (§4), a blockchain protocol that maintains the advantages of PoW but with significantly lower resource consumption. The key idea is to force miners to perform PoW intermittently, i.e., to insert periods of time during which miners pause mining. Like Bitcoin, *Sprints* miners collect transactions (e.g., payment orders) from users and batch them into *blocks* that they broadcast. Each block contains a hash reference to its predecessor, so the blocks together form a tree with an agreed-upon root. The system state is thus obtained by processing the transactions in the longest path in the tree, called the *longest chain*.

In addition to PoW — statistical proofs that the miner expended computational effort, *Sprints* also requires PoD, proving the miner waited for a certain time before the PoW computation. Unlike PoW, PoD computation does not require significant computational resources. Thus, a miner alternates between producing PoD with nominal power expenditure and PoW with high power expenditure – Figure 1 illustrates this process. The PoD primitive shares similarities with a Verifiable Delay Function (VDF) [20–24]. The distinction lies in the uniform delay period enforced across all miners, a feature not generally guaranteed by VDFs. Uniform delay periods are crucial for the protocol's security (§6.5), while a perfect PoD architecture is currently theoretical, near-ideal PoD solutions are attainable with a reliable hardware setup assumption [25, 26].

Like PoW systems where participation is profitable only for miners with efficient hardware [27–29], *Sprints* is profitable only for miners with efficient PoD hardware. We thus assume all miners have similar, efficient, PoD hardware; others with weaker hardware would not participate.

Security. At first glance, it might seem that an attacker has an advantage compared to a *pure PoW* system like Bitcoin: She does not have to stop calculating PoW while performing the PoD calculation, whereas honest miners will. We prove (§5) that such behavior specifically is futile, and that in general there is no sacrifice of security compared to a pure PoW system. Due to the distinct characteristics of *Sprints*, previous proof techniques [10, 30, 31] are not applicable. Specifically, the PoD puzzle delay eliminates the memorylessness of pure PoW systems and prevents the use of the common Markov chain analysis. Consequently, we are compelled to devise a novel proof approach.

The key step is showing that in a race where the attacker competes with the honest miners, the attacker has to solve PoW and PoD puzzles sequentially, thus, parallel mining would not benefit her. Intuitively, the only race between the honest chain and that of the attacker is the PoW puzzles, as both spend the same amount of time-solving PoD puzzles for the same number of blocks. The implication is that the attacker's probability of winning the race is the same as in a pure PoW blockchain. We use this result to show that if we consider a race where the honest miners and the adversary are building two chains that extend the same block and the adversary controls less than half of the PoW computational power, then her probability of building a chain that is at least r blocks deep and is longer than the honest chain is bounded by $2^{-\Omega(r)}$. Finally, we utilize this result to show that if the propagation delay is negligible, *Sprints* provides the same guarantees as pure PoW.

Implementation and evaluation. If we compare Sprints to a pure PoW system with the same block interval, since part of the block generation is computing PoD, PoW duration in Sprints is shorter. Our theoretical analysis shows that with shorter PoW average duration it is more likely for honest miners to generate blocks with the same parent, forming *forks*. The implication is that the longest chain extension is slower and the threshold attacker size is smaller.

To confirm this analysis, we implemented *Sprints* by patching the standard Bitcoin client [32] and measured the frequency of forks as a function of both the network propagation delay and the PoW duration. With system parameters similar to those of the operational Bitcoin system [33], when using only 5% of block interval for PoW, the threshold attacker size is 49%, compared to Bitcoin's 50%.

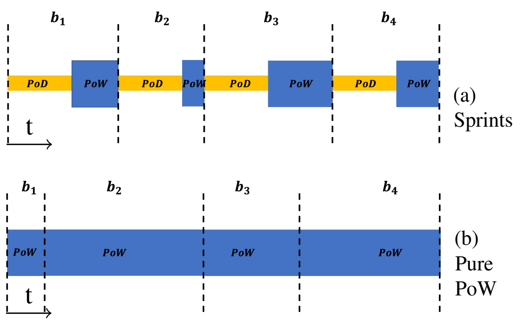
Implementing *Sprints* involves addressing several practical challenges (§6). To address spam prevention while maintaining efficient propagation, we lazily validate PoDs, i.e. nodes do not wait for the PoD validation to complete before propagating the block, allowing for the same propagation delay as a pure PoW system without sacrificing spam prevention [34]. Additionally, we adjust the difficulty of both PoW and PoD to maintain constant block intervals as hardware for PoW and PoD improves. We do this by estimating parameters using the second moments of the interval probability distribution.

Enviromental footprint. The analysis *Sprints*'s enviromental footprint (§7) is more involved than comparing the ratio of PoW per block. We show that *Sprints* miners use their budget to purchase more mining equipment compared to pure PoW, for the short duration where they PoW-mine. In other words, *Sprints* shifts a portion of the *operating expenses* (*OPEX*) to *capital expenditure* (*CAPEX*), which we show to have a lower carbon footprint by comparing the emissions from the hardware lifecycle to the emissions from electricity consumption. On the other hand, the increase in the number of mining rigs implies a potential increase in rare metal depletion. While raw materials like rare-earth have an environmental impact (e.g., they are difficult to recycle), we show that the increase is significantly smaller than the factor by which the carbon footprint decreases. We use the CO2e (carbon dioxide equivalent) metric [35] to compare systems, which allows us to quantify the emissions resulting from electricity consumption and the hardware lifecycle (manufacturing, transportation, and disposal). Importantly, the assumptions made in our analysis err on the side of conservatism, suggesting that the actual reduction in carbon footprint could be greater than our estimates indicate.

Combining the security and emission analyses, Figure 2 shows how the attack threshold and emission reduction ratio change with the PoW time ratio when using Bitcoin-like parameters (100ms network delay, 600s block interval [33]). At 5% of the block interval the CO2e is 9.2% that of Bitcoin. At the same time, the total number of mining rigs in *Sprints* increases by 27%, implying an increase in rare metal depletion by a factor of 1.27.

In summary, our main contributions are:

Figure 1: Pure PoW and *Sprints* over time.



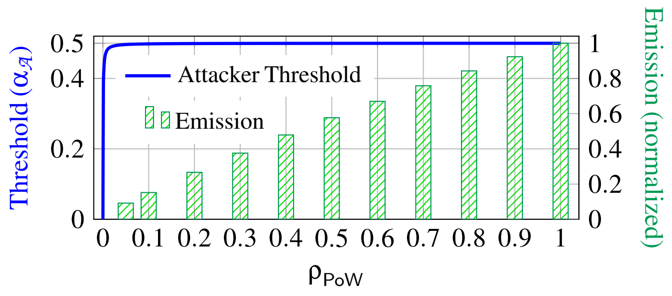


Figure 2: Attack threshold and normalized emission under as a function of portion of PoW time out of block interval

- *Sprints*, a novel blockchain protocol that introduces intermittent mining, alternating between PoD and PoW;
- proof that the security threshold is the same as in PoW;
- tuning of both PoD and PoW based on block interval;
- evaluation with full implementation over an emulated network demonstrating effective difficulty adjustment and close-to-optimal (49%) attack threshold values; and
- carbon footprint analysis showing over 10*x* CO2e reduction.

2 Related Work

We review energy-efficient alternatives to PoW and their trade-offs.

The most well-established energy-efficient alternative to PoW is Proof of Stake (PoS) [11–16]. Instead of solving hash puzzles, PoS miners participate in a mining lottery with a winning probability proportional to their token holding in the system. However, these protocols require stronger assumptions than PoW protocols regarding network connectivity, validator behavior, or the availability of a randomness oracle [17–19]. Additionally, in PoS protocols new nodes need cooperation from existing nodes to join the network [36], in contrast to *Sprints*, where new nodes can join the network by mining blocks using external computations.

Several protocols take advantage of Verifiable Delay Functions (VDFs). Verifiable Delay Functions (VDFs) [20–24] are a class of functions that cannot be accelerated by a parallel computation and can be verified efficiently. Long and Wei [37] propose a PoS protocol that incorporates a variation of Verifiable Delay Functions (VDFs) that has a random delay. However, the threshold for a successful private attack is less than 27% compared to almost 50% in *Sprints*. PoSAT [38] has a similar construction. PoSAT is vulnerable to nothing-at-stake attacks, where an attacker can mine on multiple forks for no additional cost. The authors, therefore, divide the rounds into epochs which prevents new players from joining the network during an epoch. The system has a threshold of 50% for a private attack only when the epochs are infinitely long, which turns it into a permissioned system. Thus, PoSAT has a tradeoff between decentralization and security.

HEB [39] also utilizes on-chain resources uses mechanism design to reduce electricity consumption by 50%, but reduces resilience to malicious attacks by 2, while *Sprints* has almost the same threshold as pure PoW with an 8x reduction in electricity consumption.

Several approaches reduce expenditure with particular types of PoW. REM [40] and PoET [41] use trusted hardware to reduce resource expenditure. As the protocols are based on trusted hardware, they rely on a trusted party to guarantee the hardware's integrity. *Sprints* makes no such assumptions.

Chia [9] shifts the costs of miners from electricity to hardware by combining Proof-of-Space (PoS) and Proof-of-Time (PoT). It reduces electricity consumption by replacing much of the mining costs with storage costs. However, it is resilient to attackers with under 30% of the network storage resources [10], which is significantly lower than the almost 50% threshold of *Sprints*.

Another approach to deal with this challenge is to use permissioned protocols [42–44], which are a class of protocols that require pre-authorization of nodes to participate in the protocol. However, permissioned protocols are not decentralized and require a trusted third party to authorize new nodes.

As for the analysis of PoW protocols, Dembo et al. [10] analyze the security of blockchain systems by showing that a so-called *private attack* is the worst-case attack. They define the notion of Nakamoto blocks and use their existence to prove that the system is secure. We use a different approach, where we avoid the notion of Nakamoto blocks and instead describe a single race between the attacker and the honest miners. We use this race to prove directly that the security requirements hold.

3 Model

The system consists of a set of participants called *miners*. Each miner maintains a tree data structure whose vertices are called *blocks*. Each block contains *transactions*, commands issued by system users, which are the *payload* of the block. All miners start with the same block, called *genesis*, that serves as the root of the tree. In addition to the payload, each block contains metadata. The metadata of all blocks (except the genesis block) includes a hash that points to its *parent* block; we say that a block *extends* the pointed-to block. A path with the most blocks in the tree is called a *longest chain*. There may be several longest chains and one of them, chosen by a deterministic arbitrary algorithm, is called the *main chain*. The *height*[1] of a block b is the number of blocks in the path from the genesis to b.

Time progresses in discrete steps $t = 0, 1, \ldots$ (as in, e.g., [45]). In each step, miners can *work* on two types of *puzzles*. The first type is *Proof-of-Delay (PoD)*, a function that maps an arbitrarily sized input and difficulty parameter to a small

[1]Called depth in graph-theory literature.

output. Given a random input, a PoD requires a deterministic number of steps, called *delay period* and denoted by Δ_{PoD}. In each step, a miner can choose to mine on multiple PoD puzzles. After working on a particular PoD puzzle in Δ_{PoD} distinct steps, a miner has its solution, which cannot be guessed except with negligible probability. The formal specification of a PoD is the same as that of a VDF [20] with the added requirement that all the players have the same delay period. This assumption is reasonable since miners who cannot solve the PoD puzzle in the given delay period will not be able to compete in the protocol, and thus will be forced to leave the system.

The second puzzle type is *Proof-of-Work*, a probabilistic puzzle that has an independent probability of being solved in any given step. In each step, a miner can choose to mine on a PoW puzzle. Miner p has a probability $P_w(p)$ of solving it in each step independent of previous attempts. The number of steps until success thus has a Geometric distribution with parameter $P_w(p)$.

Combining metadata M, payload D and puzzle solutions Z, a block is the tuple (M,D,Z).

The miners communicate over a Δ-synchronous broadcast network [10, 30]: If a miner sends a message in step t, then all miners receive it by step $t+\Delta$. We assume that $\Delta < \Delta_{PoD}$.

Thus, at the beginning of every step, a miner receives messages sent in previous steps. The miner can then perform local computations, i.e., work on one PoW and multiple[2] puzzles. Then the environment notifies the miners if they found solutions. Finally, the miner can broadcast blocks.

There are a total of $n+1$ miners in the system, where n are *honest*, i.e., miners that follow a predefined protocol, and a single miner is controlled by an *adversary* $\mathcal{A}$ and acts arbitrarily.

Note that the adversary can only work on a single PoW puzzle in a single step in our model. However, working on multiple PoW puzzles in parallel can be approximated by frequent puzzle changes.

The adversary controls the message delay, constrained by the bound Δ.

Next, we define a predicate that validates the correctness of a block and its puzzles. A *validity function* $V(b)$ returns *true* if some predefined conditions on the block contents are met and *false* otherwise. A block b is *valid* if $V(b) = true$. Invalid blocks with either invalid payloads or proofs are simply ignored.

An *execution* is a series of states of the system that develops based on the miners' algorithms and the environment's coin flips, i.e., the outcome of PoW puzzle-solving attempts. Each execution has a certain probability of occurring. Let Σ denote the set of all executions. Given step t_0, denote by π the t_0*−prefix* that is the collection of all executions in Σ that agree on the state of the system at step t_0. Given an execution σ, denote by $\mathcal{T}^{\sigma}(t)$ the tree that corresponds to the execution in step t, called a *mother tree* [10]. The mother tree consists of all valid blocks in the system (published or not) until step t. Note that $\mathcal{T}^{\sigma}(t)$ represents the state of the tree at the beginning of the step. Denote the depth of a valid block b by $d(b)$. Given a execution σ, denote the depth of the longest chain(s) in $\mathcal{T}^{\sigma}(t)$ by $d^{\sigma}(t)$. Each player p has a local copy of the mother tree, which is updated according to the blocks she receives, it is denoted by $\mathcal{T}^{\sigma}_{p}(t)$.

Given an execution σ, the execution view of miner p is denoted by σ_p. It includes all the information she received and the results of her local computations.

In each step, each honest miner performs an *action* that comprises the mining target for PoW and PoD and the blocks a miner publishes. The action is defined by the *mining function* $q(\sigma_p)$, given a view σ_p of miner p. The vector $Q^{\sigma}_{H}(t)$ includes the honest miners' actions in step t defined by the mining function. We denote by $Q^{\sigma}_{\mathcal{A}}(t)$ the action of the adversary at step t and execution σ. The adversarial action is decided based on a predefined *strategy* A, which is a map from step t and a state of execution at step $t-1$ to a vector that represents the mining targets for PoW and PoD, the blocks being published and the delay the adversary imposes on messages.

A longest chain protocol is thus defined by a validity function and a mining function, $(V(\cdot), q(\cdot))$. For each node, the sequence of payloads along the main chain, excluding the payloads of the last r blocks (for some r), is called a *ledger*. We use the notions of *persistence* and *progress*[3], similar to the definitions in previous work [10, 30, 31]. Given a block b, we denote by $d^{\sigma}_{b}(t,p)$ the depth of the longest chain in the view of miner p in step t that contains b and by $d^{\sigma}_{\neg b}(t,p)$ the depth of the longest chain in the view of miner p in step t that does not contain b.

Definition 1. *A protocol* $(V(\cdot),q(\cdot))$ *implements a ledger if it satisfies the following two conditions:*

Persistence *Given* $\varepsilon > 0$ *and step* t_f*, there exists* $r \in \mathbb{N}$ *such that for all adversarial strategy* A *given a randomly drawn execution* $\sigma \leftarrow \Sigma_A$*, for every step* $t \leq t_f$ *and every block* $b \in \sigma$*, if* b *is at depth* i *of the main chain and* $d^{\sigma}_{b}(t-\Delta,p) > d^{\sigma}_{\neg b}(t-\Delta,p)+r$ *for an honest miner* p*, then for every* $t' > t$ *it holds that* b *is in depth* i *in the main chain of the view of all other honest miners with probability at least* $1-\varepsilon$*.*

Progress[3] *Given* $\varepsilon > 0$ *and* t_f*, there exists* $\delta \in \mathbb{N}$*, s.t. for all steps* $t_0 \leq t_f$ *and adversarial strategy* A *the probability that a random execution* $\sigma \in \Sigma_A$ *does not include a block that was mined in* $[t_0, t_0+\delta]$ *in the main chain of all honest miners for some step* $t' > t_0+\delta$ *is smaller than* ε*.*

4 *Sprints*

We now present the *Sprints* protocol, which implements a ledger, by defining a mining function $q(\sigma_p)$ for a player p and a validity function $V(b)$.

A valid block in *Sprints* contains two puzzle solutions: a proof of delay b^{PoD} that enforces serial computation, and a

[2]The number of puzzles is polynomial in a system security parameter; we omit these details to simplify the presentation.

[3]Our Progress is called Liveness in some prior work [10, 30, 31] although it is a safety property [46].

proof of work b^{PoW}. The mining function chooses the deepest block, partial or full, in the view of the miner and returns the puzzle required. If the deepest block is a full block, then the mining function works on the PoD of the next block, as follows: Given a metadata M that contains the previous block's hash, the mining function returns the action to take a step in the PoD puzzle of M. If the deepest block is partial, i.e., the PoD puzzle of the new block was already solved, then the mining function works on the PoW puzzle, as follows: Given the PoD puzzle solution of the metadata, b^{PoD}, and the payload of the new block, D, the mining function returns the action to take a step in the PoW puzzle of $b^{PoD}||D$. If the PoW step is successful and the miner obtains a solution b^{PoW}, then the miner publishes the new block: $b = (M, D, (b^{PoD}, b^{PoW}))$.

The validity function $V(b)$ validates a block b by checking that the PoD and PoW puzzles are valid, using the validity functions $V_{PoD}(\cdot)$ and $V_{PoW}(\cdot)$, respectively:

$$V(b) := V_{PoD}(b^{PoD}, M) \wedge V_{PoW}(b^{PoW}, b^{PoD}||D).$$

Note that the PoD does not require the payload of the block, while the PoW requires the payload of the block.

If a node learns of a chain longer than the block it is currently working on, it discards its work and begins generating a block extending the new chain.

The pseudocode of the protocol is in the full version of this paper [47].

5 Security

We prove that *Sprints* achieves both persistence and progress. To do so, we tackle two main challenges:

1. **Persistence:** we aim to prove that a block in the main chain will almost surely stay there. We assess the risk of an attacker removing a block by breaking the problem into smaller "mini-games," each focused on a block at a specific depth. We then sum up the probabilities of the attacker winning these mini-games using a union bound, giving us an overall likelihood of a persistence violation.

2. **Progress:** we show that, given a sufficiently long duration, at least one honestly-mined block within that period will remain in the main chain forever with high probability. This is decomposed into:

   (a) Establishing that given $r > 0$, honest miners will discover r blocks within a long enough period.

   (b) And showing that the probability of an adversary forming a tree deeper than the tree in each honest miner's view by at least r blocks is bounded by $2^{-\Omega(r)}$.

Both proofs employ a reduction to a *simple game*. In this game the adversary and honest miners initially mine their respective block trees, stemming from a shared genesis block. The adversary's goal is to reach or surpass the length of the honest miners' tree once both trees have reached a length of at least r blocks. Subsequently, we examine an infinite set of games where the adversary achieves victory when both trees have lengths of at least $r, r+1, r+2, \ldots$, and so forth. We prove that the sum of the probabilities of the adversary winning each game is bounded by $2^{-\Omega(r)}$. To do this we find the attack that is most likely to succeed and bound the probability of its success; we say this attacker is the *worst attacker*. In particular, we should find which PoW and which PoD the worst attacker should calculate at each step. For PoD, we assume that the adversary mines all possible PoD blocks in parallel. This assumption makes the adversary stronger since any other adversary can be emulated by the one described above. Subsequently, we prove that the optimal strategy, defined as the one that maximizes the length of the adversary's subtree while minimizing the honest subtree for any given step, is the strategy that PoW mines on the deepest block in the adversary's subtree as long as no PoW solution exists of the same depth, i.e., a complete PoW puzzle does not yet extend this PoD puzzle. In the latter scenario, the adversary halts PoW until a PoD puzzle is found.

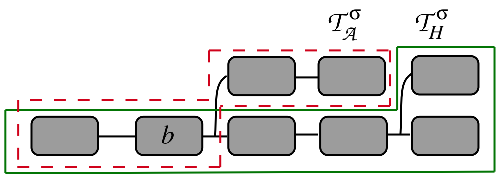
After showing that we only need to consider this worst attack, we find that when adversary and honest subtrees are of equal length, they contain the same number of PoD blocks. This reduces the competition to just PoW computations, paving the way for an easy conclusion to the proof using standard tools, namely random walk analysis.

Before detailing the proof, we present some terminology and notation (§5.1). We then discuss optimal strategies for the adversary (§5.2) and the honest miners (§5.3) subtrees. Next, we use these strategies to analyze the simple game (§5.4). Finally, we prove persistence and progress and the security of *Sprints* (§5.5).

5.1 Terminology and Notation

We define a race between the honest miners and an adversary with strategy A. We assume that the adversary always chooses to PoD-mine on all possible blocks in parallel, as every strategy A_1 that PoD-mines only some blocks creates the same tree as the one created by a strategy A_2 that is identical to A_1 but PoD-mines everywhere. Therefore, for succinctness, we define a mining action at step t as the block that the adversary chooses to PoW mine on; if there is no such block, the adversary's action is $\perp$, i.e., empty action. The adversary chooses a delay for each block b and honest miner p. To simplify the analysis, we assume that the attacker can delay the block even for the miner who found it. This assumption only strengthens the adversary, allowing us to bound the adversary's success probability.

Given a depth i, execution σ and honest block b at depth i, we define two *sub-trees* (portrayed in Figure 3): for all steps t, $\mathcal{T}^{\sigma}_{H}(t)$ is the sub-tree of $\mathcal{T}^{\sigma}(t)$ that includes b, its descendants that are known by all honest miners and its



$\mathcal{T}^{\sigma}_{\mathcal{A}}$
$\mathcal{T}^{\sigma}_{H}$
b

Figure 3: Scheme of $\mathcal{T}^{\sigma}_{\mathcal{A}}$ and $\mathcal{T}^{\sigma}_{H}$ depending on b.

ancestors; tree $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$ is the sub-tree of $\mathcal{T}^{\sigma}(t)$ that includes b and its private adversarial descendants and ancestors. Note that $\mathcal{T}^{\sigma}(t)$ can include adversarial blocks.

A block data structure containing only the PoD puzzle, $(M,D,(b^{PoW},\perp))$, is called a *partial block* and a valid block with all proofs is called a *full block*. Denote the depth of a partial block b by $\tilde{d}(b)$. Given an execution σ, we define the *PoD depth* of a tree as the deepest block in the mother tree that has a completed PoD puzzle on the subtree, it is denoted by $\tilde{d}^{\sigma}_{b}(t)$ for $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$ and by $\tilde{d}^{\sigma}_{\neg b}(t)$ for $\mathcal{T}^{\sigma}_{H}(t)$. For some π, denote by $\Sigma^{\pi_i}_{A}$ the subset of Σ^{π}, where the adversary follows a strategy A.

Next, we define the optimality of an adversarial strategy. We look separately at $\mathcal{T}^{\sigma}_{H}(t)$ and $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$ and find a single strategy that minimizes the depth of the former and maximizes the depth of the latter for all steps.

5.2 Adversary tree

We focus on the optimal strategy of the adversary on her private tree $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$.

Given a prefix π of length t_0 and an execution $\sigma \in \pi$, we start by considering only the actions of an adversary targeting the subtree $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$. At first, we assume that given some step $t_f > t_0$, the adversary aims to maximize the depth of $\mathcal{T}^{\sigma}_{\mathcal{A}}(t_f)$. Later we generalize this to a strategy that aims to maximize the depth of $\mathcal{T}^{\sigma}_{\mathcal{A}}(t)$ for all $t > t_0$.

We define the notion of an *optimal strategy*. As a first step, we define a (t_f,ℓ)-optimal strategy that maximizes the probability that at step t_f the depth of $\mathcal{T}^{\sigma}_{\mathcal{A}}$ is at least ℓ. For a set of executions Σ^{π}_{A}, denote by $\Pr_{\Sigma^{\pi}_{A}}[\cdot]$ the conditional probability $\Pr[\cdot|\sigma \in \Sigma^{\pi}_{A}]$.

Definition 2. *An attacker strategy A is (t_f,ℓ)-*optimal *if, for all prefixes π of length $t_0 < t_f$ and strategies A', it holds that $\Pr_{\Sigma^{\pi}_{A}}[d^{\sigma}_{\neg b}(t_f) \geq \ell] \geq \Pr_{\Sigma^{\pi}_{A'}}[d^{\sigma}_{\neg b}(t_f) \geq \ell]$. Strategy A is* optimal *if it is (t_f,ℓ)-optimal for all t_f and ℓ.*

For execution σ, denote by $B_{max}(\mathcal{T}^{\sigma}_{\mathcal{A}}(t))$ the *tips of the chain* the set of deepest partial blocks in step t whose PoD puzzle is completed locally for player p. Denote their depth by $\tilde{d}^{\sigma}_{\neg b}(t)$.

Definition 3 (longest-chain mining)**.** *Given an execution σ, an attacker strategy that chooses a block in $B_{max}(\mathcal{T}^{\sigma}_{\mathcal{A}}(t))$ for $t \geq t_0$ is* longest-chain mining *(*LCM*).*

We show that any longest-chain mining strategy is (t_f,ℓ)-optimal.

Lemma 1. *For all prefixes π of length t_0, ℓ and steps t_f, an LCM strategy A played from step t_0 is (t_f,ℓ)-optimal.*

Before proving Lemma 1, we introduce another lemma, which will also be useful later. We show that given two execution sets where the first has higher partial depth at step t_0 and Δ_{PoD} steps later, it has a higher probability to be deeper at step $t_f > t_0$ if the attacker follows a (t_f,ℓ)-optimal strategy.

Lemma 2. *Let there be two prefixes π_1 and π_2 of length t_0, given t_f and ℓ, and a (t_f,ℓ)-optimal strategy A. Consider two sets of executions, Σ_1 and Σ_2, such that $\Sigma_i = \Sigma^{\pi_i}_{A}$. We assume that for π_1 and π_2, in step t_0 no PoD puzzle is being calculated so that at some point in the future, an execution from Σ_2 will be deeper than any execution from Σ_2 due to this puzzle. Formally:*

$$\forall \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2, t \in [t_0, t_0 + \Delta_{PoD}] : \tilde{d}^{\sigma_1}_{\neg b}(t) \geq \tilde{d}^{\sigma_2}_{\neg b}(t).$$

Then it holds that

$$\Pr_{\Sigma_1}[d^{\sigma}_{\neg b}(t_f) \geq \ell] \geq \Pr_{\Sigma_2}[d^{\sigma}_{\neg b}(t_f) \geq \ell].$$

The full proof of Lemma 1 and Lemma 2 is given in the full version of this paper [47]. The main idea is that we prove both lemmas by double induction starting from t_f. We run the induction for decreasing step period and prove both lemmas together at each step of the induction. We first show that the base of the induction holds for $t_0 \in [q_0, t_f - 1]$ for both lemmas Next, we assume that both lemmas hold for $t_0 \in [q_0 - n, t_f - 1]$ for some n. We then prove the induction step for Lemma 2 for $t_0 = t_f - n - 1$ using the assumption. Finally, we prove the induction step for Lemma 1 for $t_0 = t_f - n - 1$ using the result we just proved for Lemma 2 for $t_0 = t_f - n - 1$. This concludes the proof of both lemmas.

After showing that LCM is a (t_f,ℓ)-optimal strategy, we show that there is no benefit for the adversary from PoW mining while she calculates the deepest block's PoD puzzle. For this purpose, we first define useless actions we call *backward mining*, where the adversary mines a PoW puzzle that would not extend the depth of the chain:

Definition 4. *Given an execution σ, an action $Q^{\sigma}_{\mathcal{A}}(t) = b_{PoD}$ at step t is* backward mining *if the adversary already computed a partial block b'_{PoD} that extends a block in $\mathcal{T}^{\sigma}_{H}$, such that $\tilde{d}(b_{PoD}) < \tilde{d}(b'_{PoD})$ or if there is a full block b such that $\tilde{d}(b_{PoD}) = d(b)$.*

Next, we define an upgrade of the LCM strategy without useless backward mining actions.

Definition 5. *A strategy is (t_f,ℓ)-*intermittent LCM *if it is (t_f,ℓ)-optimal and does not perform backward mining actions.*

It remains to show that every LCM can be transformed to (t_f,ℓ)-intermittent LCM without affecting its optimality.

Lemma 3. *For all t_f, ℓ and an LCM strategy A, the intermittent LCM strategy A' that is identical to A, except that every backward mining action is replaced with $\perp$, is (t_f,ℓ)-optimal.*

Proof. Given a prefix π of length t_0, We look at two sets of executions. (1) Σ_A^π for a (t_f,ℓ)-optimal strategy A that for some $\sigma \in \Sigma_A^\pi$ chooses some backward mining action $Q_{\mathcal{A}}^\sigma(t_0) = b$. and (2) $\Sigma_{A'}^\pi$, where the strategy A' chooses $\perp$ at step t_0 but for $t > t_0$ it is identical to A. As before, the probability to find a block exactly at t_0 is $P_w(\mathcal{A})$. Note that this probability is not relevant for $\Sigma_{A'}^\pi$. Denoted the subset of Σ_A^π where no block is found at step t_0, by Σ_1. For all $\sigma_1 \in \Sigma_1$, there exists a unique $\sigma_2 \in \Sigma_{A'}^\pi$ such that $\mathcal{T}_b^{\sigma_1} \equiv \mathcal{T}_b^{\sigma_2}$ and vice versa. σ_1 and σ_2 agree on all random coins from t_0+1. If a block is found at step t_0, the tree depth of both executions is still equal, therefore:

$$\Pr_{\Sigma_1}[d_{\neg b}^\sigma(t_f) \geq \ell]) = \Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell]. \tag{1}$$

The block that is found in step t_0 for all $\sigma_1 \in \Sigma_A^\pi \setminus \Sigma_1$ does not extends the depth of $\mathcal{T}_b^{\sigma_1}$. Therefore, for all $\sigma_2 \in \Sigma_{A'}^\pi$, it holds that $\tilde{d}_b^{\sigma_1}(t) \leq \tilde{d}_b^{\sigma_2}(t)$ for $t \in [t_0+1, t_0+\Delta_{PoD}+1]$. The conditions of Lemma 2 thus hold, so:

$$\Pr_{\Sigma_A^\pi \setminus \Sigma_1}[d_{\neg b}^\sigma(t_f) \geq \ell] \leq \Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell]. \tag{2}$$

Using complete probability:

$$\begin{aligned}
&\Pr_{\Sigma_A^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell] = \\
&\quad P_w(\mathcal{A}) \cdot \Pr_{\Sigma_A^\pi \setminus \Sigma_1}[d_{\neg b}^\sigma(t_f) \geq \ell] + \\
&\quad (1 - P_w(\mathcal{A})) \cdot \Pr_{\Sigma_1}[d_{\neg b}^\sigma(t_f) \geq \ell]) \\
&\quad \overset{\text{Equation (1) and Equation (2)}}{\leq} \\
&\quad P_w(\mathcal{A}) \cdot \Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell] + \\
&\quad (1 - P_w(\mathcal{A})) \cdot \Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell] = \\
&\qquad \Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell].
\end{aligned}$$

Note that although $P_w(\mathcal{A})$ has no meaning in the context of $\Sigma_{A'}^\pi$, we used a simple algebraic trick to disassemble $\Pr_{\Sigma_{A'}^\pi}[d_{\neg b}^\sigma(t_f) \geq \ell]$ to two parts.

We apply the described process recursively, each time eliminating a single backward mining action. We end with a new (t_f,ℓ)-optimal intermittent LCM strategy as required by the lemma. □

We can now conclude that any intermittent LCM strategy is an optimal strategy.

Corollary 1. *Intermittent LCM is an optimal strategy.*

Proof. By Lemma 3, for all ℓ and t_f any intermittent LCM strategy is (t_f,ℓ)-optimal, so it is an optimal strategy. □

5.3 Honest tree

We now focus on the subtree $\mathcal{T}_H^\sigma(t)$. We look for a strategy that minimizes the depth of the tree.

Definition 6 (Maliciously optimal strategy)**.** *A strategy A is (t_f,ℓ)-*maliciously optimal *if, for all prefixes π of length $t_0 < t_f$ with block b, and for all strategies A', it holds that*

$$\Pr_{\Sigma_A^\pi}[d_b^\sigma(t_f) \geq \ell] \leq \Pr_{\Sigma_{A'}^\pi}[d_b^\sigma(t_f) \geq \ell].$$

We call A a maliciously optimal *if it is (t_f,ℓ)-maliciously optimal for all $t_f > t_0$ and for all ℓ.*

We consider a strategy where the attacker does not mine new blocks on $\mathcal{T}_H^\sigma(t)$ and delays any honest block by Δ, which is the maximal delay she can impose.

Definition 7. *Given a prefix π, a strategy A is a* Maximum Delay and No Mining strategy (MDNM) *if the adversary chooses to mine no blocks and to maximize the delay for the arrival of all honest blocks to be Δ for all miners.*

Denote the hash rate of the honest miners that received the block that is a tip of the chain in execution σ at step t by $\lambda^\sigma(t)$. Similarly, denote by $\lambda_m^\sigma(t)$ the total hash rate of miners that heard of blocks of depth at least m.

We show that an MDNM strategy maintains some advantages.

Lemma 4. *Given two prefixes π_1 and π_2 of length t_0, ℓ, and a MDNM strategy A, such that for all $\sigma_1 \in \Sigma_A^{\pi_1}$, $\sigma_2 \in \Sigma_A^{\pi_2}$ it holds that*

$$\forall t \in [t_0, t_0+\Delta_{PoD}] : d_b^{\sigma_1}(t) \geq d_b^{\sigma_2}(t)$$

and

$$\forall t \in [t_0, t_0+\Delta+\Delta_{PoD}] : \lambda_{d_b^{\sigma_2}(t)}^{\sigma_1}(t) \geq \lambda^{\sigma_2}(t),$$

it holds that for all $\ell > 0$ and step t_f:

$$\Pr_{\Sigma_A^{\pi_1}}[d_b^\sigma(t_f) > \ell] \geq \Pr_{\Sigma_A^{\pi_2}}[d_b^\sigma(t_f) > \ell].$$

We overview the proof with the details deferred to the full version of this paper [47].

Proof sketch. The goal is to demonstrate that a prefix with greater depth and miner engagement is more likely to maintain its lead. We prove by reverse induction.

Basis For the initial time frame $t_0 \in [t_f - \Delta_{PoD}, t_f]$, the lemma trivially holds. A deeper prefix with more miners retains its advantage in this brief period, where at most one full block can be found.

Assumption We assume the statement holds for all $t_0 \in [t_f - \Delta_{PoD} - n, t_f]$.

Induction Step Extending to the previous step $t_f - \Delta_{PoD} - n - 1$, we consider two cases: if a new block is found in the step and if not. In both cases, the leading prefix maintains its lead due to higher probabilities of block addition and mining power. □

Next, we use Lemma 4 to prove the following with the details deferred to the full version of this paper [47].

Lemma 5. *An MDNM strategy is maliciously optimal.*

Proof sketch. The goal is to show that an MDNM strategy is maliciously optimal. We prove this too by reverse induction.

Basis For $t_0 \in [t_f - \Delta_{PoD}, t_f]$, MDNM is optimal since the blockchain can grow by at most one block in this period, and MDNM is not worse than any other strategy.

Assumption We assume the lemma holds for all $t_0 \in [t_f - \Delta_{PoD} - n, t_f]$.

Induction Step Extending the proof to $t_f - \Delta_{PoD} - n - 1$, we consider two scenarios: one where an honest miner finds a new block and another where no new block is found. In both cases, MDNM retains its optimality. We apply Lemma 4 to compare the probabilities of extending the blockchain depth under MDNM and alternative strategies, confirming that MDNM is optimal. □

5.4 Basic race

We now introduce a two-epoch race where the adversary tries to fork a chain of blocks. We then find an upper bound on the probability that the adversary succeeds in the attack. From now on, we assume that $\Delta = 0$.

Given an adversarial strategy A and execution σ, we define a block $b_s^\sigma(q)$ as the first published depth-q block in σ.

We consider a race where the adversary mines a secret tree denoted by $\mathcal{T}_\mathcal{A}^\sigma(t,q)$, whose root is $b_s^\sigma(q)$ and it is a function of step t and depth q of $b_s^\sigma(q)$. We assume that the adversary did not mine any blocks before a block at depth q is published. The adversary never sends blocks from $\mathcal{T}_\mathcal{A}^\sigma(t,q)$ to the honest miners until the end of the race and they stay secret. We also consider a public subtree $\mathcal{T}_H^\sigma(t,q)$ whose blocks are public and has $b_s^\sigma(q)$ as its root. All blocks that have $b_s^\sigma(q)$ as an ancestor and are not in $\mathcal{T}_\mathcal{A}^\sigma(t,q)$ are in $\mathcal{T}_H^\sigma(t,q)$. The attacker can mine on $\mathcal{T}_H^\sigma(t,q)$ and publish the blocks at any time. Given an integer $r > 0$, we say that the adversary has won the race if, at some step t, it holds that $d(\mathcal{T}_\mathcal{A}^\sigma(t,q)) \geq q + r$ and $d(\mathcal{T}_\mathcal{A}^\sigma(t,q)) \geq d(\mathcal{T}_H^\sigma(t,q))$. Note that in this race the adversary cannot mine before a block of depth q is published by miners. Therefore, we conclude that the probability that the adversary wins the race does not depend on q.

Given a random execution $\sigma \in \Sigma_A$, we denote the probability that the adversary wins the race by $\chi(A,r)$ and the respective event by $e_\chi(A,r,q)$.

With foresight, we define the sum

$$S(r) = \max_A \sum_{i=0}^{\infty} \chi(A, r+i) \tag{3}$$

and show that $S(r) = 2^{-\Omega(r)}$. We later use this result to prove persistence and progress.

Lemma 6. *For a minority attacker ($\alpha_\mathcal{A} < \alpha_H$) and $\Delta = 0$ it holds that $S(r) = 2^{-\Omega(r)}$.*

Proof. First we bound $\chi(A,m)$.

For a block $b_s^\sigma(q)$ of depth q, we break the race into two epochs. The first epoch ends when the honest miners find m blocks and the second epoch ends when the adversary wins the race. Denote by $d_\mathcal{A}$ the number of blocks that the adversary has at the beginning of the first epoch and by $\Pr[d_\mathcal{A} = k]$ the probability of a given k.

Next, to simplify the calculations we observe that different values of Δ_{PoD} do not affect the probability that the adversary wins the race. We prove that the attacker has the same probability to win for $\Delta_{PoD} > 0$ and for $\Delta_{PoD} = 0$. Intuitively, the introduction of PoD affects both the honest and the adversary miners equally, so that we can cancel it out. This is only true for strategies that mine blocks sequentially, such as LCM mining with interruption. In this case, the number of PoD blocks is the same for both $\mathcal{T}_\mathcal{A}^\sigma(t,q)$ and $\mathcal{T}_H^\sigma(t,q)$.

We observe that given an intermittent LCM strategy A there is a strategy A' where the adversary stops mining after she finds m blocks earlier than the honest miners find their first m blocks. Note that in this scenario whether the adversary stops mining or not, she already won the race. Strategy A' has the same probability to win the race as A, as they are different only when the adversary has found m blocks, and won the race. For strategy A', the race always ends when both $\mathcal{T}_H^\sigma$ and $\mathcal{T}_\mathcal{A}^\sigma$ have identical depth. Therefore, because both the honest and the adversary trees have an identical number of PoD blocks at the end of the race, we can cancel out the effect of PoD blocks on both trees. In other words, for the strategy A' every execution where the adversary wins for $\Delta_{PoD} > 0$ has a parallel execution for $\Delta_{PoD} = 0$ with the same probability to happen. Therefore, the probability of winning for $\Delta_{PoD} > 0$ is the same as for $\Delta_{PoD} = 0$.

Denote by $P_\mathrm{w}(k)$ the probability that the attacker wins the race in the second epoch for a given k. The probability of the attacker winning is:

$$\chi(A,m) \leq \max_A \chi(A,m) = \sum_{k=0}^{m-1} P_\mathrm{w}(k) \cdot \Pr[d_\mathcal{A} = k] + \sum_{k=m}^{\infty} \Pr[d_\mathcal{A} = k]. \tag{4}$$

We use the stars and bars [48] problem in combinatorics to calculate $\Pr[d_\mathcal{A} = k] = \alpha_\mathcal{A}^k \cdot \alpha_H^m \binom{k+m-1}{m-1}$ and gambler's ruin [49] for $P_\mathrm{w}(k) = (\frac{\alpha_\mathcal{A}}{\alpha_H})^{m-k}$.

Plugging into Equation (4) we get:

$$\chi(A,m) \leq \alpha_{\mathcal{A}}^m \cdot \sum_{k=0}^{m-1} \alpha_H^k \cdot \binom{k+m-1}{m-1} + \alpha_H^m \cdot \underbrace{\sum_{k=m}^{\infty} \alpha_{\mathcal{A}}^k \cdot \binom{k+m-1}{m-1}}_{a_k} \leq \alpha_{\mathcal{A}}^m \cdot \alpha_H^{m-1} \cdot m \cdot \binom{2m-2}{m-1} + \alpha_{\mathcal{A}}^m \cdot \alpha_H^m \cdot \binom{2m-1}{m} \cdot C \leq \frac{\alpha_{\mathcal{A}} \cdot m}{\sqrt{\pi(m-1)}} \cdot (4\alpha_{\mathcal{A}}\alpha_H)^{m-1} + \frac{2 \cdot C}{\sqrt{\pi m}} \cdot (4\alpha_{\mathcal{A}}\alpha_H)^m = 2^{-\Omega(m)}, \quad (5)$$

where C is a constant that does not depend on n. We used the fact that $4\alpha_{\mathcal{A}}\alpha_H \leq 1$, the known inequality $\binom{2r}{r} \leq \frac{4^r}{\sqrt{\pi r}}$, the fact that a_H is monotonic for $k<r$ and that $\frac{b_{k+1}}{a_k} \leq 2\alpha_{\mathcal{A}}$ for $k \geq r$.

As the tail of a geometric sum decreases exponentially, we conclude that $\sum_{k=r}^{\infty} \chi(A,k) = 2^{-\Omega(r)}$, thus due to Equation (3) and Equation (5) we conclude that $S(r) \leq \sum_{i=0}^{\infty} \max_{A_i} \chi(A_i, r+i) = 2^{-\Omega(r)}$. □

5.5 Persistence and Progress

We are now ready to prove our main results. We first prove that persistence holds.

Lemma 7. *For $\Delta = 0$, and a minority attacker, persistence holds.*

Proof. Given $\sigma \in \Sigma_A$, step t_f and $r > 0$, denote by $\neg Pers(\sigma, t_0, r)$ the event that $r-$persistence is violated for some $t < t_f$ in σ. We say that a block violates $r-$persistence if it holds for this block at some step $t \leq t_f$ that the chain that includes the block is longer by at least r blocks than any chain that does not contain the block, but for some $t' > t$ the block is not in the main chain. We observe that $r-$persistence does not hold if for at least one block persistence is violated. Next, we consider a scenario where the attacker tries to violate persistence for any block at a specific depth i. Denote by $\neg Pers_i(\sigma, r)$ the event where persistence was violated for a block at depth i.

Finally, we break the race into a collection of sub-races where the adversary tries to violate persistence for a specific depth i by mining only on a specific private tree that starts at a specific honest block at depth $j < i$, denoted by $\mathcal{T}_{\neg j}^{\sigma}(t', i, r)$, we denote the event where the attacker was able to do so by $\neg Pers_i^j(\sigma, r)$. As before we denote by $\mathcal{T}_j^{\sigma}(t', i, r)$ the tree that starts at depth j and includes all blocks that are not in the adversary's private tree. Observe that persistence is violated if two conditions hold: (1) at some step t the chain that includes b at depth i in $\mathcal{T}_j^{\sigma}(t', i, r)$ is deeper by r than $\mathcal{T}_{\neg j}^{\sigma}$, and (2) at some step $t' > t$, $\mathcal{T}_{\neg j}^{\sigma}(t', i, r)$ is deeper than $\mathcal{T}_j^{\sigma}(t', i, r)$. The adversary can maximize the probability of the first condition to be 1, if she does not publish her blocks and keeps her chain private. Note to maximize the probability of the second condition the attacker should choose a strategy that maximizes $\mathcal{T}_{\neg j}^{\sigma}(t', i, r)$ and minimizes $\mathcal{T}_j^{\sigma}(t', i, r)$ for all t', as we saw in Corollary 1 and Lemma 5 LCM mining with interruptions on $\mathcal{T}_{\neg j}^{\sigma}(t', i, r)$ and MDNM mining on $\mathcal{T}_j^{\sigma}(t', i, r)$ achieve these goals respectively. Moreover, these two strategies can be used at the same time and therefore an optimal strategy to maximize the second condition is their combination. Therefore, we can think of the game where the attacker tries to maximize $\Pr[\neg Pers_i^j(\sigma, r) | \sigma \in \Sigma_A]$ as a race described in §5.4. Thus, it holds $\chi(A, r+i-j) = \Pr[\neg Pers_i^j(\sigma, r) | \sigma \in \Sigma_A]$. There are at most i possible honest blocks on the main chain before the i-th block, thus,

$$\Pr[\neg Pers_i(\sigma, r) | \sigma \in \Sigma_{A_i}] \leq \Pr[\bigcup_{j=0}^{i} \neg Pers_i^j(\sigma, r) | \sigma \in \Sigma_{A_i}]$$

Denote by t_i^{σ} the time when the first block at depth i was mined. We bound the probability that persistence does not hold for all A', q and r:

$$\Pr[\neg Pers(\sigma, t_0, r) | \sigma \in \Sigma_{A'}] \leq \sum_{i=1}^{t_f} S(r) = t_f \cdot S(r) = 2^{-\Omega(r)} \quad (6)$$

The full details of Equation (6) are in the full version of this paper [47]. We used union bound and Lemma 6.

We choose r such that: $\Pr[\neg Pers(A', t_f, r)] \leq \varepsilon$ for all A' and thus we conclude that for every ε, there exists r such that persistence holds with a probability at least $1 - \varepsilon$.

Note that because the connection between r and t_f is polynomial, as they are both polynomial in a security parameter [30], for every ε there exists large enough t_f so there is r that is significantly smaller than t_f so that the probability of persistence is at least $1 - \varepsilon$. Thus, persistence holds non-vacuously. □

Next, we prove that progress holds.

Lemma 8. *For $\Delta = 0$, and a minority attacker, progress holds.*

Proof. Given some $\varepsilon > 0$, we show that there exists δ so that progress holds with a probability larger than $1 - \varepsilon$ for all steps $t_0 \leq t_f$. To find such δ, we first denote by r a parameter, such that δ is a function of r and ε. To bound the probability that progress does not hold, we look at the first r honest blocks generated after step t_0; denote this set of blocks by $B^{\sigma}(t_0, r)$. Our goal is to find a δ such that all the blocks in $B^{\sigma}(t_0, r)$ were mined in the period $[t_0, t_0 + \delta]$ with high probability. For the adversary to exclude the blocks from $B^{\sigma}(t_0, r)$ from the main chain she has to build a chain that does not include any of them, i.e, this chain has to win the race against the tree that includes blocks from $B^{\sigma}(t_0, r)$. We look at the chain of all the ancestors of blocks in $B^{\sigma}(t_0, r)$ without the blocks themselves. Given this chain, we look at the subtrees that start at honest blocks and include only adversarial blocks. Similarly to what we did with persistence, we separate into sub-races where every race is between an adversarial subtree and the honest chain.

Given an adversarial strategy A and $\sigma \in \Sigma_A$ we denote by $\Pr[\neg Prog_q^j(\sigma,t_0,r)|\sigma\in\Sigma_A]$, the probability that the attacker was able to exclude all the first r honest blocks after the first block of depth q, denote by b_q, by building a chain that has a common honest ancestor with $B^\sigma(t_0,r)$ at depth j. We denote this common ancestor by b_j and the step when it was mined by t_j.

We denote by $\mathcal{T}_H^\sigma(t,t_0,r)$ the tree that includes all blocks from $B^\sigma(t_0, r)$, their ancestors, and their descendants. We denote by $\mathcal{T}_{\mathcal{A}}^\sigma(t,t_0,r)$ the trees that includes all the ancestors of honest block b_j and all the descendants of b_j that are not in $B^\sigma(t_0,r)$ and are not decedents of blocks in $B^\sigma(t_0,r)$.

From Corollary 1 intermittent LCM is an optimal strategy on $\mathcal{T}_{\mathcal{A}}^\sigma(t,t_0,r)$. From Lemma 5, MDNM is a maliciously optimal strategy on $\mathcal{T}_H^\sigma(t,t_0,r)$. As intermittent LCM and MDNM can be executed in parallel, and as they are both optimal and maliciously optimal respectively on their respective trees, it is guaranteed that if the adversary uses both of them on $\mathcal{T}_{\mathcal{A}}^\sigma(t,t_0,r)$ and $\mathcal{T}_H^\sigma(t,t_0,r)$, she will maximize the probability that none of the blocks in $B^\sigma(t_0,r)$ will be in the main chain forever.

Given an adversarial strategy A, denote the probability that non of the first r honest blocks after step t_0 are in the main chain forever, by $P_1(A,t_0,r)$ and the corresponding event by $e_1(\sigma,t_0,r)$. Denote by $\neg Prog_q(\sigma,t_0,r)$ the event that the first r honest blocks that were mined after b_q was published do not stay in the main chain forever. As we showed, $\Pr[\neg Prog_q^j(\sigma,r)|\sigma\in\Sigma_A]$ is maximal when the adversary's strategy A is to mine using intermittent LCM on the adversarial chain and MDNM on the honest miners' chain. There are at most q honest blocks that are ancestors of the block from $B^\sigma(t_0,r)$. Thus,

$$\begin{aligned}&\Pr[\neg Prog_q(\sigma,t_0,r)|\sigma\in\Sigma_A]\le\\ &\quad\Pr[\bigcup_{j=0}^{q}\neg Prog_q^j(\sigma,t_0,r)|\sigma\in\Sigma_A]\le\\ &\qquad\max_{A'}\Pr[\bigcup_{j=0}^{q}e_\chi(\sigma,r+j)|\sigma\in\Sigma_{A'}]\le S(r).\end{aligned}$$

Denote by $e_{\mathsf{d}}(\sigma,t_0,q)$ the event where the deepest public block before step t_0 is of depth q.

Due to complete probability and the Poisson tail bounds [50], it holds that:

$$\begin{aligned}&P_1(A',t_0,r)=\\ &\quad\sum_{q=0}^{\infty}\Pr[\neg Prog_q(\sigma,t_0,r)\wedge e_{\mathsf{d}}(\sigma,t_0,q)|\sigma\in\Sigma_{A'}]\le\\ &\quad\max_A\sum_{q=0}^{n-1}\Pr[\neg Prog_q(\sigma,t_0,r)|\sigma\in\Sigma_A]+2^{-\Omega(n-\lambda_T t)}\le\\ &\qquad\qquad\sum_{q=0}^{n-1}S(r)+2^{-\Omega(n-\lambda_T t)}.\end{aligned}$$

Therefore, we can choose n and r_1 such that for $\frac{\varepsilon}{2}$ the probability $P_1(A,t,r_1)$ that one of the next r_1 blocks will not stay in the main chain forever is smaller than $\frac{\varepsilon}{2}$.

Next, for all r, we define $\delta(r) \triangleq r^2 + r\cdot\Delta_{PoD}$, where we add Δ_{PoD} so that the period is long enough to account for the times of proof of delays puzzles. We calculate the probability $P_2(A,\delta,r)$ (with the corresponding event $e_2(A,\delta,r)$) that there are fewer than r honest blocks within a period δ using Erlang distribution. For simplicity we look at $\Delta_{PoD}=0$, as for $\Delta_{PoD}>0$ we can increase the number of steps by $r\cdot\Delta_{PoD}$ to account for the times of proof of delay:

$$\begin{aligned}&P_2(A,\delta,r)=1-\text{Erlang}(r^2;r,\lambda_H)=\\ &\quad\sum_{n=0}^{r-1}\underbrace{\frac{1}{n!}e^{-\lambda_H r^2}(\lambda_H r^2)^n}_{a_n}\overset{(1)}{\le}e^{-\lambda_H r^2}\cdot\lambda_H{}^r\cdot e^{2r\log r}=2^{-\Omega(r)}.\end{aligned}$$

For (1) we use the fact that $\frac{a_n}{a_{n-1}}\le\lambda_H r^2$ and the formula for a sum of geometric series. Thus, there is r_2 such that the probability P_2 is smaller than $\frac{\varepsilon}{2}$. We choose $r_3=\max(r_1,r_2)$. Using the union bound:

$$\begin{aligned}&\Pr[e_1(A,t,r_3)\cup e_2(A,\delta,r_3)]\le\\ &\qquad P_1(A,t,r_3)+P_2(A,\delta,r_3)\le\frac{\varepsilon}{2}+\frac{\varepsilon}{2}=\varepsilon.\end{aligned}$$

We conclude that the probability for a block in period $\delta(r_3)$ to stay in the main chain forever is at least $1=1-\varepsilon$. Therefore, at least one block that was mined in the period $[t,t+\delta(r_3)]$ will be included in the main chain forever with high probability and thus progress holds. □

We now combine both lemmas to prove our main theorem:

Theorem 1. Sprints *implements a ledger (Definition 1).*

Proof. Using Lemma 7 and Lemma 8 we can conclude that the *Sprints* fulfills the requirements in Definition 1. □

6 Implementation and Evaluation

We turn our attention to practical aspects of *Sprints*. Having addressed the scenario without network delay in our security proof in the previous section, we now investigate the scenario with network delay through experiments, aiming to assess the honest party's main chain extension rate with practical network latency.

We implement *Sprints* using Wesolowski VDF [22] for PoD. We apply optimization to reduce the network delay and dynamically adjust PoW and PoD difficulty (§6.1). Our attack threshold analysis (§6.2) and experiments (§6.3) with practical fork rates show that the security of our protocol is close (98%) to that of Bitcoin.

6.1 Implementation

We implemented a prototype of *Sprints* by modifying Bitcoin Core [51]. We made two major changes. First, we modified the data structure of block headers to include proofs of delay and adapted the mining process as well as the block validation process accordingly. Second, we modified the difficulty adjustment algorithm to adjust both the PoD and the PoW parameters.

To generate a block, a miner first calculates a VDF using the previous block hash as input. Then, the PoD is included in the block header and a standard proof of work with the new block header as input is calculated.

PoD implementation: In PoW systems, the mining algorithm is memoryless, permitting miners to dynamically alter the transactions within a block as they mine. Altering the block header, which constitutes the PoW puzzle, incurs no additional computational time, thus avoiding any notion of 'sunk cost.' In contrast, PoD is inherently not memoryless, which poses challenges for dynamic transaction inclusion. To mitigate this limitation in *Sprints*, the PoD algorithm has been designed to be independent of the block's content. Consequently, a miner needs to solve the PoD puzzle only once and can utilize that solution irrespective of subsequent modifications to the block.

We implement PoD as Wesolowski VDF [22], using the implementation provided by the POA Network [52]. To mitigate the risk of a single party gaining undue advantage through an algorithmic breakthrough in one specific VDF, a more robust implementation of PoD can require miners to solve multiple, distinct VDFs in parallel. Importantly, these VDFs would employ different, unrelated algorithms. A PoD would only be considered complete when all of these diverse VDFs have been successfully solved. This approach can reduces the likelihood that a single miner could dominate the process, as it would necessitate a breakthrough in multiple algorithms simultaneously. However, the introduction of parallel VDFs with varying algorithms presents new challenges, particularly that of difficulty adjustment. Addressing these challenges is beyond the scope of the current work and is deferred to future research.

Reducing propagation latency: In Bitcoin, blocks are verified before propagation to prevent spam. While PoW verification is quick, PoD verification is considerably slower. In our implementation, PoD verification typically takes 100ms to 500ms. Worse yet, since blocks are verified at every hop, repeated PoD verification can add significant propagation latency [53].

To address the problem, we postpone the VDF verification when propagating since PoW alone already creates a significant barrier for denial of service (rapid publication of invalid blocks). Specifically, each node that receives a block verifies the PoW, then concurrently broadcast the block and validates the PoD. A node processes a block only after the PoD puzzle was verified. Since both proofs are verified before a block is processed, this network-level optimization does not affect the logic of the consensus mechanism. Additionally, since PoW is verified before propagation, exploiting this optimization for a denial of service attack is costly and inefficient, as creating a block with valid PoW but invalid PoD is still computationally intensive.

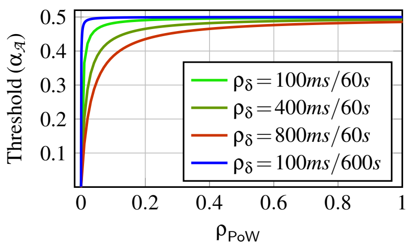


Figure 4: Attacker threshold against PoW time ratio ρ_{PoW} with different delay ratio ρ_δ.

Difficulty adjustment: We assume only principals with the fastest hardware for VDF computation participate in the protocol, but the best hardware is expected to get more efficient over time [54]. Meanwhile, the total hash power may fluctuate [55] due to hardware improvement or miners joining and leaving the system. Therefore, Sprints adjusts the difficulty parameters for PoD and PoW to keep both the block interval and the average PoW ratio, i.e., the portion of time spent on PoW, constant. It estimates PoD time and average PoW time based only on the total block generation times using the first and second moments. The details are in the full version of this paper [47].

6.2 Attacker threshold analysis

So far we analyzed *Sprints*' security without network delays, We now consider the effect of network delay on *Sprints* miners. The probability of forks is a function of the ratio between PoW time and PoD time. Intuitively, the less PoW time, the more likely for nodes to finish PoW around the same time which leads to forks.

Forked blocks are blocks not on the main chain. We refer to the ratio between forked blocks and all blocks as *fork rate*, denoted by ϕ. Conservatively assuming the adversary does not incur any forks (this means the adversary has strong control over the network which makes the result stronger), the honest mining power is reduced by $(1-\phi)$, thus the adversary threshold becomes $\alpha_{\mathcal{A}} < (1-\alpha_{\mathcal{A}})(1-\phi)$ [56], i.e., $\alpha_{\mathcal{A}} < \frac{1-\phi}{2-\phi}$.

To derive ϕ, we first calculate the probability that when a miner i mines a block at height h, another miner j also mines a block at height h before learning of i's block, creating a fork. Denote the propagation delay from miner i to miner j by T_{ij}. Let $\beta_j \in [0,1)$ denote node j's share of mining power. Assuming difficulty is well adjusted, miner j can mine a block in time Δ_{PoW} with probability β_j. Therefore, when $\Delta_{PoW} \neq 0$, node j can mine a block in time T_{ij} with probability $\frac{\beta_j T_{ij}}{\Delta_{PoW}}$. Once miner i generates a block, the mean number of forks generated by other miners is, therefore, $\sum_{j \neq i} \frac{\beta_j T_{ij}}{\Delta_{PoW}}$. Calculating the total probability over all miners i, we obtain the average number of forks n_f as follows:

$$n_f = \sum_{i=1}^{N} \beta_i \left(\sum_{j=1, j \neq i}^{N} \frac{\beta_j T_{ij}}{\Delta_{PoW}} \right). \quad (7)$$

Data propagation in a decentralized blockchain system is performed with gossip over a p2p unstructured overlay network. The propagation delay is due to the transmission of blocks over multiple hops, and we express it based on the average one-hop delay between a pair of nodes, denoted δ. For each pair of miners denote by d_{ij} the number of hops between them, so the overall propagation delay is $T_{ij} = d_{ij} \times \delta$.

Assuming all nodes have same mining power, i.e. $\beta_i = \frac{1}{N}$ ($i = 1, 2, \cdots, N$), Equation (7) could be simplified as $n_f = \frac{N-1}{N} \cdot \frac{d_{avg}\delta}{\Delta_{PoW}}$ where d_{avg} is the average number of hops between any two nodes in the network. d_{avg} is determined by the network topology. Note that we have a coefficient $\frac{N-1}{N}$ because the average distance does not take the distance to the node itself into account. The fork rate is then

$$\phi = \frac{n_f}{1+n_f} = \frac{N-1}{N\frac{\Delta_{PoW}}{d_{avg}\delta} + (N-1)}. \quad (8)$$

According to the relationship between fork rate and attack threshold, the attack threshold is

$$\alpha_{\mathcal{A}} = \frac{1}{2 + \frac{N-1}{N} \cdot \frac{d_{avg}\delta}{\Delta_{PoW}}}.$$

The fork rate and attacker threshold are thus a function of the *delay ratio* between the one-hop propagation delay and the block interval, denoted $\rho_\delta = \frac{\delta}{\Delta_{block}}$ and the *PoW ratio* between PoW mean duration and the block interval, denoted as $\rho_{\mathsf{PoW}} = \frac{\Delta_{PoW}}{\Delta_{block}}$.

Figure 4 visualizes the attacker threshold for various ρ_δ and ρ_{PoW} values. With the increase of PoW ratio ρ_{PoW} and decrease of delay ratio ρ_δ, the attacker threshold increases and approaches 0.5. For example, suppose the one-hop network latency is 100ms (approximately the average latency in Bitcoin [33]) and the block interval is 600s (i.e., $\rho_\delta = \frac{100ms}{600s}$), even if *Sprints* performs PoW only for 5% of the time ($\rho_{\mathsf{PoW}} = 0.05$), the attacker threshold is still 49% (compared to 50% in Bitcoin.)

6.3 Evaluation

We have derived an analytical relationship between attacker threshold, fork rates, and network parameters. Now we empirically validate the analysis by running *Sprints* with real-world parameters. We describe our setup (6.3.1), validate our theoretical results (6.3.2), and evaluate *Sprints* under practical parameters (6.3.3).

6.3.1 Setup

We deploy a network of 100 nodes running *Sprints* on our testbed with two 64-core AMD processors (256 hardware threads in total). Each node is given two hardware threads, so they have roughly the same mining power. Like previous work [57], we create a random topology by connecting each node to four random neighbors and we fix the topology throughout the experiments. We added network latency to outbound traffic using the Linux `tc` tool. We did not explicitly limit the bandwidth since messages sent in our experiments are small; this is representative of the nominal block size in practice (e.g., Bitcoin's Compact Blocks [58] and Prism's Proposal blocks [57]).

From propagation traces, we identified that the average number of hops of block propagation in our network is $d_{avg} = 4.5$ (See our full report [47] for details).

6.3.2 Theory Validation

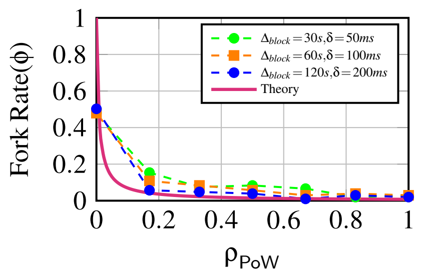
We now use our experimental setup to validate the analysis of §6.2. In all experiments, we run *Sprints* until 100 blocks are generated and calculate the fork rate from the log.

Parameter choice According to Equation (8), when N and d_{avg} are fixed, fork rates are determined by ρ_δ (network latency normalized by block interval) and ρ_{PoW} (PoW ratio). We choose $\rho_\delta \in \{\frac{100ms}{60s}, \frac{400ms}{60s}, \frac{800ms}{60s}\}$ to cover a wide range of latencies. For each ρ_δ, we run experiments with different block intervals $\Delta_{block} \in (30s, 60s, 120s)$ and thus different one-hop network latency $\delta = \rho_\delta \times \Delta_{block}$.

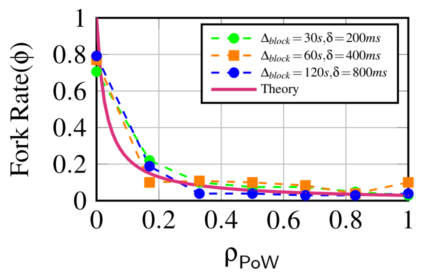
Results Figure 5 plots the results. The three subgraphs correspond to the values of ρ_δ. In all graphs, the y axis is the fork rate and the x-axis is the PoW ratio ρ_{PoW} ranging from zero to one. In each graph, there are four lines. The solid line shows fork rate by Equation (8). Markers, connected by dashed lines, show the experimental results for each block interval Δ_{block} (and thus network latency $\delta = \Delta_{block} \times \rho_\delta$). Each dot represents an experiment with 100 blocks.

We observe that the experimental results are closer to the theory with a larger one-hop delay δ. This is because the analysis only takes the network delay into account, while in reality there are other sources of delay, e.g., disk I/O, block validation, etc. These additional delays are more significant when δ is small. Nonetheless, the experimental results of each graph are close to each other, confirming that the fork rate is affected mostly by the ratio ρ_δ.

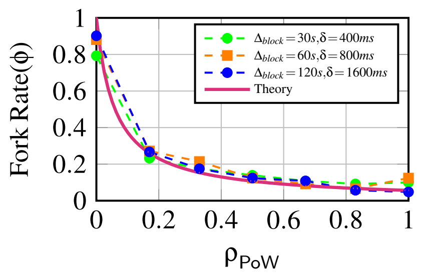
Finally, note that the measured fork rate without PoW is much smaller than the theory. From Equation (8), when taking $\Delta_{PoW} \to 0$, all the miners produce a block at about the same time. The fork rate is then $\phi = \frac{N-1}{N}$ (about one). However, in practice, variance in network delays and computation times



(a) $\rho_\delta = 100ms/60s$



(b) $\rho_\delta = 400ms/60s$



(c) $\rho_\delta = 800ms/60s$

Figure 5: Fork rate under different ρ_δ, ρ_{PoW} and Δ_{block}. Each graph corresponds to a different delay ratio ρ_δ. Solid lines plot the theoretical analysis in Equation (8). Dotted lines plot the experiment results under different Δ_{block}.

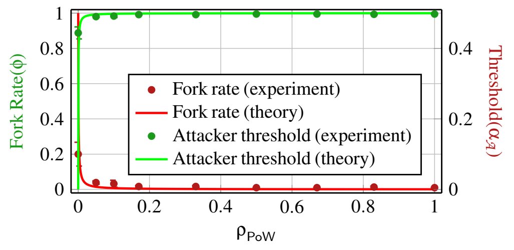


Figure 6: Fork rates and attack threshold under Bitcoin-like parameters (100ms network delay and 600s block interval.)

reduce the synchrony of block generation even with $\Delta_{PoW} = 0$, resulting in a smaller fork rate.

6.3.3 Real-world parameters

We conclude by studying *Sprints*'s behavior with practical, Bitcoin-like parameters: We measure the fork rate under $\Delta_{block} = 600s$ and the one-hop network delay $\delta = 100ms$ [33]. Figure 6 shows the results. The x axis is the ratio of PoW time ρ_{PoW}, while the left y axis shows the fork rate, and the right y axis shows the attacker threshold. The red line plots the fork rate in theory and the blue dots are the experiment results. Each dot represents 3 experiments, each with 100 blocks. The green line shows the attacker threshold derived from fork rates, and the orange dots represent the attacker threshold derived from experimental fork rates. According to Figure 6, *Sprints* achieves a good attacker threshold even with small ρ_{PoW}. For example, when $\rho_{PoW} = 0.05$, *Sprints* can withstand an attacker with $\alpha_{\mathcal{A}} = 49\%$ mining power, close to the ideal attacker threshold 50%.

6.4 Non-Negligible Propagation Delay

The measurements (§6.2) demonstrate the practicality of *Sprints* when propagation delay is negligible compared to the block interval. Future work might consider a shorter block interval, in the order of Δ, to allow for better performance [57, 59]. However, this would call for a different security analysis; our approach (§5), which reduces *Sprints* to standard PoW, becomes inapplicable in the presence of staggered PoD within each branch—specifically, when honest blocks are subjected to PoD and propagation delays. Recall that the approach presented in previous work [10, 30, 31] does not apply to *Sprints* due to its inherently non-Markovian nature.

6.5 Imperfect PoD

So far we have assumed that the adversary cannot solve the PoD faster than the honest miners. This is the main difference between the PoD and VDF primitives. Let us briefly consider a weaker model where the adversary has an *advantage* A_{adv} in solving the PoD, i.e., the adversary can solve the PoD in $A_{adv}^{-1} \cdot \Delta_{PoD}$. In practice, a perfect PoD construction does not exist, although near-perfect PoD ($A_{adv} < 1.05$) may be feasible with trusted setup and hardware [25, 26].

To estimate the impact of the advantage on the adversary's probability, we derive the adversary's probability of finding a block, i.e., the portion of blocks she mines from the total block number, denoted by P^{blk}. Denote by P^H the probability that the attacker finds a block during her head-start while other miners still calculate the PoD. From complete probability, it holds that $P^{blk} = P^H + \alpha_{\mathcal{A}} \cdot (1 - P^H)$. Using the CDF of exponential distribution, $P^H = 1 - \exp(-\frac{\alpha_{\mathcal{A}}}{\rho_{PoW}}(1 - A_{adv}^{-1}))$. Combining the two equations, we get $P^{blk} = 1 - (1 - \alpha_{\mathcal{A}}) \cdot \exp(-\frac{\alpha_{\mathcal{A}}}{\rho_{PoW}}(1 - A_{adv}^{-1}))$. This result is presented graphically in Figure 7 with $A_{adv} = 1.05$ [25] and $A_{adv} = 2$. We plot the success probability for different values of $\alpha_{\mathcal{A}}$. This result indicates that when the advantage is small it can affect the adversary's probability of success primarily when the ratio of PoW time to PoD time (ρ_{PoW}) is small and the adversary's hash rate is large. However, a large advantage of $A_{adv} = 2$ significantly increases the adversary's success probability. In this scenario, at large values of ρ_{PoW}, the attacker's advantage grows significantly, making Sprints less fair towards smaller miners. This observation underscores a practical constraint on the ρ_{PoW} parameter. System designers tradeoff minimizing carbon footprint and maintaining robust security considering different potential A_{adv} values.

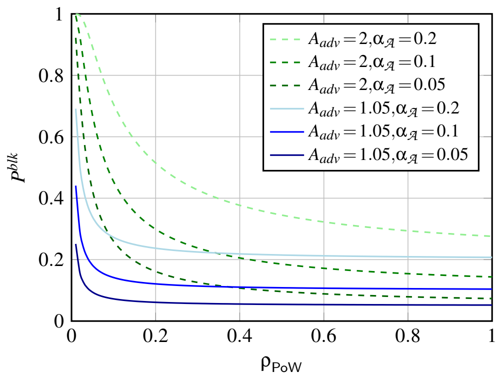


Figure 7: Attacker's block finding success probability P^{blk} against PoW time ratio ρ_{PoW} for various combinations of advantage ratios A_{adv}, attacker hash rates $\alpha_{\mathcal{A}}$.

7 Carbon Footprint Quantification

The primary objective of *Sprints* is to reduce carbon footprint while preserving the security standards that are characteristic of conventional pure Proof-of-Work (PoW) protocols. Our analysis uses conservative assumptions for a lower-bound estimate of carbon footprint reduction. To facilitate a fair comparison between *Sprints* and pure PoW, we compare them assuming the same revenue per second and profit margins for both systems and equal electricity costs so that the aggregate expenses, encompassing hardware acquisition and electricity-related costs, are equivalent in both systems. We compare the carbon footprint of the electricity and hardware that is utilized in both systems using CO2e (Carbon dioxide equivalent [35]). CO2e is a measure expressing the total impact of greenhouse gas emissions including power, hardware manufacturing, transportation and disposal in terms of the amount of CO2e that would have the same carbon footprint.

Most previous studies have focused solely on the carbon emission associated with electricity consumption (e.g., [60–63]). Another study [64] considered the carbon footprint of hardware production and disposal in a similar way to our analysis.

Although *Sprints* reduces each mining rig's electricity consumption by limiting its activity to a ρ_{PoW} portion of the time, this does not lead to a proportional reduction in the overall electricity consumption. This occurs due to the variation in the number of mining rigs between the two systems, i.e., a portion of the expenses are reallocated from electricity consumption to hardware acquisition and miners in *Sprints* purchase more mining hardware. However, the added electricity consumption due to the increased number of rigs in the system is less significant than the reduction in consumption achieved by shortening each rig's active time. For instance, when $\rho_{\text{PoW}} = 0.05$, the total electricity consumption in *Sprints* is 15.7 times lower than in pure PoW. We conservatively assume that the lifespan of mining hardware is the same in both *Sprints* and pure PoW, this is despite the thermal cycling effect that is expected to reduce the lifespan of mining hardware in *Sprints* [65, 66]. This potential reduction in hardware lifespan could further amplify *Sprints*'s carbon footprint reduction, as it would necessitate more frequent hardware replacements, thereby increasing the CAPEX costs.

While reducing electricity consumption is important, our primary objective is to decrease the overall carbon footprint. To assure this, we take into account the environmental consequences of hardware production and disposal, using data from previous studies [67–69]. Our results (Figure 2) show that despite an increase in total hardware in *Sprints*, the system's overall carbon emission decreases while taking into account the carbon footprint of hardware production and disposal. For example, when ρ_{PoW} is set to 5%, the total carbon footprint of *Sprints* is reduced by 90.8% compared to pure PoW while rare metal depletion increases by 27%. Our OPEX assumptions are conservative and don't account for peak demand-based charges, potentially underestimating *Sprints*'s efficiency [70].

In our analysis, we exclude the electricity consumption and carbon footprint of PoD computation. This is because PoD's non-parallelizable nature inherently limits its power consumption. Thus, a single PoD-optimized device per miner is expected to suffice. In fact, Ethereum Foundation's plans to develop energy-efficient, cost-effective ASICs for similar computations, such as VDFs, in the form of low-power USB sticks [71].

The full details of our analysis are presented in the full version of this paper [47].

8 Conclusion

We present *Sprints*, a hybrid PoD-PoW protocol that shifts costs from OPEX to CAPEX, decreasing the carbon footprint with almost the same security threshold as pure PoW. Moreover, we show that even when keeping the same block interval as Bitcoin and reducing the PoW portion to 5%, the security threshold is only reduced from 50% to 49%, achieving a reduction of 10.9x in resource expenditure with an increase of 1.27x in rare metal depletion.

The *Sprints* design of a hybrid PoW-PoD system can pave the way to an eco-friendly decentralized PoW blockchain protocol.

Acknowledgments

This work was supported in part by IC3 and by Avalanche Foundation.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized Business Review*, p. 21260, 2008.

[2] “Cryptocurrency prices, charts and market capitalizations.” [Online]. Available: https://coinmarketcap.com/

[3] J. Hamlin, “Big investors are finally serious about crypto. but experienced talent is still scarce.” Mar 2022. [Online]. Available: https://www.institutionalinvestor.com/article/b1x0gr2y3dzzp3/Big-Investors-Are-Finally-Serious-About-Crypto-But-Experienced-Talent-Is-Still-Scarce

[4] L. Wintermeyer, “Institutional money is pouring into the crypto market and its only going to grow,” Aug 2021. [Online]. Available: https://www.forbes.com/sites/lawrencewintermeyer/2021/08/12/institutional-money-is-pouring-into-the-crypto-market-and-its-only-going-to-grow/?sh=2660a69d1459

[5] A. Lobo, “The environmental impact of cryptocurrency,” 2022.

[6] C. Campbell, “Why china is cracking down on bitcoin mining,” Jun 2021. [Online]. Available: https://time.com/6051991/why-china-is-cracking-down-on-bitcoin-mining-and-what-it-could-mean-for-other-countries/

[7] E. Szalay, “EU should ban energy-intensive mode of crypto mining, regulator says,” *Financial Times*. *https://www.ft.com/content/8a29b412-348d-4f73-8af4-1f38e69f28cf*.

[8] I. Tsabary, A. Spiegelman, and I. Eyal, “Tuning PoW with Hybrid Expenditure,” in *3rd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2021)*, ser. Open Access Series in Informatics (OASIcs), V. Gramoli, H. Halaburda, and R. Pass, Eds., vol. 97. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 3:1–3:17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2022/15900

[9] B. Cohen and K. Pietrzak, “The Chia network blockchain,” 2019.

[10] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, “Everything is a race and nakamoto always wins,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 859–878.

[11] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[12] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” *arXiv preprint arXiv:1710.09437*, 2017.

[13] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, “Scalable and probabilistic leaderless bft consensus through metastability,” *arXiv preprint arXiv:1906.08936*, 2019.

[14] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, “Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 913–930.

[15] B. David, P. Gaži, A. Kiayias, and A. Russell, “Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[16] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[17] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, “A survey on long-range attacks for proof of stake protocols,” *IEEE Access*, vol. 7, pp. 28 712–28 725, 2019.

[18] Consensus mechanisms: Proof of stake attack and defense. Accessed: 2023-04-13. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/

[19] V. Bagaria, A. Dembo, S. Kannan, S. Oh, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, “Proof-of-stake longest chain protocols: Security vs predictability,” in *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, 2022, pp. 29–42.

[20] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, “Verifiable delay functions,” in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[21] K. Pietrzak, “Simple verifiable delay functions,” in *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[22] B. Wesolowski, “Efficient verifiable delay functions,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 379–407.

[23] D. Boneh, B. Bünz, and B. Fisch, “A survey of two verifiable delay functions.” *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[24] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass, "Continuous verifiable delay functions," in *EUROCRYPT (3)*, ser. Lecture Notes in Computer Science, vol. 12107. Springer, 2020, pp. 125–154.

[25] Y. Winetraub, E. Sagi, Y. Michalevsky, C. Künzang, and J. Gross, "Spacevdf: Verifiable delay functions using cryptographic satellites," https://grants.protocol.ai/publications/spacevdf-verifiable-delay-functions-using-cryptographic-satellites/winetraub2023.pdf, March 2023, cryptosat, Protocol Labs.

[26] C. Baum, B. David, E. Pagnin, and A. Takahashi, "Cascade:(time-based) cryptography from space communications delay," *Cryptology ePrint Archive*, 2023.

[27] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, "Bdos: Blockchain denial-of-service," in *Proceedings of the 2020 ACM SIGSAC conference on Computer and Communications Security*, 2020, pp. 601–619.

[28] N. D. Bhaskar and D. L. K. Chuen, "Bitcoin mining technology," in *Handbook of digital currency*. Elsevier, 2015, pp. 45–65.

[29] K. R. Hari, S. Y. Sai *et al.*, "Cryptocurrency mining–transition to cloud," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 9, 2015.

[30] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[31] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 281–310.

[32] Bitcoin, "Bitcoin/bitcoin: Bitcoin core integration/staging tree." [Online]. Available: https://github.com/bitcoin/bitcoin

[33] "Bitcoin propagation time," https://www.dsn.kastel.kit.edu/bitcoin/#propagation.

[34] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," in *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer, 2018, pp. 439–457.

[35] O. Rabo, "What is co2e and how is it calculated?" 11 2020, accessed: 2023-04-04. [Online]. Available: https://www.coolerfuture.com/blog/co2e

[36] Y. Huang, J. Tang, Q. Cong, A. Lim, and J. Xu, "Do the rich get richer? fairness analysis for blockchain incentives," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 790–803.

[37] J. Long and R. Wei, "Nakamoto consensus with verifiable delay puzzle," *arXiv preprint arXiv:1908.06394*, 2019.

[38] S. Deb, S. Kannan, and D. Tse, "Posat: Proof-of-work availability and unpredictability, without the work," *arXiv preprint arXiv:2010.08154*, 2020.

[39] I. Tsabary, A. Spiegelman, and I. Eyal, "Tuning pow with hybrid expenditure," *arXiv preprint arXiv:1911.04124*, 2019.

[40] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "{REM}:{Resource-Efficient} mining for blockchains," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1427–1444.

[41] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (poet)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, pp. 282–297.

[42] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[43] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 1–11.

[44] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[45] P. Gaži, A. Kiayias, and A. Russell, "Tight consistency bounds for bitcoin," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 819–838.

[46] G. Tel, *Introduction to distributed algorithms*. Cambridge university press, 2000.

[47] M. Mirkin, L. Zhou, I. Eyal, and F. Zhang, "Sprints: Intermittent blockchain pow mining," *Cryptology ePrint Archive*, 2023. [Online]. Available: https://eprint.iacr.org/2023/626.pdf

[48] P. Flajolet and R. Sedgewick, *Analytic combinatorics*. cambridge University press, 2009.

[49] J. L. Coolidge, "The gambler's ruin," *The Annals of Mathematics*, vol. 10, no. 4, pp. 181–192, 1909.

[50] C. Canonne, “A short note on poisson tail bounds.” [Online]. Available: https://github.com/ccanonne/probabilitydistributiontoolbox/blob/master/poissonconcentration.tex

[51] “Bitcoin,” https://github.com/bitcoin/bitcoin.

[52] P. Network. (2019) Verifiable delay function (vdf) implementation in rust. [Online]. Available: https://github.com/poanetwork/vdf

[53] Z. Yang, B. Qin, Q. Wu, W. Shi, and B. Liang, “Experimental comparisons of verifiable delay functions,” in *International Conference on Information and Communications Security*. Springer, 2020, pp. 510–527.

[54] D. Zhu, J. Tian, M. Li, and Z. Wang, “Low-latency hardware architecture for vdf evaluation in class groups,” *IEEE Transactions on Computers*, 2022.

[55] blockchain.com, “Total hash rate of bitcoin,” January 2023. [Online]. Available: https://www.blockchain.com/explorer/charts/hash-rate

[56] T. Roughgarden, “Foundations of blockchains,” 2021. [Online]. Available: https://timroughgarden.github.io/fob21/l/l8.pdf

[57] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, “Prism: Deconstructing the blockchain to approach physical limits,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 585–602.

[58] Bitcoin core :: Compact blocks faq. [Online]. Available: https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/

[59] H. Yu, I. Nikolić, R. Hou, and P. Saxena, “Ohie: Blockchain scaling made simple,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 90–105.

[60] V. Kohli, S. Chakravarty, V. Chamola, K. S. Sangwan, and S. Zeadally, “An analysis of energy consumption and carbon footprints of cryptocurrencies and possible solutions,” *Digital Communications and Networks*, vol. 9, no. 1, pp. 79–89, 2023.

[61] G. Wang, J. Xu, L. Ran, R. Zhu, B. Ling, X. Liang, S. Kang, Y. Wang, J. Wei, L. Ma *et al.*, “A green and efficient technology to recover rare earth elements from weathering crusts,” *Nature Sustainability*, vol. 6, no. 1, pp. 81–92, 2023.

[62] B. A. Jones, A. L. Goodkind, and R. P. Berrens, “Economic estimation of bitcoin mining’s climate damages demonstrates closer resemblance to digital crude than digital gold,” *Scientific Reports*, vol. 12, no. 1, p. 14512, 2022.

[63] M. Platt, J. Sedlmeir, D. Platt, J. Xu, P. Tasca, N. Vadgama, and J. I. Ibañez, “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, pp. 1135–1144.

[64] S. Köhler and M. Pizzol, “Life cycle assessment of bitcoin mining,” *Environmental Science & Technology*, vol. 53, no. 23, pp. 13 598–13 606, 2019.

[65] Braiins. (2021, 7) Optimizations for bitcoin mining with intermittent energy. Accessed: 2023-03-27. [Online]. Available: https://braiins.com/blog/optimizations-bitcoin-mining-intermittent-energy

[66] ANSYS, “Thermal cycling failure in electronics,” https://www.ansys.com/blog/thermal-cycling-failure-in-electronics, July 13 2022, accessed on March 27, 2023.

[67] A. de Vries, “Renewable energy will not solve bitcoin’s sustainability problem,” *Joule*, vol. 3, no. 4, pp. 893–898, 2019.

[68] C. L. Weber, “Uncertainty and variability in product carbon footprinting: Case study of a server,” *Journal of Industrial Ecology*, vol. 16, no. 2, pp. 203–211, 2012.

[69] M. Stutz, “Product carbon footprint (pcf) assessment of a dell optiplex 780 desktop–results and recommendations,” in *Towards life cycle sustainability management*. Springer, 2011, pp. 495–500.

[70] A. Engineering, “Maximum demand indicator,” 2023, accessed: 2023-04-04. [Online]. Available: https://www.allumiax.com/blog/maximum-demand-indicator

[71] D. Khovratovich, M. Maller, and P. R. Tiwari, “Not so slowth: Invertible vdf for ethereum and others,” 2021.