Parallel Proof-of-Work with Concrete Bounds

Patrik Keller
patrik.keller@student.uibk.ac.at
University of Innsbruck
Austria

Rainer Böhme
rainer.boehme@uibk.ac.at
University of Innsbruck
Austria

ABSTRACT

Authorization is challenging in distributed systems that cannot rely on the identification of nodes. Proof-of-work offers an alternative gate-keeping mechanism, but its probabilistic nature is incompatible with conventional security definitions. Recent related work establishes concrete bounds for the failure probability of Bitcoin's *sequential* proof-of-work mechanism. We propose a new family of state replication protocols that use *parallel* proof-of-work. Our bottom-up design from an agreement sub-protocol allows us to give concrete bounds for the failure probability in adversarial synchronous networks. State updates can be sufficiently secure to support commits after one block, removing the risk of double-spending in many applications. We offer guidance on the optimal choice of parameters for a wide range of network and attacker assumptions. Simulations show that the proposed construction is robust even against partial violations of our design assumptions.

ACM Reference Format:
Patrik Keller and Rainer Böhme. 2022. Parallel Proof-of-Work with Concrete Bounds. In *4th ACM Conference on Advances in Financial Technologies (AFT '22), September 19–21, 2022, Cambridge, MA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3558535.3559773

1 INTRODUCTION

Bitcoin's use of proof-of-work puzzles to secure state replication *without* relying on the identification of nodes was praised as a technical novelty [2]. While initially supported with heuristic arguments [52], the security of the so-called Nakamoto consensus has been analyzed rigorously over the past decade [20, 30, 32, 45, 56]. All of these works prove *asymptotic* security in various models. Only recently, in AFT '21, Li et al. [50] gave *concrete* security bounds for the failure probability in adversarial synchronous networks. While asymptotic bounds establish that a protocol is secure if one waits "long enough," concrete bounds tell users *how* long they have to wait before accepting a state update as final. All major threats against Bitcoin's security, including double-spending and selfish mining, exploit this uncertainty in some way or another [15, 25, 34, 42].

Nakamoto consensus uses *sequential* proof-of-work: each puzzle refers back to exactly one previous puzzle solution (Fig. 1, left half). A number of *non-sequential* proof-of-work protocols deviate from this scheme to improve throughput or mitigate known security threats [10, 46, 64, 66]. The approaches seem promising, but their design is heuristic. For example, Bobtail [10] argues that multiple

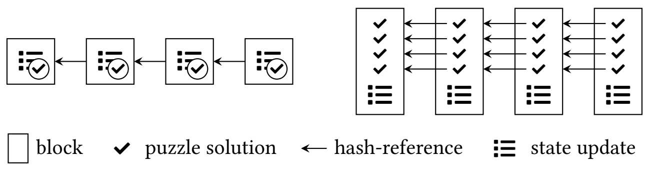


Figure 1: Schematic comparison of sequential (Bitcoin, left) and parallel (proposed, right) proof-of-work blockchains.

puzzles per block imply more regular block intervals which makes it harder for attackers to double-spend. All proposals lack security proofs, let alone concrete bounds. Therefore, one fundamental question remains open: can non-sequential proof-of-work improve the security of state replication?

This work proposes a principled construction of state replication. We start from the assumptions established by the literature on sequential proof-of-work [20, 30, 32, 45, 50, 56]. We then show how agreement on the latest state can be reached with bounded worst-case failure probabilities. By repeating the agreement procedure, we obtain a family of replication protocols that inherit the concrete error bound. The proposed scheme uses k independent puzzles per block; we thus call it *parallel* proof-of-work (Fig. 1, right half).

To showcase the advantage of parallel proof-of-work, we evaluate a protocol instance that uses $k = 51$ puzzles per block while maintaining Bitcoin's expected block interval of 10 minutes. It guarantees consistency *after one block* up to a defined failure probability of $2.2 \cdot 10^{-4}$ for an attacker with 25 % compute power and two seconds worst-case message propagation delay (cf. Table 3 below in Sect. 2.6). Attacking consistency, e. g., by rewriting an already confirmed block, requires spending work on thousands of blocks without success. For comparison, the optimal configuration of sequential proof-of-work, a "fast Bitcoin" with 7 blocks per minute, has a failure probability of 9 % in the same conditions [50].[1] An attacker would succeed once in roughly every 2 hours.

This paper makes several contributions. We define a family of proof-of-work agreement protocols $\mathcal{A}_k$, provide upper bounds for the worst-case failure probability, and show how to find optimal parameters for different attacker and synchrony parameters. Then we construct a family of replication protocols $\mathcal{B}_k$, which invoke $\mathcal{A}_k$ iteratively to secure a blockchain. We implement $\mathcal{B}_k$ and evaluate it for robustness and security within and beyond the design assumptions using network simulation. We parametrize our simulations to allow for a direct comparison to sequential proof-of-work as used in Bitcoin. We offer guidance on how to parametrize $\mathcal{B}_k$ for other settings. For replicability and future research, we make the protocol and simulation code available online [44].

[1]Bitcoin as deployed is clearly worse. Li et al. [50] do not even provide a failure probability for the common rule of thumb to wait six blocks (1 hour).

The paper is organized along these contributions. Section 2 presents and analyzes the agreement protocol. We specify the replication protocol in Section 3 and evaluate it in Section 4. We discuss the relation to the relevant literature, limitations, and future work in Section 5. Section 6 concludes.

2 PROOF-OF-WORK AGREEMENT

Agreement protocols allow their participants to unambiguously agree on a single value [59]. We consider a family of agreement protocols $\mathcal{A}_k$ where the participants cast votes for the value with the most existing votes until k votes exist for the same value. We rate-limit using proof-of-work: each vote requires a puzzle solution. The votes are independent, hence puzzles can be solved in parallel.

In the remainder of this section, we state our model for distributed systems and proof-of-work (Sec. 2.1), we specify $\mathcal{A}_k$ (Sec. 2.2), and evaluate its security considering worst-case message scheduling (Sec. 2.3) as well as adversarial behavior (Sec. 2.4). We also guide the choice of parameters (Sec. 2.5) and compare to the best known security bounds for sequential proof-of-work (Sec. 2.6).

2.1 Model

We describe an environment that simulates the execution of $\mathcal{A}_k$ over continuous time. We encode assumptions on computation, communication, and proof-of-work by explicitly defining some aspects of the environment. In other places, we give room for worst-case behavior. For example, we do not restrict the number of participating nodes to reflect a "permissionless" system.

2.1.1 Event-Based Computation. We specify the protocol as a set of event-handlers. The environment maintains the local state for each participating node and it invokes event-handlers for initialization, proof-of-work, and message delivery. Each invocation takes place at a single point in time. We write $\langle$ event $\rangle$ for events without associated data and $\langle$ event | data $\rangle$ otherwise.

The environment invokes the $\langle$ init | x $\rangle$ handler for each node at time 0. The initialization values x can be different for each node. When a node A invokes the procedure TERMINATE(x), the value x becomes A's local result of the agreement protocol. The environment will stop invoking further event-handlers for A. We say A terminated with return value x.

2.1.2 Communication. We adopt the Δ-synchronous communication model from prior analyses of Nakamoto consensus [20, 32, 45, 50, 56]. Nodes share information by calling the BROADCAST procedure. If a node A calls BROADCAST(m) at time t, then the environment invokes the $\langle$ deliver | m $\rangle$ handler on each receiving node $B \neq A$ at or before time $t + \Delta$. This reflects a setting where a network-level attacker can delay any message for up to Δ time.

2.1.3 Proof-of-Work. We adopt the continuous time model for proof-of-work from prior analyses of Nakamoto consensus [20, 50, 60, 66]: the environment has a proof-of-work mechanism P_λ that activates nodes at random times. In stochastic terms, P_λ is a homogeneous Poisson process with rate λ. For this presentation, it is instructive to describe P_λ as a stochastic clock, where the delays between consecutive ticks are independent and exponentially distributed with rate parameter λ. We define $\bar{d} = 1/\lambda$ for the expected delay between consecutive ticks.

Algorithm 1 Agreement protocol $\mathcal{A}_k$

```
upon event ⟨ init | $x$ ⟩ do
    $p \leftarrow x$                                    ▷ preferred value
    for $y \in \mathbb{N}$ do votes($y$) ← 0
upon event ⟨ activation ⟩ do       ▷ proof-of-work, see Sec. 2.1.3
    BROADCAST(vote $p$)
    votes($p$) ← votes($p$) + 1
upon event ⟨ deliver | vote $x$ ⟩ do
    votes($x$) ← votes($x$) + 1
    if votes($x$) > votes($p$) then $p \leftarrow x$
upon $\exists x$ | votes($x$) $\geq k$ do
    TERMINATE($x$)
```

Let t_i denote the time of the i-th tick. The environment activates exactly one node per tick at the corresponding time t_i by invoking the $\langle$ activate $\rangle$ handler. We call this invocation the i-th activation.

In practice, proof-of-work is implemented using hash-based cryptographic puzzles that have to be solved by trial-and-error. Attackers cannot choose *which* node finds the next solution. In our model, the environment may choose which node is activated at each tick. Thereby, we eliminate one source of randomness and replace it with a worst-case realization.

2.2 Protocol $\mathcal{A}_k$

We specify $\mathcal{A}_k$ in Algorithm 1. During initialization, each node sets the preferred value (ln. 2) and initializes the vote counters to zero for all values (ln. 3). Whenever a node is activated by the environment (through proof-of-work), it broadcasts a vote for its preferred value and updates the vote counter accordingly (ln. 5-6). All nodes count the received votes and update their preference to the value with the highest counter (ln. 8-9). After receiving the k-th vote for a value x, the nodes terminate returning x (ln. 11).

Similar protocols exist in the literature on Byzantine Fault Tolerance (BFT). Typically, they rely on round-based execution where each node is activated once per round. Such rate-limiting requires strong identification of all participating nodes. We eliminate this requirement by sourcing the activations from proof-of-work [2].

2.3 Security Against Network-Level Attacks

Good agreement protocols ensure that all nodes terminate (liveness) and that they terminate with the same value (safety) [29]. We analyze how the choice of k affects the liveness and safety of $\mathcal{A}_k$. Our analyses depend on the parameters of the environment, i. e., the maximum propagation delay Δ and the proof-of-work rate λ. For now, we assume that all nodes follow the protocol. We analyze adversarial behavior in Section 2.4.

Liveness is straightforward. For n nodes, there are at most n different initialization values. After $n \cdot k$ activations, at time $t_{n \cdot k} + \Delta$ more specifically, there must be one value for which each of the nodes has at least k votes. This implies termination of all nodes.

Safety is straightforward for the special case $\Delta = 0$. Message broadcast and the corresponding message delivery happen at the same time. At the first activation t_1, the activated node broadcasts a vote for its preferred value. The other nodes receive it immediately

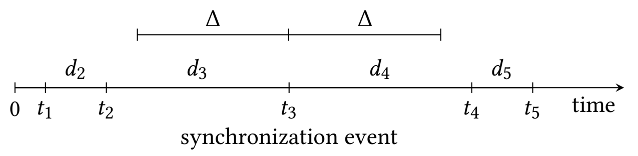


Figure 2: Activation times t_i and activation delays d_i for one particular realization of P_λ. The third activation is a synchronization event.

and update their preferred value accordingly. From then on, all nodes stay synchronized as they keep voting for the same value. At time t_k all nodes have k votes for the same value and terminate.

For $\Delta > 0$, safety becomes more involved. The broadcast of one vote might overlap with the next activation, two votes might cancel out, and the synchronization can be delayed. In order to analyze these failure modes, we first define our notion of safety.

Definition 1 (Failure)**.** We say inconsistent termination or inconsistency failure, if an execution results in two or more nodes terminating with different values.

Definition 2 (Safety)**.** $\mathcal{A}_k$ is ε-safe for a given parametrization (Δ, λ) of the environment, if the probability that $\mathcal{A}_k$ executed in the environment results in inconsistent termination is at most ε. Probabilities are taken over the realization of the stochastic clock P_λ. Initialization values, message propagation delays, and choice of activated nodes are set to the worst case for the given realization.

To show ε-safety, we first argue that certain realizations $\{t_i\}$ of the random activation times imply synchronization on a single preferred value. We then measure the probability of such realizations.

Definition 3 (Activation delay)**.** Let $\{t_i\}$ be a realization of the random activation times and let $t_0 = -\infty$. We define $d_i = t_i - t_{i-1}$. We call d_i the i-th activation delay.

Definition 4 (Synchronization event)**.** We say that t_i is a synchronization event, if both $d_i > \Delta$ and $d_{i+1} > \Delta$.

Figure 2 illustrates these definitions for one realization. Similar concepts were previously called *uniquely successful round* [30], *convergence opportunity* [56], and *loner* [20, 50].

Proposition 1. *If t_i is a synchronization event, then all running nodes prefer the same value at time $t_i + \Delta$.*

Proof. Let $d_i > \Delta$ and $d_{i+1} > \Delta$. This restriction imposes the following order of events,

$$t_{i-1} < t_{i-1} + \Delta < t_i < t_i + \Delta < t_{i+1}\ . \tag{1}$$

Observe that the first $i-1$ votes are fully propagated at the time of the i-th activation. Just before the i-th activation, all nodes see the same votes. The order of votes does not matter. If nodes prefer different values, then there is a tie between these preferred values.

One node is activated at time t_i and votes for its preferred value x. The other nodes receive the vote until $t_i + \Delta$. Receiving nodes that prefer x leave their preference unchanged. Receiving nodes that prefer a different value adopt x because the new vote is breaking the tie. Activation $i+1$ happens later, thus there is no other vote that can interfere. □

Table 1: Discrete state transitions modelling Proposition 2.

state after i transitions		state after $i \to i+1$	
#	interpretation	$d_{i+2} \le \Delta$	$d_{i+2} > \Delta$
s_1	$d_{i+1} \le \Delta \wedge$ no synchronization event	s_1	s_2
s_2	$d_{i+1} > \Delta \wedge$ no synchronization event	s_1	s_3
s_3	$\exists\, j \le i \mid t_j$ is synchronization event	s_3	s_3

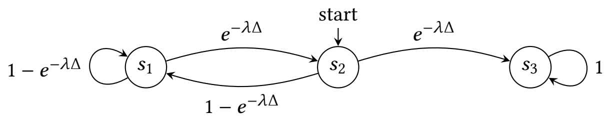

$$\begin{tikzcd}
s_1 \arrow[loop left, "1-e^{-\lambda\Delta}"] \arrow[r, bend left, "e^{-\lambda\Delta}"] & s_2 \arrow[l, bend left, "1-e^{-\lambda\Delta}"] \arrow[r, "e^{-\lambda\Delta}"] & s_3 \arrow[loop right, "1"]
\end{tikzcd}$$

Figure 3: Graphical representation of the Markov chain.

Proposition 2. *Let $\{t_i\}$ be a realization where the first synchronization event happens before t_{2k}. Then all nodes running $\mathcal{A}_k$ return the same value.*

Proof. Two nodes terminating with different values requires at least $2k$ votes (Alg. 1, l. 10). Let t_i denote the first synchronization event in $\{t_i\}$ and let $i < 2k$. At time t_i, less than $2k$ votes exist and all nodes are aware of all existing votes. If one node has terminated returning x, then all nodes have terminated returning x. Otherwise, all nodes are still running. By Proposition 1 all nodes prefer the same value y at time $t_i + \Delta$. Nodes activated at or after t_{i+1} will vote for y until all nodes terminate returning y. □

Proposition 2 provides a sufficient condition for consistency which depends on the realization of the stochastic clock. To measure the space of realizations satisfying this condition, we construct a discrete Markov chain with three states. The random state transitions happen at the ticks of the stochastic clock P_λ. Before the first synchronization event, we use two states to track whether the last delay was greater than Δ (state s_2) or not (state s_1). The model enters the terminal state s_3 if two consecutive delays are greater than Δ. Since $d_1 = \infty$ by Definition 3, we set the start state to s_2. By construction, the Markov chain is in state s_3 after i transitions if and only if there was a synchronization event at or before time t_i. Table 1 describes the states and transitions more formally.

For $i > 1$, the activation delays d_i are independent and exponentially distributed with rate λ by the definition of the stochastic clock (Sec. 2.1.3). The probability that $d_i \le \Delta$ is $1 - e^{-\lambda\Delta}$. This gives us the Markov chain depicted in Figure 3.

Proposition 3 (Safety). *$\mathcal{A}_k$ is $b_0(\lambda, \Delta, k)$-safe for*

$$\boldsymbol{M}(\lambda, \Delta) = \begin{pmatrix} 1-e^{-\lambda\Delta} & e^{-\lambda\Delta} & 0 \\ 1-e^{-\lambda\Delta} & 0 & e^{-\lambda\Delta} \\ 0 & 0 & 1 \end{pmatrix}, \tag{2}$$

$$\boldsymbol{v} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix},\ \textit{and} \tag{3}$$

$$b_0(\lambda, \Delta, k) = 1 - \left(\boldsymbol{v} \times \boldsymbol{M}(\lambda, \Delta)^{2k-1}\right)[3]\,, \tag{4}$$

where [3] *denotes selection of the third element.*

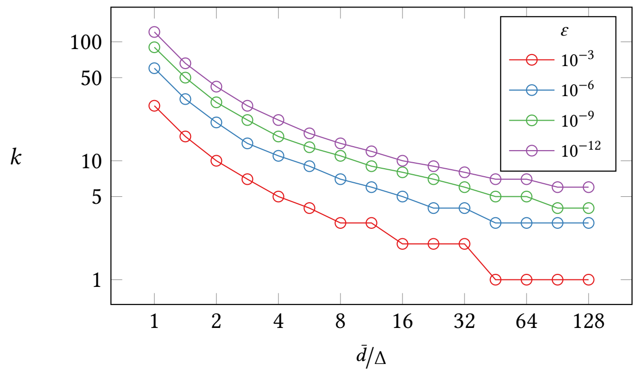


Figure 4: Minimum k, such that $\mathcal{A}_k$ is ε-safe by Proposition 3.

Proof. $\mathbf{M}(\lambda,\Delta)$ describes the Markov chain depicted in Figure 3 in matrix form. We assign vector $\mathbf{v}$ to initialize the Markov model in state s_2. The third element of the result of $\mathbf{v}\times\mathbf{M}(\lambda,\Delta)^{2k-1}$ describes the probability that the Markov model is in state s_3 after $2k-1$ random transitions. Our claim follows by the construction of the Markov chain, Definition 2, and Proposition 2. □

Observe that b_0 depends only on the vote threshold k and the product of λ and Δ. It is instructive to interpret λ as the inverse of the expected activation delay $\bar{d}$ and use the synchrony parameter Δ as a unit of time. Figure 4 visualizes the bound $b_0(\lambda,\Delta,k)$ for different combinations of $\bar{d}$ (as multiple Δ) on the x-axis and k on the y-axis. Both parameters have a positive effect on the safety of $\mathcal{A}_k$.

Proposition 3 guarantees ε-safety for $\mathcal{A}_k$ against strong network-level attacks. We now extend the argument to adversarial voting.

2.4 Security Against Malicious Voting

We now consider attackers who can cast votes. While equivocation is a major concern in the setting with identified nodes, the use of proof-of-work in $\mathcal{A}_k$ completely removes this issue [2]: in practice, every vote is authenticated with a costly solution to a computational puzzle [22]. Therefore, the only remaining attack strategy is to withhold votes to release them later. We address this next.

2.4.1 Attacker Votes. We assume that the attacker controls at most α of the total proof-of-work capacity.[2] We model this by splitting the stochastic clock P_λ in two independent parts P_A and P_D. The clock P_A ticks at rate $A=\alpha\cdot\lambda$. With each tick, the attacker gains one *attacker vote* which can be withheld arbitrarily before being sent. The clock P_D ticks at rate $D=(1-\alpha)\cdot\lambda$ and results in activations of defender nodes which follow the protocol as specified.

Recall that the sum of multiple Poisson processes is another Poisson process with cumulated rate. The probability that an individual tick of $P_\lambda = P_{A+D}$ results in an attacker vote is α.

2.4.2 Malicious Voting. We study withholding attacks by considering two phases. In the **balancing** phase nodes are not yet synchronized on the same preferred value. Recall that without malicious voting, the nodes would synchronize at the first synchronization

[2]The analysis generalizes to multiple attackers: we conservatively assume that all attackers coordinate perfectly and sum up their shares.

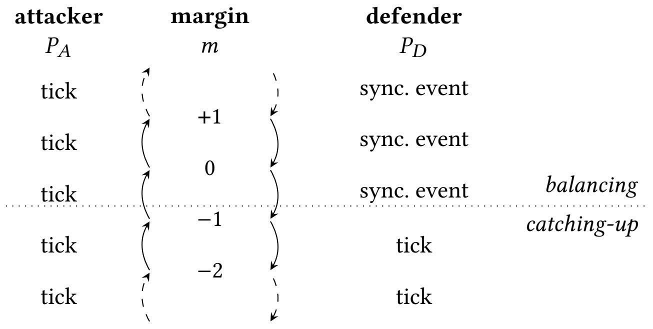


Figure 5: Security of $\mathcal{A}_k$: attacker's margin and the transition between *balancing* and *catching-up* phases.

event (see Prop. 1). A vote-withholding attacker can prevent synchronization by releasing withheld votes around synchronization events. The balancing phase continues while the attacker can balance synchronization events with his stock of withheld votes. If the attacker does not release a withheld vote around a synchronization event, e. g. because his stock is empty, the nodes synchronize. This is when the attack transitions to the *catching-up* phase.

During the **catching-up** phase, all nodes prefer the same value. With each tick of the stochastic clock P_D, the nodes cast one vote for this value and thereby reinforce the synchronization. However, the attacker can destroy the synchronization by releasing sufficiently many votes for a different value. If this happens, the attack transitions back to the *balancing* phase.

Both phases can be characterized with an integer depth. In the *balancing* phase it matters how many votes are currently withheld by the attacker. In the *catching-up* phase it matters how many votes the attacker has to cast in order to destroy the synchronization. Our attack model tracks these depths in a single **margin** variable m. Positive m represent withheld votes during *balancing* and negative m represent number of votes to be *caught-up* (see Fig. 5).

2.4.3 Markov Chain Model. We quantify the safety of $\mathcal{A}_k$ against malicious voting by measuring the space of realizations of the joint clock P_λ where withholding and propagation delays enable inconsistent termination. We generalize the Markov chain model of Section 2.3 to include the states for different margins m. The new state space is $(m,s)\in\mathbb{Z}\times\{\perp,\top\}$. To track the occurrence of synchronization events, we set $s=\top$ if and only if the last activation delay was greater Δ. The initial state is $(0,\top)$: zero votes are withheld and $s=\top$ since $d_1>\Delta$ by Definition 3.

As before, transitions happen at each tick. For attacker votes (probability α), we increment m. Depending on the phase, the attacker withholds (increasing the stock of withheld votes) or catches up by one; both map to the same transition. For defender activations (probability $1-\alpha$), we distinguish the two phases. In the *balancing* phase ($m\geq 0$), we only decrement m if the tick is a synchronization event. In the *catching-up* phase ($m<0$), we always decrement m. Figure 6 illustrates the state transitions and transition probabilities.

2.4.4 Numerical Solution. In principle, within l steps, the model can reach any state with $-l\leq m\leq l$. Calculating the exact state probabilities after l transitions requires us to raise a square matrix

$$\begin{tikzcd}
m \geq 0, s \arrow[r, bend left, "\alpha", "\text{attacker withholds}"'] \arrow[r, bend right, "(1-\alpha)\,(1-e^{\lambda\Delta})", "d_{i+1} \leq \Delta"'] & m+1, s \\
 & m, \bot \\
m \geq 0, \bot \arrow[r, "(1-\alpha)\,e^{\lambda\Delta}", "d_{i+1} > \Delta"'] & m, \top \\
m \geq 0, \top \arrow[r, "(1-\alpha)\,e^{\lambda\Delta}", "\text{synchronization event}"'] & m-1, \top
\end{tikzcd}$$

balancing phase

catching-up phase

$$\begin{tikzcd}
 & m+1, s \\
m < 0, s \arrow[ur, bend left, "\alpha", "\text{attacker catches up by one}"'] \arrow[r, "(1-\alpha)\,(1-e^{\lambda\Delta})", "d_{i+1} \leq \Delta \text{ consistent vote}"'] \arrow[dr, bend right, "(1-\alpha)\,e^{\lambda\Delta}", "d_{i+1} > \Delta \text{ consistent vote}"'] & m-1, \bot \\
 & m-1, \top
\end{tikzcd}$$

Figure 6: Generalized Markov chain model for attackers who can send and withhold votes. Transition probabilities are annotated above the arrows and interpretations below.

with $n = 2(2l+1)$ rows to the power of l. Each matrix multiplication is $O(n^{2.8})$ [67]. Thus, the analysis is infeasible for larger l.

We set a cut-off at $m = \pm 25$ to make the problem tractable. We assume that an attacker who manages to withhold 25 votes during the *balancing* causes inconsistent termination. Similarly, an attacker lagging behind 25 votes in the *catching-up* phase cannot catch up at all. With these assumptions, the number of states is bounded by 102 and the matrix multiplications stay tractable. Using such cut-offs is common practice in the related literature [34, 61].

A second simplification in this model is that it does not track how many votes are cast for each value. Adding this information would blow up the state space excessively. We work around this problem by ignoring the termination rule of $\mathcal{A}_k$ and assume that the nodes continue voting forever. We thus need to rephrase our notions of success and failure for the purpose of this analysis.

Recall that inconsistent commits require at least $2k$ votes. We count an execution as successful if all nodes prefer the same value after $2k$ steps. This is easy to check by inspecting the phase after $2k$ transitions: *catching-up* means success and *balancing* means failure.

We calculate the failure probability of $\mathcal{A}_k$ for different combinations of α, Δ, $\lambda = 1/\bar{d}$, and k by exponentiation of the probability matrix of the generalized Markov chain model. We visualize this in Figure 7, following the setup of Figure 4, but with more lines for different assumptions of attacker strength α. As expected, increasing α pushes up the required k for any given failure bound ε and expected activation delay $\bar{d}$. For example, assuming a proof-of-work puzzle takes 8 times the maximum propagation delay, while without attacker, $k = 3$ were sufficient for 10^{-3}-safety, k must increase to $k = 9$ if an attacker is present and controls 10 % of the proof-of-work capacity; or to $k = 88$ for 33 % attacker strength.

In practice, a protocol designer can adjust the puzzle difficulty and should care about the protocol runtime, to which we turn next.

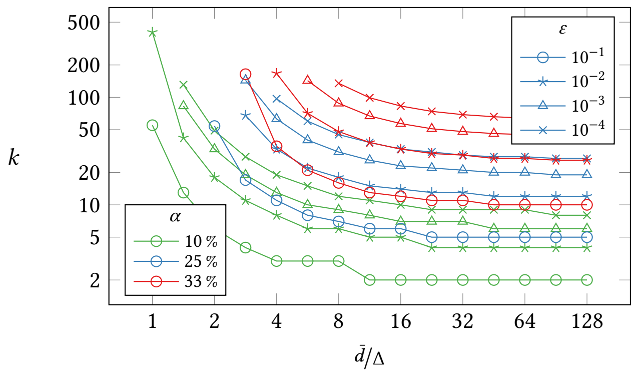


Figure 7: Minimal k such that $\mathcal{A}_k$ satisfies the given failure probability bound ε for a given attacker and expected activation delay $\bar{d}$ as multiple of Δ.

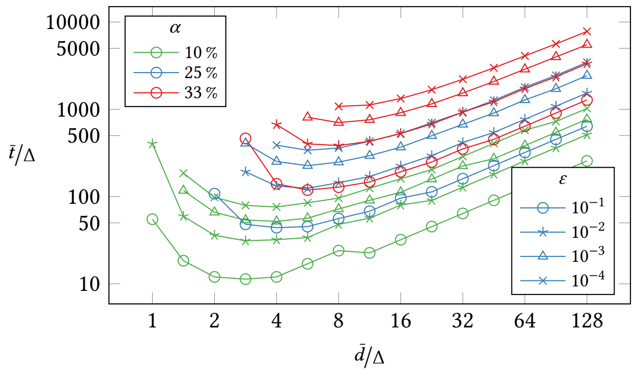


Figure 8: Protocol runtime after choosing the minimal k such that $\mathcal{A}_k$ satisfies the given failure probability bound ε for a given attacker and expected activation delay $\bar{d}$. Both axes show times as multiples of Δ.

2.5 Choosing Efficient Parameters

The aim here is to guide the choice of protocol parameters to minimize the protocol runtime for given assumptions about the real world. $\mathcal{A}_k$'s failure probability depends on the synchrony parameter Δ, the proof-of-work rate λ, the attackers compute power α, and the threshold k. The protocol operator can choose λ and k, while Δ and α are worst-case assumptions. Safety increases with k or by decreasing λ. But both options slow down termination: either we wait for more votes or we wait longer for each vote.

Recall that the protocol runtime is stochastic. Termination requires k votes for the same value and thus at least k activations. The time of the k-th activation is the sum of k exponentially distributed delays, i. e., gamma distributed with shape parameter k. Whenever nodes vote for different values—due to network delays or withholding—termination requires more activations and the shape parameter of the gamma distribution increases.

We optimize the protocol runtime for the optimistic case where k activations enable termination. We call $\bar{t} = k \cdot \bar{d} = k/\lambda$ the optimistic expected protocol runtime. Figure 8 shows $\bar{t}$ (in multiples of Δ) for

Table 2: Configurations of $\mathcal{A}_k$ minimizing protocol runtime $\bar{t}$.

α	ε	$\bar{d}/\Delta$	k	$\bar{t}/\Delta$
0	10^{-1}	3.0	2	6
0	10^{-2}	2.6	5	13
0	10^{-3}	2.5	8	20
0	10^{-4}	2.7	10	27
$1/10$	10^{-1}	2.8	4	11
$1/10$	10^{-2}	3.3	9	29
$1/10$	10^{-3}	3.2	16	51
$1/10$	10^{-4}	3.8	20	75
$1/4$	10^{-1}	4.0	10	40
$1/4$	10^{-2}	5.1	24	123
$1/4$	10^{-3}	5.7	40	226
$1/4$	10^{-4}	5.8	58	339
$1/3$	10^{-1}	6.1	19	115
$1/3$	10^{-2}	7.4	51	375
$1/3$	10^{-3}	7.9	88	699
$1/3$	10^{-4}	8.8	121	1067

We set attacker α and failure bound ε. We optimize activation delay $\bar{d}$ and threshold k. The synchrony parameter Δ serves as unit of time.

the same parameters as used in Figure 7. Observe that depending on ε and α, different activation delays $\bar{d}$ minimize the protocol runtime.

As the curves in Figure 8 are neither convex nor continuous and expensive to evaluate, we identify the minima using Bayesian optimization [54] and report them in Table 2.

Now we see that in the above example ($\varepsilon = 10^{-3}$, $\alpha = 1/10$), the protocol runtime can be reduced from $\bar{t} = 72\,\Delta$ to $51\,\Delta$ without sacrificing safety by choosing $(k, \bar{d}) = (16, 3.2\,\Delta)$ instead of $(9, 8\,\Delta)$. For perspective, with a network latency bound $\Delta = 2$ seconds, the puzzle difficulty should be adjusted to one solution every 6.4 seconds. The protocol $\mathcal{A}_{16}$ would terminate in about 102 seconds.

2.6 Comparison to Sequential Proof-of-Work

In their contribution to AFT '21, Li et al. [50] provide concrete bounds for the failure probability of Nakamoto consensus. The "achievable security latency function" $\bar{\epsilon}(t)$ for Nakamoto consensus as stated in [50, Theorem 3.5] provides an upper bound for the failure probability after waiting for a given confirmation time t. Our models and assumptions are compatible, but we derive failure probabilities after termination and hence after stochastic runtime.

To enable comparison of sequential and parallel proof-of-work, we fix the time frame to $\bar{t} = 10$ minutes. We also fix the attacker α and propagation delay Δ. For parallel proof-of-work, we optimize k for minimal failure probability of $\mathcal{A}_k$ subject to $\bar{d} \cdot k = \bar{t}$. The resulting configuration exhibits an expected protocol runtime of 10 minutes. For sequential proof-of-work, we optimize the block interval $\bar{d}_{\text{seq}}$ for minimal failure probability after 10 minutes.

Table 3 compares the failure probability ε of $\mathcal{A}_k$ with the achievable security ε_{seq} of Nakamoto consensus for various combinations of Δ and α. Note that Li et al. [50] do not define $\bar{\epsilon}(t)$ for all combinations of α and Δ. We omit the undefined values from the table. Observe that our concrete bounds for parallel proof-of-work consistently outperform the bounds for sequential proof-of-work by at least two orders of magnitude.

Remark 1. The true advantage is smaller because Li et al. [50] consider a long-running blockchain protocol, whereas our agreement protocol has a limited time horizon. Moreover, concurrent work presents improved bounds for sequential proof-of-work [33, 37].

Table 3: Advantage of parallel over sequential proof-of-work.

Parameters		Parallel			Sequential [50]		
Δ	α	k	$\bar{d}$	ε	$\bar{k}_{\text{seq}}$	$\bar{d}_{\text{seq}}$	ε_{seq}
1	$1/10$	77	7.8	$6.3 \cdot 10^{-20}$	192.7	3.1	$9.8 \cdot 10^{-15}$
1	$1/4$	95	6.3	$7.3 \cdot 10^{-7}$	136.6	4.4	$1.8 \cdot 10^{-3}$
1	$1/3$	76	7.9	$1.9 \cdot 10^{-3}$	103.5	5.8	$2.6 \cdot 10^{-1}$
2	$1/10$	76	7.9	$3.9 \cdot 10^{-13}$	96.9	6.2	$3.4 \cdot 10^{-7}$
2	$1/4$	51	11.8	$2.2 \cdot 10^{-4}$	68.8	8.7	$8.8 \cdot 10^{-2}$
2	$1/3$	43	14.0	$1.8 \cdot 10^{-2}$	–	–	–
4	$1/10$	39	15.4	$1.2 \cdot 10^{-7}$	49.0	12.2	$1.8 \cdot 10^{-3}$
4	$1/4$	28	21.4	$5.3 \cdot 10^{-3}$	34.8	17.2	$5.2 \cdot 10^{-1}$
4	$1/3$	24	25.0	$6.9 \cdot 10^{-2}$	–	–	–

Δ	propagation delay, seconds	ε	failure probability of $\mathcal{A}_k$
α	attacker's compute	$\bar{d}_{\text{seq}}$	block interval, seconds
k	number of votes	$\bar{k}_{\text{seq}}$	expected number of blocks
$\bar{d}$	activation delay, seconds	ε_{seq}	failure prob. after 10 minutes

3 PROOF-OF-WORK BLOCKCHAIN

Protocol $\mathcal{A}_k$ solves agreement using parallel proof-of-work. In this section, we propose a replication protocol $\mathcal{B}_k$ that repeatedly runs $\mathcal{A}_k$ to continuously agree on a growing sequence of values. In a nutshell, $\mathcal{B}_k$ is a blockchain protocol where the participants use $\mathcal{A}_k$ to agree on each appended block.

Time is divided in epochs of variable length, determined by the runtime of $\mathcal{A}_k$. Each epoch extends the blockchain by one block and confirms the value of the preceding epoch's block using $\mathcal{A}_k$. The safety guarantees of $\mathcal{A}_k$ imply that possible conflicting block proposals for the current epoch are resolved in the next epoch.

3.1 Prerequisites

In addition to the network assumptions of $\mathcal{A}_k$ (Sect. 2.1.2), we assume interfaces to an application layer and the availability of cryptographic primitives.

3.1.1 Application. $\mathcal{B}_k$ enables state replication and may serve as a basis for different applications [2, 48, 62]. For example, a simple cryptocurrency could append a list of transactions to each block. Jointly, the confirmed blocks would form a distributed ledger. More advanced applications could add scalability layers that only replicate selected decisions using $\mathcal{B}_k$ while handling other state updates separately [24, 46, 58].

We require that the application offers an interface with two procedures. getUpdate returns a valid state update that $\mathcal{B}_k$ can use for block proposals. applyUpdate passes replicated state updates to the application. The application may have other means to access the broadcast network directly. For example, cryptocurrencies share transactions provisionally before they are written in blocks.

3.1.2 Cryptography. $\mathcal{B}_k$ uses cryptographic hash functions for the hash-linking of blocks and the proof-of-work puzzle. The hash function used for the linking must be cryptographically secure. The hash function used for the proof-of-work puzzle requires the same stronger assumptions as in Nakamoto consensus [2]. In principle, one could separate these concerns and use two different hash functions. For simplicity, we use a single hash function $\mathcal{H}$ satisfying both requirements. The reader can safely assume $\mathcal{H}$ = SHA3.

In addition, $\mathcal{B}_k$ uses a secure digital signature scheme [43, Def. 12.1, p. 442] given by the procedures GENERATEKEYPAIR, CHECKSIGNATURE, and SIGN.

3.2 Protocol $\mathcal{B}_k$

We start with presenting core aspects of $\mathcal{B}_k$ in Sections 3.2.1 to 3.2.8 and integrate them into a complete protocol in Section 3.2.9.

3.2.1 Votes. A vote is a triple (r, p, s), with $\mathcal{H}(r, p, s) \leq t_{\mathrm{v}}$. We say r is the value voted for, p is the public key of the voting node, and s is the proof-of-work puzzle solution. The threshold t_{v} represents $\mathcal{B}_k$'s proof-of-work difficulty parameter and is set externally.

3.2.2 Quorums. A k-quorum is a set of k valid votes for the same value. A list $Q = \{(p_i, s_i)\}$ represents a valid k-quorum for value r, if the following conditions hold:

(1) $|Q| = k$
(2) $\forall\, 1 \leq i \leq k \colon \mathcal{H}(r, p_i, s_i) \leq t_{\mathrm{v}}$
(3) $\forall\, 1 \leq i < k \colon \mathcal{H}(r, p_i, s_i) < \mathcal{H}(r, p_{i+1}, s_{i+1})$

The first condition defines the quorum size k. The second condition ensures that all votes are valid. The third condition eliminates duplicates and imposes a canonical order which we use for leader selection. We write $Q[1]$ to address the first vote in the quorum.

Remark 2. The above definitions allow for single nodes providing multiple votes to a single quorum using the same public key. This is intentional. Sybil attacks are mitigated by the scarcity of votes, not by the scarcity of public keys.

3.2.3 Leader Selection. We say that node A is leader for the epoch that produces Q if A contributed the smallest vote $Q[1]$. Only leaders are allowed to propose new blocks. Nodes verify leadership based on the public key p_1, which is part of $Q[1]$.

Remark 3. Leader selection originates from the distributed system literature (e. g. [13, 31, 55, 68]), where it is used to improve performance in the optimistic case that the leader follows the rules. A similar, leader-based performance improvement has been proposed for Nakamoto consensus [24]. Our leader selection mechanism is an optimization as well. It reduces the number of proposals per epoch and thereby improves communication efficiency. Recall that the agreement protocol $\mathcal{A}_k$ resolves conflicting preferences even if all nodes started with their own preferred value (Sect. 2). Thus, $\mathcal{B}_k$ is secure even if leaders equivocate or multiple leaders are selected.

3.2.4 Blocks. A block is a proposed extension to the blockchain. Besides the application payload, a block holds additional values that ensure orderly execution of the agreement $\mathcal{A}_k$ and the leader selection mechanism according to Sections 3.2.1 to 3.2.3. A valid block b contains the following information.

(1) parent(b) is either the hash of a previous valid block or equal to the protocol parameter H_0, which characterizes the instance of the protocol;[3]
(2) quorum(b) is a valid k-quorum for parent(b);
(3) payload(b) is the proposed state update returned from GETUPDATE;
(4) signature(b) is a valid signature of the triple (parent(b), quorum(b), payload(b)) signed with the private key corresponding to the public key in quorum$(b)[1]$.

[3]In blockchain jargon, H_0 is the hash of the genesis block.

The first condition imposes a sequential order on the list of blocks. The second condition ensures that all nodes agree on the previous block before proposing a new block ($\mathcal{A}_k$, Sect. 2). The forth condition restricts the ability to propose blocks to selected leaders (Sect. 3.2.3).

3.2.5 Local Block Tree. Each node locally maintains a hash-linked *tree* of blocks $\mathcal{T}$. We write $\mathcal{T}[h]$ to access the block b with $\mathcal{H}(b) = h$. For each block b, nodes maintain

(1) height(b), the number of predecessors of b in $\mathcal{T}$,
(2) state(b), the application state associated with b, and
(3) votes(b), the set of votes that confirm b.

3.2.6 Block Preference. Nodes prefer block a over block b if,

(1) height$(a) >$ height(b), or
(2) height$(a) =$ height$(b) \wedge |$votes$(a)| > |$votes$(b)|$.

In other words, they follow the longest chain (1) between epochs and the voting protocol $\mathcal{A}_k$ within each epoch (2). This rule is ambiguous if there are multiple blocks of equal height and with the same number of confirming votes. In this case, nodes prefer the block first received. The embedded voting protocol $\mathcal{A}_k$ makes the nodes agree on the same parent block until the end of the epoch.

Remark 4. Under normal operation with a constant set of nodes (i. e., no late joining), the longest chain rule will only be invoked to disambiguate the last epoch. The ε-safety guarantee of $\mathcal{A}_k$ ensures that longer forks are unlikely.

3.2.7 Proof-of-Work Voting. Nodes continuously try to find and share valid votes for their preferred block. Recall that a valid vote $v = (\mathcal{H}(b), p, s)$ satisfies $\mathcal{H}(v) \leq t_{\mathrm{v}}$, where b is the preferred block and p is the node's public key. Due to the properties of the hash function (Sect. 3.1.2), the best solution strategy is iterative trial and error for different values of s. Solving this hash puzzle on physical hardware implements the stochastic clock P_λ presented in Section 2.1.3 for the arrival of votes in a distributed system. Parameter t_{v} must be adjusted to the desired puzzle solving rate λ for a given technology and proof-of-work capacity.

3.2.8 Proposing. Nodes assume leadership whenever possible. I. e., they constantly check whether they can form a quorum Q where the smallest vote $Q[1]$ is their own. If so, they request a state update from the application, integrate it as payload into a new valid block (Sect. 3.2.4), and broadcast it.

3.2.9 Integration. Algorithm 2 integrates Sections 3.2.1 to 3.2.8 into the complete protocol $\mathcal{B}_k$. During initialization, nodes generate a key pair for the digital signature scheme (Sect. 3.1.2) and initialize the empty block tree (ln. 1–3). Two event-handlers process incoming messages (ln. 4 and 6). Valid votes are stored (ln. 10–12) and valid blocks are appended to the local block tree (ln. 13–19). In the background, nodes continuously try to solve proof-of-work puzzles in order to cast votes for their preferred value (ln. 36–42). The path between H_0 and the preferred value in the local block tree

Algorithm 2 Blockchain protocol $\mathcal{B}_k$

```
 1: upon event ⟨ init ⟩ do
 2:     pk, sk ← generateKeyPair()
 3:     height($\mathcal{T}[H_0]$) ← 0                          ▷ $H_0$: hard-coded value
 4: upon event ⟨ deliver | vote $v$ ⟩ do                      ▷ from line 42
 5:     count($v$)
 6: upon event ⟨ deliver | block $b$ ⟩ do                     ▷ from line 35
 7:     for all $(p, s)$ in quorum($b$) do                     ▷ count all votes
 8:         count(parent($b$), $p$, $s$)
 9:     store($b$)
10: procedure count(vote $(r, p, s)$)
11:     if $\mathcal{H}(r, p, s) \leq t_v$ then                ▷ vote validity, Sect. 3.2.1
12:         votes($\mathcal{T}[r]$) ← votes($\mathcal{T}[r]$) ∪ {(p, s)}
13: procedure store(block $b$)
14:     if $b$ is valid by Sect. 3.2.4 and $\mathcal{H}(b) \notin \mathcal{T}$ then
15:         votes($b$) ← ∅
16:         $a$ ← $\mathcal{T}$[parent($b$)]
17:         height($b$) ← height($a$) + 1
18:         state($b$) ← applyUpdate(state($a$), payload($b$))
19:         $\mathcal{T}[\mathcal{H}(b)]$ ← $b$
20: procedure preferred
21:     Find most preferred block $b$ according to Sect. 3.2.6.
22:     return $\mathcal{H}(b)$
23: procedure leader($r$)
24:     Try to form $k$-quorum $Q$ for $r$ as leader by Sect. 3.2.3.
25:     if possible then return $Q$
26:     else return ⊥
27: upon change to $\mathcal{T}$ do
28:     $r$ ← preferred ; $Q$ ← leader($r$)
29:     if Q then                                              ▷ build block according to Sect. 3.2.8
30:         parent($b$) ← $r$
31:         quorum($b$) ← $Q$
32:         payload($b$) ← getUpdate(state($\mathcal{T}[r]$))
33:         signature($b$) ← sign($b$, sk)
34:         store(b)
35:         broadcast(block b)
36: procedure proofOfWork                                      ▷ background task
37:     loop
38:         $r$ ← preferred ; $s$ ← $s$ + 1
39:         if $\mathcal{H}(r, \text{pk}, s) \leq t_v$ then
40:             votes($\mathcal{T}[r]$) ← votes($\mathcal{T}[r]$) ∪ {(pk, s)}
41:             if not leader($r$) then
42:                 broadcast(vote $(r, \text{pk}, s)$)
```

represents the current version of the blockchain. Whenever the local block tree changes (ln. 27 triggered from ln. 12 and 40), nodes try to assume leadership and propose a new block (ln. 29–35).

We visualize a typical execution of $\mathcal{B}_k$ in Figure 15 in the Appendix and compare it to sequential proof-of-work.

Remark 5. $\mathcal{B}_1$, i. e., $\mathcal{B}_k$ with $k = 1$, closely resembles Bitcoin as proposed by Nakamoto [52]. However, we highlight one key difference: in Bitcoin, blocks carry a first proof-of-work confirmation of the payload proposed within the block itself. In $\mathcal{B}_k$, the proof-of-work solutions confirm the *previous* block. This enables parallel puzzle solving for $k > 1$.

3.3 Finality

Finality means that the application accepts a commit when it is deemed safe. Deterministic finality is not achievable with stochastic protocols, but our concrete safety bounds help to make an informed decision on when to accept. By implementing $\mathcal{A}_k$ in $\mathcal{B}_k$, we ensure that the commit of the state update in a block with height i is ε-safe as soon as a block with height $i + 1$ is observed.

For example, the configuration $\alpha = 1/4$, $k = 51, \bar{t} = 600$ from Table 3 is 0.0002-safe. This implies that the worst case attacker (within the model) succeeds in causing inconsistent commits in one of 5,000 attempts. In practice, such an attacker would find it easier to temporarily increase the share in compute power above $\alpha = 1/2$, where every system solely based on proof-of-work fails. With proof-of-work capacity being available for rent [9], this turns into an economic argument which is orthogonal to the design assumptions of $\mathcal{B}_k$. This leads us to a brief discussion of incentives.

3.4 Incentives

It is possible to motivate participation in $\mathcal{B}_k$ by rewarding puzzle solutions. This requires some kind of virtual asset that can be transferred to a vote's public key. Claiming the reward would depend on the corresponding private key.

$\mathcal{B}_k$ could allot a constant reward per puzzle. As votes occur k times more frequently than blocks, $\mathcal{B}_k$'s mining income would be less volatile than in Nakamoto consensus, making participation more attractive to risk-averse agents with small compute power.

It is tempting to demand that the reward scheme is incentive compatible, i. e., that correct execution is a unique maximum of the nodes' utility function. However, it is not trivial to achieve incentive compatibility because utility of rewards *outside* the system may affect the willingness to participate *in* the system [27]. We do not know any blockchain protocol analysis that solves this problem convincingly. Thus, $\mathcal{B}_k$ is designed to support rewards as a means to encourage participation, but its security intentionally does not depend on incentives. This is a feature, not a bug.

4 EVALUATION

We evaluate $\mathcal{B}_k$ by discrete event network simulation. We implement $\mathcal{B}_k$ and the network simulation in OCaml. All results are reproducible with the code provided online [44].

We choose the configuration $k = 51$ and $\lambda = 51/600$, which is optimized for $\alpha = 1/4$ and $\Delta = 2''$. Its failure probability is at most $2.2 \cdot 10^{-4}$ (see Sect. 2.6, Tab. 3). The expected block interval is 10 minutes, which enables comparison to Nakamoto consensus, more specifically Bitcoin. For the purpose of this simulation, Bitcoin is equivalent to $\mathcal{B}_1$ with $\lambda = 1/600$ (see Sect. 3 Remark 5).

While the worst-case propagation delay Δ is specified at design time, realistic network latencies vary. In the simulation, we set an expected network delay δ and use it to draw individual delays for each message delivery from

(1) a **uniform** distribution on the interval $[0, 2 \cdot \delta]$, and
(2) an **exponential** distribution with rate δ^{-1}.

We also consider that votes may propagate faster than blocks because they are much smaller and their validation does not depend on the application state. To this end we define

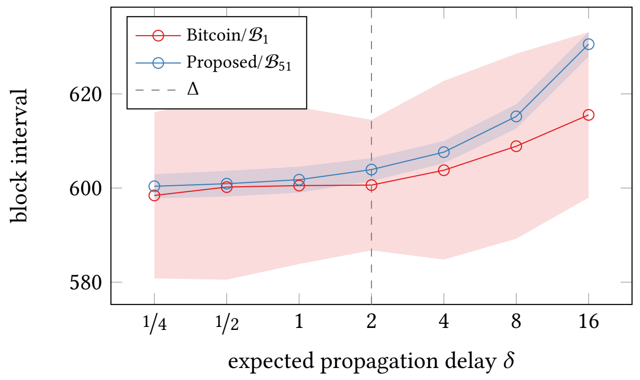


Figure 9: The effect of latency on the average block interval in the simple/exponential scenario. Time in seconds.

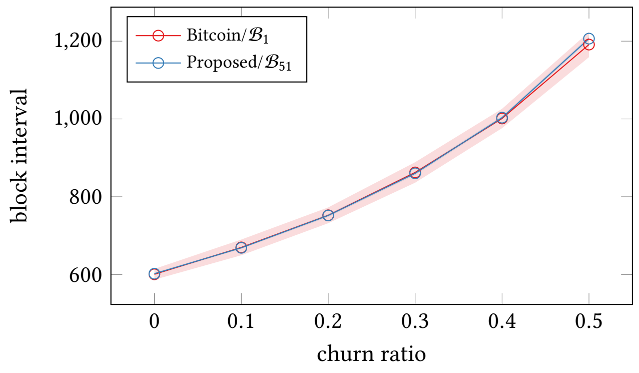


Figure 10: The effect of churn on the average block interval in the realistic/exponential scenario. The churn ratio describes how many of the nodes are passive at any given time.

(1) a **simple** treatment where $\delta = \Delta = 2''$ for all messages, and
(2) a **realistic** treatment where blocks propagate with $\delta_{\mathbf{b}} = 2''$ and votes eight times faster, $\delta_{\mathbf{v}} = 1/4''$.

The cross product of the two distributions and two treatments of small messages gives us four scenarios to be simulated. Note that for all scenarios some delays will be greater than the assumed worst-case propagation delay Δ. For some measurements, we will raise δ beyond Δ to put the protocol under even more pressure.

Unless stated otherwise, measurements are based on a simulated network with 1024 nodes.[4] For each experiment, we average over 64 independent executions up to block height 4096. All figures showing variation do this by plotting ±1.96 standard deviations around the mean of the 64 independent executions. For all executions of $\mathcal{B}_{51}$, we checked for inconsistent commits, which did not occur. As another plausibility check, we verified that the simulated block intervals of $\mathcal{B}_1$ and $\mathcal{B}_{51}$ match the theoretical distributions described in Section 2.5.

4.1 Robustness

We evaluate the robustness of $\mathcal{B}_{51}$ against excessive latency, churn, and leader failure by measuring block intervals. Recall that the simulated protocols are configured for a 10 minute interval in optimal conditions. The puzzle solving rate is constant. Stress implies wasted proof-of-work and hence higher observed block intervals.[5]

4.1.1 Latency. We use the **simple/exponential** scenario and vary the expected propagation delay δ from 1/4 to 16 seconds. Recall that the choice of $k = 51$ is optimized for $\Delta = 2$ seconds. Larger expected propagation delays put the protocol under stress. Figure 9 shows the effect of latency on the block interval. We observe that even excessive random propagation delays ($\delta = 16$ seconds) slow down $\mathcal{B}_{51}$-consensus by only about 5 %. The **simple/uniform** scenario exhibits similar behavior. We refrain from exploring the **realistic** treatment as it is not obvious how real network latency would affect the ratio of $\delta_{\mathbf{b}}$ and $\delta_{\mathbf{v}}$.

[4]Measurements suggest that there are roughly 10 000 Bitcoin nodes, while 80 % of the compute power is held by the top 10 agents [51].
[5]This metric relates to the orphan rate in the literature on sequential proof-of-work.

4.1.2 Churn. We simulate churn by muting a fraction of nodes (churn ratio) for one hour each. Muted nodes solve proof-of-work puzzles but do not send or receive messages. Accordingly, the votes and blocks created by muted nodes represent lost work. We expect that the block interval is inversely proportional to the churn ratio: if 50 % of the nodes are muted, the average block interval is twice as long. Figure 10 supports this claim for both protocols.

4.1.3 Leader Failure. $\mathcal{B}_k$ separates proof-of-work (votes) from blocks. Leaders selected during the epoch may fail to propose at the end of the epoch. We model such failures by dropping block proposals randomly with constant probability (leader failure rate).

A special property of $\mathcal{B}_k$ is that it can reuse votes for different proposals. Honest nodes reveal at most one new vote with their proposal. Accordingly, a lost proposal wastes at most the work of one vote. Therefore, $\mathcal{B}_{51}$ can recover fast. The results in Figure 11 support this claim. For perspective, the right end of the graph simulates a situation where an attacker can monitor all nodes' network traffic and disconnect leaders at discretion with 50 % success probability. Still, the block interval grows only by about 2.5 %. This effect is similar to the robustness against excessive latencies discussed in Section 4.1.1.

For $\mathcal{B}_1$, voting, leader selection, and proposing happens in a single message. Leader failure is similar to churn and hence has a much stronger effect (see Fig. 10).

4.2 Security

Zhang and Preneel [70] propose to evaluate blockchain protocols with respect to the four security aspects

(1) *subversion gain*, to what extent an attacker can rewrite confirmed blocks,
(2) *chain quality*, how much of the confirmed blocks are proposed by the attacker,
(3) *censorship susceptibility*, how long the attacker can block certain transactions, and
(4) *incentive compatibility*, how much reward the attacker can collect by deviating from the protocol.

Our approach is to derive subversion gain from the ε-safety of $\mathcal{A}_k$ and then evaluate chain quality and censorship susceptibility jointly.

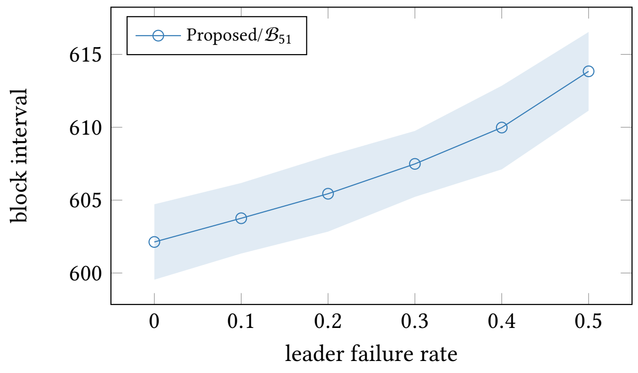


Figure 11: The effect of leader failure on the block interval in the realistic/exponential scenario. The leader failure rate is the probability that a selected leader fails to propose a block.

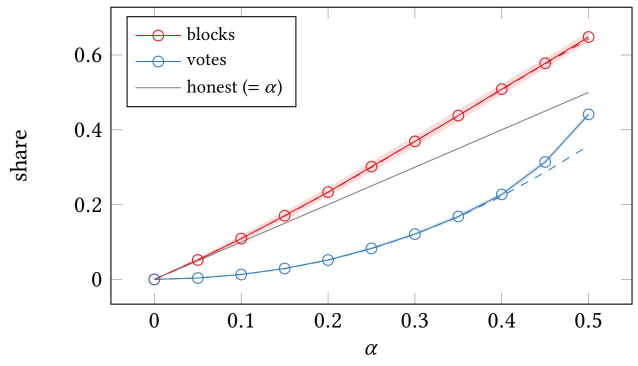


Figure 12: The attacker's share of confirmed blocks and votes as functions of α in the realistic/exponential scenario. The attacker uses vote withholding against $\mathcal{B}_{51}$. The gray line shows the expected shares without attack. The dashed lines show results from the Markov chain model. The solid lines show the validation in the network simulator.

This is sufficient because both aspects depend on the attacker being selected as a leader. Turning to incentive compatibility, we argue in Section 3.4 why it seems impossible to prove this for realistic utility functions. Zhang and Preneel use a restricted notion in which the attacker utility is the share of rewards assigned by the protocol. We can evaluate their definition of incentive compatibility, along with chain quality and censoring.

4.2.1 Subversion Gain. We provide a consistency analysis for the agreement $\mathcal{A}_k$ in Section 2. The proposed $\mathcal{B}_{51}$ executes $\mathcal{A}_{51}$ for each appended block. The probability that an $\alpha = 1/4$ attacker in a $\Delta = 2''$ synchronous network succeeds in causing inconsistent state updates (e. g., double spend) is $2.2 \cdot 10^{-4}$ (see Tab. 3). The proposed protocol meets this guarantee after one block confirmation, i. e., after about 10 minutes.We assume that applications wait for this confirmation and conclude that subversion gain is not a practical concern for $\mathcal{B}_{51}$.

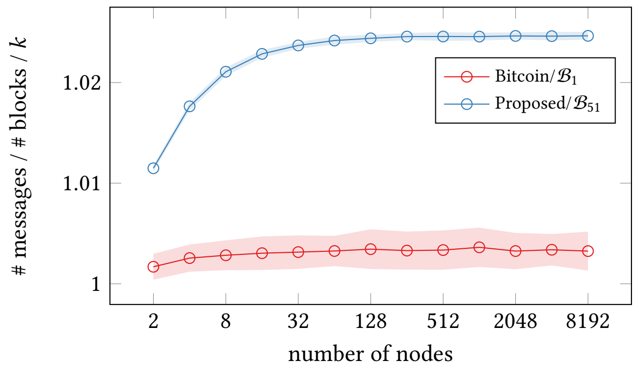


Figure 13: Number of broadcast messages per block divided by k for networks of different size in the realistic/exponential scenario.

4.2.2 Chain Quality, Censoring, and Incentives. Chain quality measures the share of confirmed blocks proposed by the attacker. Censoring is possible only if the attacker controls the proposed block payload. Thus, chain quality and censoring reduce to the question of how often an attacker can take leadership.

A common weakness of sequential proof-of-work protocols relates to information withholding. Block withholding, proposed by Eyal and Sirer [25], enables selfish mining against Bitcoin. $\mathcal{B}_k$ is not vulnerable to block withholding because selected leaders who do not propose a block are quickly replaced (see Sect. 4.1.3). The remaining information to be considered in withholding attacks are votes (see Sect. 2.4 for $\mathcal{A}_k$; related [10]). In $\mathcal{B}_k$, the attacker can prolong an epoch by withholding votes until the honest nodes can form a k-quorum themselves. The attacker can use the additional time to mine the smallest vote and be selected as leader.

We first analyze the effectiveness of vote withholding in a single epoch using a Markov chain model (see Appendix A). Then we use the network simulator to confirm the results for executions of the protocol over multiple epochs.

Figure 12 shows the success rate of the attacker in red and his number of confirmed votes in blue. Solid lines originate from the network simulator and dashed lines from the Markov chain model. Both evaluation methods concur in the main result: a withholding attacker can become the leader in about $1.3 \cdot \alpha$ cases (65 % for $\alpha = 50\%$). His advantage in taking leadership comes at the price of fewer confirmed votes. If rewards are proportional to votes, this tells us that vote withholding is disincentivized. For comparison with Nakamoto consensus, block withholding strategies give an $\alpha = 1/3$ attacker an advantage of $1.5 \cdot \alpha$. This factor raises to $2 \cdot \alpha$ for $\alpha = 1/2$ [61]. Moreover, successful selfish miners receive more rewards than without attack. The results indicate that $\mathcal{B}_{51}$ offers higher chain quality, is less susceptible to censorship, and offers fewer incentives to attack than Nakamoto consensus.

4.3 Overhead

Nakamoto consensus requires at least one message broadcast per appended block, namely the block itself, independent of the number of participating nodes. $\mathcal{B}_k$ adds k message broadcasts per block—one for each vote. We evaluate the actual number of sent messages in the network simulator. Figure 13 shows the number of broadcast messages as a function of the number of blocks and k. Observe that $\mathcal{B}_{51}$ plateaus at about $1.025 \cdot k$, i. e., 52 broadcasts per block. This number remains stable as the network scales up.

While the constant factor k may matter for practical networks, it is worth pointing out that vote messages are much smaller than blocks. Under the conservative assumptions of 256 bits each for the block reference and the public key, and 64 bits for the puzzle solution, a vote is as small as 72 B.[6]

The votes also cause a constant storage overhead. $\mathcal{B}_k$ persists the complete quorum of k votes for future verification. Note that the reference r is redundant in all votes and needs to be stored only once. Hence, under the assumptions leading to 72 B message size, the storage overhead of $\mathcal{B}_{51}$ is about 2 kB per block. This is less than 0.2 % of Bitcoin's average block size in April 2022.

4.4 Detecting Network Splits

The assumption of a Δ-synchronous network is unavoidable for proof-of-work protocols: delaying the propagation of a defender's puzzle solution is equivalent to reducing his compute power. With unbounded delays, even a weak attacker can solve sufficiently many puzzles before the defender's solutions propagate [56].

While network splits clearly violate this assumption, we still want to highlight that $\mathcal{B}_k$ allows for faster detection of such events than Nakamoto consensus. In $\mathcal{B}_k$, each vote carries one puzzle solution. The activation delay is exponentially distributed with rate λ (see Sect. 2.1.3). In an intact network, the time between received votes should follow the same distribution. This allows nodes to test the hypothesis of being eclipsed. For $\mathcal{B}_{51}$, a node can distinguish a network split from normal conditions with high confidence after 82 seconds of not receiving a vote (error probability $p = 0.1\,\%$). For comparison, the same hypothesis test would require more than an hour of observation in Bitcoin.

5 DISCUSSION

We discuss our contributions from several perspectives. Section 5.1 compares the security analysis of $\mathcal{A}_k$ to the relevant literature. Section 5.2 positions the family of protocols $\mathcal{B}_k$ in the design space of blockchain protocols. Limitations and directions for future work are discussed in Section 5.3.

5.1 Related Security Analyses

Our security analysis of $\mathcal{A}_k$ is inspired by the literature on Bitcoin security. Table 4 summarizes selected landmark contributions.

The first formal security argument of the so-called "Bitcoin backbone protocol" [30] discretized time in slots. Each slot represents one puzzle trial attempt. Communication is synchronous, i. e., messages are delivered at the end of each slot. Security proofs for consistency and chain quality were given asymptotically in the number of slots. The work formally established the eventual consistency of Nakamoto consensus in synchronous networks.

[6]Bitcoin shortens public keys to 160 bits and uses solutions of 32 bits. Its blocks are in the order of 1 MB.

Table 4: Comparison of related security analyses.

	Garay et al., 2015 [30]	Pass et al., 2017 [56]	Kiffer et al., 2018 [45]	Gaži et al., 2020 [32]	Dembo et al., 2020 [20]	Li et al., 2021 [50]	this paper
time							
- discrete slots	✓	✓	✓	✓			
- continuous					✓	✓	✓
synchrony	slot	Δ	Δ	Δ	Δ	Δ	Δ
security							
- eventual	✓	✓	✓	✓	✓		
- ε-bounded						✓	✓
Markov chains			✓				✓

Follow-up work generalized the main results of [30] for a Δ-synchronous communication model. It allows messages to be delivered in future slots [56]. Further refinements using Markov chain models resulted in tighter, but still asymptotic bounds [45]. Recently, two research groups independently derived optimal bounds [20, 32] for the attacker threshold α. Dembo et al. [20] use continuous time and model proof-of-work as a Poisson process.

All analyses cited above use asymptotic security notions. A recent contribution to AFT '21 breaks with this tradition and provides concrete failure bounds for Nakamoto consensus after waiting for a given confirmation time [50] (comp. Sect. 2.6). Concurrent work improves these bounds [33] and simplifies the analysis [37].

Our analysis of $\mathcal{A}_k$ establishes ε-safety in Δ-synchronous networks. We use Poisson processes to model proof-of-work in continuous time, and Markov chains as an analytic tool.

5.2 Related Protocols

We do not attempt to provide a complete map of the design space for replication protocols since other researchers have specialized on this task [5, 11, 12, 29]. Instead, we compare $\mathcal{B}_k$ to some of its closest relatives along selected dimensions.

Research on agreement and state replication began in the late 1970s, initially only considering benign but unreliable behavior [48, 59]. In the early 1980s, Lamport extended the discussion to adversarial behavior. He coined the term Byzantine Fault Tolerance (BFT) [49], impliying that at most f out of $n = 3f + 1$ identified nodes may deviate from the protocol. Early BFT protocols [e. g., 13, 21] rely on communication between all n nodes for each value agreed upon. This results in quadratic communication complexity and renders the protocols impractical for more than a dozen nodes.

In that light, Bitcoin [52] can be seen as a technical breakthrough [2, 53]. Nodes can join the network without obtaining permission from a central gatekeeper. Active participation, i. e., proposing new state updates, is limited by the nodes' ability to solve proof-of-work puzzles. This yields sub-quadratic communication complexity and allows Bitcoin to scale to thousands of nodes.

However, the sequential proof-of-work scheme proposed with Bitcoin faces a fundamental trade-off between security and throughput. It supports only one state update per block, while the time between blocks must be in the order of seconds to minutes [50]. Bitcoin-NG [24] tries to resolve this conflict by separating leader selection and transaction processing. The miner of a block becomes responsible for appending multiple consecutive state updates until the next leader emerges with the next mined block. To discipline the leader, Bitcoin-NG relies on incentives.

A number of protocols extend on the idea and try to shift trust from a single to multiple leaders. E. g., Byzcoin [46] selects a committee from the last successful miners, who then run a conventional BFT protocol to agree on the latest state. Similar layered protocols evolved concurrently [18] and afterwards [3, 57, 58]. However, the synchronization between the different consensus layers increases protocol complexity [3, 57, 58] and is a source of concern [18, 46]. Moreover, all layered protocols assume that the attacker cannot corrupt committee members selectively.

$\mathcal{B}_k$ does not layer different consensus mechanisms. It implements state replication directly from proof-of-work and broadcast primitives. As demonstrated in Section 4.1.3, $\mathcal{B}_k$ tolerates selective corruption of committee members.

Other protocols with non-sequential proof-of-work have been proposed. E. g., Phantom [65] replaces the linear blockchain data structure with a directed acyclic graph (DAG) for better scalability and faster first confirmation in latent networks. However, it also increases protocol complexity; in particular, deriving a total order of state updates from the DAG is NP-hard. A related idea is to operate multiple sequential blockchains in parallel [4, 26, 45, 69]. A close relative of $\mathcal{B}_k$ is Bobtail [10]. It uses multiple proof-of-work puzzles per block, like $\mathcal{B}_k$, but binds a preliminary state update to each vote. Votes reference earlier votes, hence Bobtail mixes elements of parallel and sequential proof-of-work. All cited non-sequential protocols lack a principled analysis on when to accept state updates. $\mathcal{B}_k$ is the first non-sequential proof-of-work protocol that comes with concrete bounds for safety.

A separate line of research explores alternatives to the wasteful proof-of-work primitive. Proof-of-stake protocols [e. g., 14, 16, 17] select committee members for participation in an agreement procedure based on the distribution of stake in the system's cryptocurrency. This creates a cyclic dependency which, we think, has not been solved convincingly. Other proof-of-x protocols try to provide useful services like file storage [7] or radio network coverage [38]. However, these protocols seem to rely on heuristics for truthful resource accounting.

In the meantime, research on permissioned protocols caught up. Current BFT-style protocols achieve sub-quadratic communication complexity, enabling deployments with hundreds of nodes [1, 68]. These protocols depend on identities and can scale applications as long as the BFT assumptions hold.

For digital currencies, it might be worth abandoning state machine replication completely and implement digital asset transfer directly from reliable broadcast [36]. Promising proposals exist for the BFT assumptions [6], proof-of-work [64], and proof-of-stake [63]. However, this approach restricts the versatility of the application layer. It cannot support arbitrary smart contract logic.

5.3 Limitations and Future Work

We have presented a permissionless replication protocol that achieves ε-safety in Δ-synchronous networks with computationally bounded attackers. Although our model is widely accepted in the literature [30, 56], it is worth discussing its assumptions.

We assume a fixed puzzle solving rate λ, but in practice agents can add and remove compute power at their discretion. Practical systems try to stabilize λ with a control loop known as difficulty adjustment algorithm (DAA) [8, 28, 39, 41, 47]. For $\mathcal{B}_k$, the accuracy of a DAA could increase in k as every additional vote provides a data point for the estimation of λ. Turning to the synchronous network assumption, as shown in Section 4.4, the response time to detect network splits decreases for larger k. This relates to the CAP theorem [35], which states that every distributed system has to sacrifice one out of consistency, availability, and partition tolerance. $\mathcal{B}_k$, as specified in Algorithm 2, favors availability over consistency. The trade-off could be changed in favor of consistency by implementing the split detection. Such a variant of the protocol could notify the application layer to withhold commits and trigger out-of-band resolutions.

The perhaps most problematic assumption is that the attacker's share in compute power α is small (see Table 2). Violations, especially $\alpha > 1/2$, are catastrophic, but have been observed in practice [23]. Note that the theory in Section 2.4 does apply for values of $\alpha > 1/3$, but the resulting failure probabilities ε are unattractive. This contrasts with the BFT literature, which requires a hard upper threshold of $n = 3f + 1$ to satisfy an absolute notion of safety.

This leads us to future work. Our evaluation of $\mathcal{B}_k$ is limited to one instance, $\mathcal{B}_{51}$, and uses independent propagation delays on a fully connected graph. We chose the instance for comparability with Bitcoin. Tests of other protocol configurations with more realistic topologies could complete the picture. However, as the literature reports discrepancies between the topology implied at design time and the one observed in practice [19, 51], it is not obvious what topology would be appropriate. A different direction is to explore improvements in the optimistic case by including application-level payloads into $\mathcal{B}_k$'s vote messages. E. g., one could add transactions that do not require consensus [6, 36, 63] or implement a staging of state updates [71]. Finally, as explained in Section 3.4, we refrain from designing an incentive mechanism for $\mathcal{B}_k$. Possible approaches are to search reward-optimizing strategies using Markov Decision Processes (MDPs) [61, 70, 72] or reinforcement learning [40].

6 CONCLUSION

The proposed family of protocols $\mathcal{A}_k$ proves that unidentified nodes can reach agreement with guaranteed liveness and ε-safety in a Δ-synchronous network using proof-of-work. The family of protocols $\mathcal{B}_k$ shows that parallel proof-of-work enables blockchain protocols with concrete security bounds. With k chosen as described, $\mathcal{B}_k$ enables permissionless state replication that serves certain applications better than existing systems.

It is worth noting that proof-of-work is a wasteful way of establishing agreement. Many alternatives exist if nodes are identifiable. The value of this research is to get better guarantees from protocols when there is no alternative to proof-of-work.

REFERENCES

[1] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Communication Complexity of Byzantine Agreement, Revisited. In *Symposium on Principles of Distributed Computing*. ACM, 317–326.

[2] Ittai Abraham and Dahlia Malkhi. 2017. The Blockchain Consensus Layer and BFT. *Bulletin of the EATCS* 123 (2017).

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2017. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. arXiv:1612.02916 [cs]

[4] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the Blockchain to Approach Physical Limits. In *Conference on Computer and Communications Security*. ACM, 585–602.

[5] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2019. SoK: Consensus in the Age of Blockchains. In *Conference on Advances in Financial Technologies*. ACM, 183–198.

[6] Mathieu Baudet, George Danezis, and Alberto Sonnino. 2020. FastPay: High-Performance Byzantine Fault Tolerant Settlement. In *Conference on Advances in Financial Technologies*. ACM, 163–177.

[7] Juan Benet, David Dalrymple, and Nicola Greco. 2017. *Proof of Replication*. Technical Report. Protocol Labs.

[8] George Bissias. 2020. Radium: Improving Dynamic PoW Targeting. In *Workshop on Cryptocurrencies and Blockchain Technology (Lecture Notes in Computer Science, Vol. 12484)*, Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomarti (Eds.). Springer, 374–389.

[9] George Bissias, Rainer Böhme, David Thibodeau, and Brian Levine. 2022. Pricing Security in Proof-of-Work Systems. In *Workshop on the Economics of Information Security*.

[10] George Bissias and Brian N. Levine. 2020. Bobtail: Improved Blockchain Security with Low-Variance Mining. In *Network and Distributed Systems Security Symposium*. Internet Society.

[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Symposium on Security and Privacy*. IEEE, 104–121.

[12] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild (Keynote Talk). In *Symposium on Distributed Computing (Leibniz International Proceedings in Informatics, Vol. 91)*, Andréa W. Richa (Ed.). 1:1–1:16.

[13] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems* 20, 4 (2002), 398–461.

[14] Jing Chen and Silvio Micali. 2019. Algorand: A Secure and Efficient Distributed Ledger. *Theoretical Computer Science* 777 (2019), 155–183.

[15] Mauro Conti, E. Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. Fourthquarter 2018. A Survey on Security and Privacy Issues of Bitcoin. *IEEE Communications Surveys Tutorials* 20, 4 (Fourthquarter 2018), 3416–3452.

[16] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, 23–41.

[17] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-Synchronous Proof-of-Stake Blockchain. In *Advances in Cryptology – EUROCRYPT 2018 (Lecture Notes in Computer Science, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 66–98.

[18] Christian Decker, Jochen Seidel, and Roger Wattenhofer. 2016. Bitcoin Meets Strong Consistency. In *International Conference on Distributed Computing and Networking*. ACM, 1–10.

[19] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. 2019. TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, 550–566.

[20] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2020. Everything Is a Race and Nakamoto Always Wins. In *Conference on Computer and Communications Security*. ACM, 859–878.

[21] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988), 288–323.

[22] Cynthia Dwork and Moni Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology – CRYPTO '92 (Lecture Notes in Computer Science, Vol. 740)*, Ernest F. Brickell (Ed.). Springer, 139–147.

[23] Attah Elikem. 2019. Five Most Prolific 51% Attacks in Crypto: Verge, Ethereum Classic, Bitcoin Gold, Feathercoin, Vertcoin. https://cryptoslate.com/prolific-51-attacks-crypto-verge-ethereum-classic-bitcoin-gold-feathercoin-vertcoin/.

[24] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *Symposium on Networked Systems Design and Implementation*. USENIX, 45–59.

[25] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 8437)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, 436–454.

[26] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2020. Ledger Combiners for Fast Settlement. In *Theory of Cryptography (Lecture Notes in Computer Science, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 322–352.

[27] Bryan Ford and Rainer Böhme. 2019. Rationality Is Self-Defeating in Permissionless Systems. arXiv:1910.08820 [cs]

[28] Daniel Fullmer and A. Stephen Morse. 2018. Analysis of Difficulty Control in Bitcoin and Proof-of-Work Blockchains. In *Conference on Decision and Control*. IEEE, 5988–5992.

[29] Juan Garay and Aggelos Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *Topics in Cryptology – CT-RSA 2020 (Lecture Notes in Computer Science, Vol. 12006)*, Stanislaw Jarecki (Ed.). Springer, 284–318.

[30] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology – EUROCRYPT 2015 (Lecture Notes in Computer Science, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, 281–310.

[31] Hector Garcia-Molina. 1982. Elections in a Distributed Computing System. *IEEE Trans. Comput.* C-31, 1 (1982), 48–59.

[32] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2020. Tight Consistency Bounds for Bitcoin. In *Conference on Computer and Communications Security*. ACM, 819–838.

[33] Peter Gaži, Ling Ren, and Alexander Russell. 2021. *Practical Settlement Bounds for Proof-of-Work Blockchains.* Cryptology ePrint 805. IACR.

[34] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Conference on Computer and Communications Security*. ACM, 3–16.

[35] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News* 33, 2 (2002), 51–59.

[36] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Symposium on Principles of Distributed Computing*. ACM, 307–316.

[37] Dongning Guo and Ling Ren. 2022. Bitcoin's Latency–Security Analysis Made Simple. In *Conference on Advances in Financial Technologies*. ACM.

[38] Amir Haleem, Andrew Allen, Andrew Thompson, Marc Nijdam, and Rahul Garg. 2018. *Helium: A Decentralized Wireless Network.* Technical Report. Helium Systems. 20 pages.

[39] Thomas M. Harding. 2020. Real-Time Block Rate Targeting. *Ledger* 5 (2020).

[40] Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramèr, Giulia Fanti, and Ari Juels. 2021. SquirRL: Automating Attack Analysis on Blockchain Incentive Mechanisms with Deep Reinforcement Learning. In *Network and Distributed Systems Security Symposium*. Internet Society.

[41] Geir Hovland and Jan Kucera. 2017. Nonlinear Feedback Control and Stability Analysis of a Proof-of-Work Blockchain. *Modeling, Identification and Control* 38, 4 (2017), 157–168.

[42] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-Spending Fast Payments in Bitcoin. In *Conference on Computer and Communications Security*. ACM, 906–917.

[43] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography* (second ed.). CRC Press.

[44] Patrik Keller. 2021. Protocol implementation and network simulator. https://github.com/pkel/hotpow/tree/ppow.

[45] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. 2018. A Better Method to Analyze Blockchain Consistency. In *Conference on Computer and Communications Security*. ACM, 729–744.

[46] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Security Symposium*. USENIX, 279–296.

[47] Daniel Kraft. 2016. Difficulty Control for Blockchain-Based Consensus Systems. *Peer-to-Peer Networking and Applications* 9, 2 (2016), 397–413.

[48] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565.

[49] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[50] Jing Li, Dongning Guo, and Ling Ren. 2021. Close Latency–Security Trade-off for the Nakamoto Consensus. In *Conference on Advances in Financial Technologies*. ACM, 100–113.

[51] Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stütz, and Bernhard Haslhofer. 2020. All That Glitters Is Not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network. In *Network Operations and*

$$\begin{tikzcd}
 & a+1, d, \top \\
a, d, \top \arrow[ur, "\alpha", "\text{attacker finds vote}"'] \arrow[r, "(1-\alpha)/(a+d+1)", "\text{defender finds smallest vote}"'] \arrow[dr, "(1-\alpha)\cdot(a+d)/(a+d+1)", "\text{defender finds vote}"'] & a, d+1, \bot \\
 & a, d+1, \top
\end{tikzcd}$$

Attacker has smallest vote.

Defender has smallest vote.

$$\begin{tikzcd}
 & a, d+1, \bot \\
a, d, \bot \arrow[ur, "1-\alpha", "\text{defender finds vote}"'] \arrow[r, "\alpha/(a+d+1)", "\text{attacker finds smallest vote}"'] \arrow[dr, "\alpha\cdot(a+d)/(a+d+1)", "\text{attacker finds vote}"'] & a+1, d, \top \\
 & a+1, d, \bot
\end{tikzcd}$$

Figure 14: Markov chain of vote withholding to gain leadership in an epoch of $\mathcal{B}_k$. Smallest vote implies leadership.

Management Symposium. IEEE.

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

[53] Arvind Narayanan and Jeremy Clark. 2017. Bitcoin's Academic Pedigree. *Commun. ACM* 60, 12 (2017), 36–45.

[54] Fernando Nogueira. 2021. Bayesian Optimization: Open source constrained global optimization tool for Python. https://github.com/fmfn/BayesianOptimization.

[55] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Annual Technical Conference.* USENIX, 305–319.

[56] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Advances in Cryptology – EUROCRYPT 2017 (Lecture Notes in Computer Science, Vol. 10211)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, 643–673.

[57] Rafael Pass and Elaine Shi. 2017. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *International Symposium on Distributed Computing (Leibniz International Proceedings in Informatics, Vol. 91)*, Andréa W. Richa (Ed.). 39:1–39:16.

[58] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with Optimistic Instant Confirmation. In *Advances in Cryptology – EUROCRYPT 2018 (Lecture Notes in Computer Science, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 3–33.

[59] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234.

[60] Ling Ren. 2019. *Analysis of Nakamoto Consensus.* Cryptology ePrint 943. IACR.

[61] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 2016. Optimal Selfish Mining Strategies in Bitcoin. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 9603)*, Jens Grossklags and Bart Preneel (Eds.). Springer, 515–532.

[62] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22, 4 (1990), 299–319.

[63] Jakub Sliwinski and Roger Wattenhofer. 2021. Asynchronous Proof-of-Stake. In *Stabilization, Safety, and Security of Distributed Systems (Lecture Notes in Computer Science, Vol. 13046)*, Colette Johnen, Elad Michael Schiller, and Stefan Schmid (Eds.). Springer, 194–208.

[64] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. *SPECTRE: A Fast and Scalable Cryptocurrency Protocol.* Cryptology ePrint 1159. IACR.

[65] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. 2021. Phantom Ghostdag: A Scalable Generalization of Nakamoto Consensus. In *Conference on Advances in Financial Technologies.* ACM, 57–70.

[66] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure High-Rate Transaction Processing in Bitcoin. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 8975)*, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, 507–527.

[67] Volker Strassen. 1969. Gaussian Elimination Is Not Optimal. *Numer. Math.* 13, 4 (1969), 354–356.

[68] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Symposium on Principles of Distributed Computing.* ACM, 347–356.

[69] H. Yu, I. Nikolic, R. Hou, and P. Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *Symposium on Security and Privacy.* IEEE, 112–127.

[70] Ren Zhang and Bart Preneel. 2019. Lay down the Common Metrics: Evaluating Proof-of-Work Consensus Protocols' Security. In *Symposium on Security and Privacy.* IEEE, 1190–1207.

[71] Ren Zhang, Dingwei Zhang, Quake Wang, Shichen Wu, Jan Xie, and Bart Preneel. 2022. NC-max: Breaking the Security-Performance Tradeoff in Nakamoto Consensus. In *Network and Distributed Systems Security Symposium.* Internet Society.

[72] Roi Bar Zur, Ittay Eyal, and Aviv Tamar. 2020. Efficient MDP Analysis for Selfish-Mining in Blockchains. In *Conference on Advances in Financial Technologies.* ACM, 113–131.

A MARKOV CHAIN MODEL FOR CHAIN QUALITY AND CENSORING

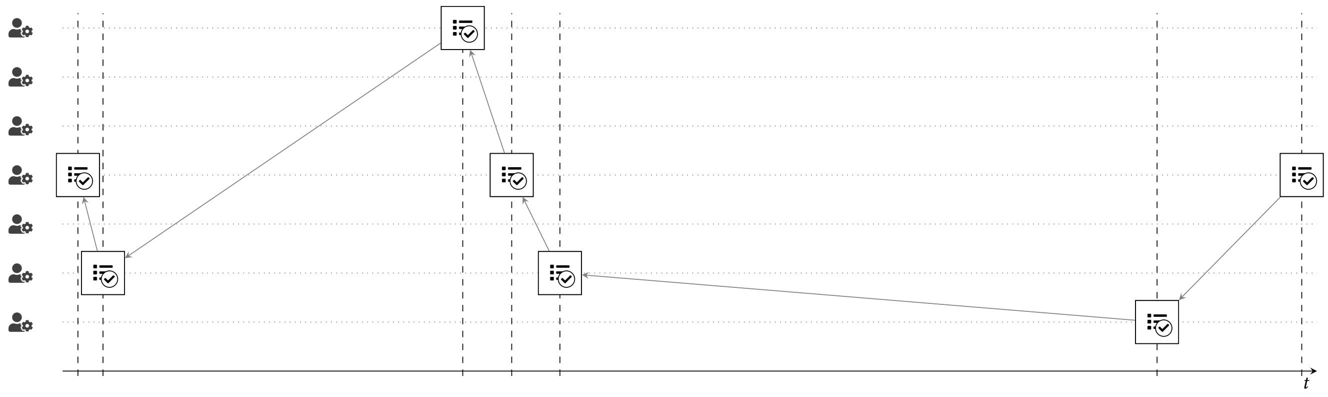
We describe the Markov chain model used in Section 4.2.2. Let the triple (a, d, l) be the current Markov state, where $a \in \mathbb{N}$ denotes the number of withheld attacker votes, $d \in \mathbb{N}$ denotes the number of votes found by the defender, and $l \in \{\bot, \top\}$ is true if the attacker currently holds the smallest vote. The initial state is $(1, 0, \top)$ with probability α and $(0, 1, \bot)$ otherwise.

Figure 14 depicts the probabilistic state transitions. As in Section 2.4, transitions occur at the ticks of the stochastic clock. With probability α, the attacker finds a new vote. The probability that it is the smallest (leading) vote is $1/(a + d + 1)$. This expression follows from $a + d$ old votes cutting the domain into $a + d + 1$ bins. As the hash function's outputs are indistinguishable from a uniform distribution, the expected bin size is $1/(a + d + 1)$. To simplify the figure, we do not show the two terminal states SUCCESS and FAIL. The former is reached when the attacker proposes a valid block ($l \wedge a + d \geq k$). Conversely, if $\neg l \wedge d \geq k$, the defenders propose a block. In all other cases, the epoch continues.

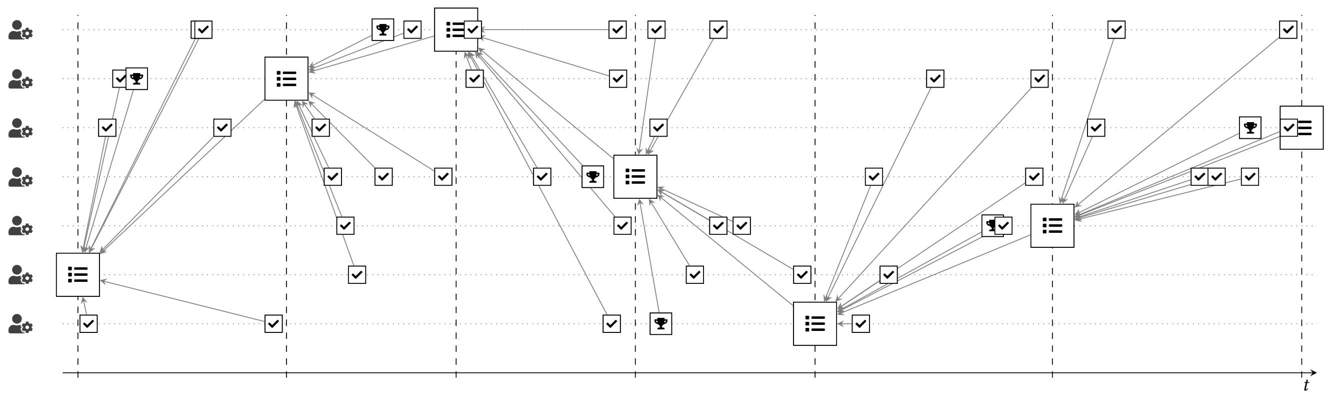
For $k = 51$, the resulting Markov chain has 5204 states. We evaluate it with Monte Carlo simulation for 1 000 000 epochs, $k = 51$ and α in the range $[0, 1/2]$. To validate these results in the context of the protocol and network latency, we implement the same attack in the network simulator and collect data from 64 independent executions by 1024 nodes up to block height 4096. In both cases, we measure chain quality and censorship susceptibility by counting terminations in the state SUCCESS. In addition, we analyze incentive compatibility by counting attacker votes.

✓	puzzle solution	□	message
🏆	leading puzzle solution	←	hash-reference
☰	state update	¦ ¦	epoch

(a) Legend.



(b) In Bitcoin, puzzles are solved sequentially. Puzzle solving times are exponentially distributed, implying irregular block intervals.



(c) In $\mathcal{B}_k$, multiple smaller puzzles are solved in parallel for each block. One of the $k = 8$ votes is chosen as leader and the corresponding miner proposes the next block (see Sec. 3.2.3). The embedded agreement protocol $\mathcal{A}_k$ guarantees ε-safety for the previous block (see Sec. 2). Observe that $\mathcal{B}_k$ has more regular block intervals and enables more frequent rewards for miners.

Figure 15: Simulated executions of Bitcoin and $\mathcal{B}_8$ on $n = 7$ nodes (y-axis) over time (x-axis).