HotPoW: Finality from Proof-of-Work Quorums

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

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

Abstract

A fundamental conflict of many proof-of-work systems is that they want to achieve inclusiveness and security at the same time. We analyze and resolve this conflict with a theory of proof-of-work quorums, which enables a new bridge between Byzantine and Nakamoto consensus. The theory yields stochastic uniqueness of quorums as a function of a security parameter. We employ the theory in HotPoW, a scalable permissionless distributed log protocol that supports finality based on the pipelined three-phase commit previously presented for HotStuff [65]. We evaluate HotPoW and variants with adversarial modifications by simulation. Results show that the protocol can tolerate network latency, churn, and targeted attacks on consistency and liveness with a small storage overhead compared to plain Nakamoto consensus and less complexity than protocols that rely on sidechains for finality.

1 Introduction

Bitcoin surprised scholars in distributed systems, as well as in security [11]. Authors have called the new composition of known concepts a "sweet spot" [64] in the design space for protocols, and praised the complex way the components are put together as a "true leap of insight" [49] of Nakamoto [48]. Likely the most intriguing part is the way Bitcoin uses proof-of-work puzzles to secure a distributed log.

The role of proof-of-work in Nakamoto consensus can be contemplated in several ways. First and most intuitively, the computational puzzles can be interpreted as a rate limit on new identities, which discourage Sybil attacks [19] in a lottery for blocks and new coins. Second, proof-of-work can be conceived as a game-proof variant of a probabilistic back-off mechanism, as used in media access control in computer networks. It reduces the risk of collisions when many nodes concurrently seek write access to a shared medium, the ledger. Proof-of-work has been formalized in cryptographic security models of Nakamoto consensus [27, 55]. However, we are not aware of work pointing out the fundamental conflict between inclusiveness and security inherent to the way proof-of-work is used in the known distributed log protocols.

This conflict precludes reliable and fast commits. Arguably, it is the reason why practical protocols trade finality for eventual consistency. But the lack of finality limits the applicability for high-value transactions [10, 29], a potential show-stopper discussed even beyond the technical community [4, 12].

We tackle this conflict directly, leading to a theory of proof-of-work quorums, which enables new ways of using proof-of-work in permissionless distributed log protocols. We propose one such protocol, HotPoW, demonstrating that finality with reliable and short time to commit is possible. Specifically, we do not rely on sidechains, a tool used in the literature to stack Byzantine on top of Nakamoto consensus [41, 53, 54]. Sidechains can add finality and increase throughput at the price of increased complexity, overhead, and tricky issues in the synchronization between layers [23, 41].

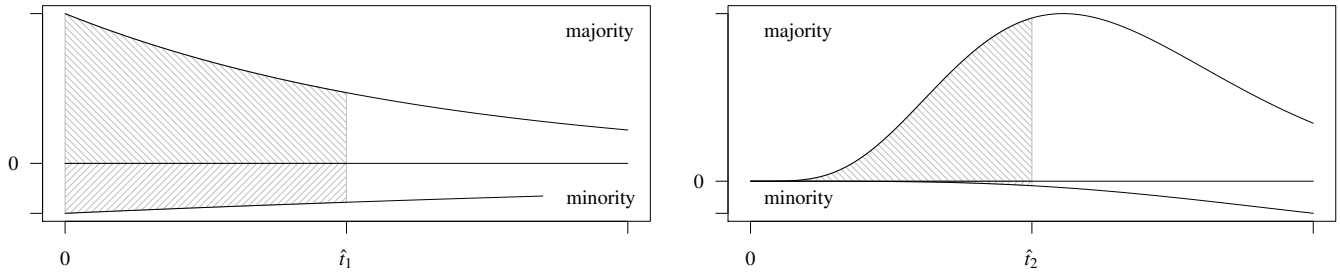
The proposed protocol is inspired by two recent breakthroughs: Bobtail [9] and HotStuff [65]. The former optimizes stochastic properties of the block delay in Nakamoto consensus. The latter adapts principles of Byzantine fault tolerance to blockchains in a clever way. It has received attention after Facebook's announcement to use it in LibraBFT [8].

We make the following contributions:

1. We draw attention to a fundamental conflict between inclusiveness and security in Nakamoto consensus and propose a principled resolution (Section 2).

2. We develop a theory of proof-of-work quorums where quorums are formed over votes generated by stochastic processes. We show that sufficiently large quorums are practically unique (Section 3).

3. We propose HotPoW, a protocol that finds consensus over a distributed log without requiring pre-defined identities. HotPoW scales at least as well as practical blockchain protocols and much better than Byzantine fault tolerance protocols. It relies on proof-of-work, but, unlike deployed systems using the longest chain rule, our



(a) Fair inclusion of minorities.

(b) Security margin against attacking minorities.

Figure 1: Probability densities of exponential (left) and gamma distributions (right) as functions over time for a $2/3$ majority and a $1/3$ minority (with flipped y-axis). The area under each curve represents the odds of winning a race.

construction supports a three-phase commit logic. State updates (transactions) are final after a predictable amount of time, and the probability of inconsistency is bounded according to our theory (Section 4).

4. We simulate executions of HotPoW as well as of variants with adversarial modifications. The results show that the protocol can tolerate network latency, churn, and targeted attacks on consistency and liveness at small overhead compared to the best deployed systems (Section 5).

Section 6 compares HotPoW to related works and discusses its limitations. Section 7 concludes. For replicability and future research, we make the protocol implementation and the simulation code available online.[1]

2 Intuition

The key conflict between inclusiveness and security faced by cryptocurrencies is as follows: *minorities should be encouraged to participate (inclusiveness), but they should not be able to make decisions alone (security).* Nakamoto consensus achieves inclusiveness by sacrificing security for an uncertain period of time (eventual consistency). This becomes problematic when irreversible real-world actions are taken based on unsettled transactions in the distributed log (double spending). A short and reliable time to commit would mitigate this risk.

Recall that Nakamoto consensus prioritizes inclusiveness by using a puzzle as gatekeeper to participation. The protocol specifies a repeated race for the first puzzle solution. Each winner proposes a state update and receives some reward. Most cryptocurrencies use puzzles—moderately hard functions—for which iterative trial and error is the best known solving algorithm. Such puzzles imply exponentially distributed solving time. Figure 1a shows the probability distributions for the solving times of a $2/3$ majority of solving power compared to a $1/3$ minority. The expected time of the end of the race is marked with $\hat{t}_1$ (in Bitcoin $\hat{t}_1 \approx 10$ minutes). Consequently, the area under each curve represents the odds of winning the race. Observe that the minority has a fair chance. This makes the protocol inclusive, but also implies that minorities have a significant chance of directly writing state updates. For improved security, we would prefer a distribution such that the minority's area under the curve is small (ideally negligible), as displayed in Figure 1b.

Since the puzzle of Nakamoto consensus behaves like in Figure 1a, a single state update is not reliable. As a result, users are recommended to wait for multiple consecutive blocks before acting upon a payment. The time needed for sequentially solving k exponential puzzles is gamma distributed with shape parameter k. In fact, Figure 1b shows the gamma distribution for $k = 6$. Note the significant gap between minority and majority: it is unlikely that a minority can generate a sequence of 6 state updates before the majority does so. In this sense, multiple puzzle solutions qualify a majority, while a single one does not.

In Nakamoto consensus, security comes at the price of waiting for multiple solutions. Bitcoin's convention of $k = 6$ implies an expected waiting time of $\hat{t}_2 \approx 60$ minutes, which is arguably too slow for many applications. Besides, Nakamoto consensus does not give a rationale on how to choose k.

A key idea for resolving this conflict is to break the one-to-one relationship between puzzle solutions and blocks. Instead of requiring a single 10 minute puzzle per block, HotPoW asks for k easier puzzles each expected to take $10/k$ minutes. In other words, HotPoW achieves security by appending puzzle solutions *in parallel* rather than sequentially, as illustrated in Figure 2. Since the puzzles are independent, we end up with the same block rate but k times the number of solutions. The expected computational effort stays the same, but we accumulate a qualifying number of solutions for *every* block. This means we get the shape of Figure 1b much faster: $\hat{t}_2 \approx \hat{t}_1$.

For a principled construction of HotPoW, we reduce the payload "authenticated" [5] by proof-of-work to a minimum:

1. a reference to a recent point in time (e. g., a hash link to the last seen block)

[1] https://github.com/pkel/hotpow/tree/arxiv_v3

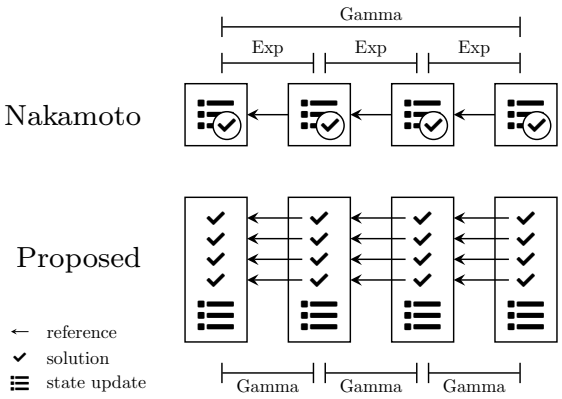


Figure 2: Sequential puzzles in Nakamoto consensus imply exponentially distributed block delays. Multiple smaller puzzles in parallel lead to a gamma distribution for each block.

2. a reference to an identity (public key or commitment)

A triple of a puzzle solution and these two references forms a verifiable ephemeral identity. The puzzle solution binds resources in order to prevent Sybil attacks, the reference in time ensures freshness, and the identifier enables authorized actions, such as claiming a reward.

The main difference between proof-of-work systems and the well-studied class of Byzantine fault tolerant (BFT) systems [14, 20, 44] is that the former do not rely on external identification of the participating nodes. Inspired by the early work of Aspnes et al. [3], HotPoW uses proof-of-work to bootstrap ephemeral identities and plugs them into HotStuff [65], a state of the art blockchain-based BFT system. In HotStuff, each block carries a certificate about a qualified majority of nodes (quorum) confirming the last seen block. HotStuff's proof of finality is based on the qualifying properties of each quorum. This motivates us to explore whether and to what extent a set of proof-of-work solutions can qualify a majority. In Section 3, we will show that qualifying majorities are possible within a single block. This allows us to transfer HotStuff's finality to the permissionless setting.

The recurse to HotStuff enables us to fix the number of blocks to wait before accepting a state update as final at the necessary number of phases to commit, thereby resolving a drawback of Nakamoto consensus. As illustrated in Figure 3, HotStuff uses a three-phase commit, which can be pipelined for subsequent state updates on a blockchain. In a nutshell, the first phase locks a single proposal, the second phase confirms majority uptake of this lock, and the third phase ensures that the knowledge of this knowledge is propagated. We refer to [65] for the rationales and failure modes. In this sense, HotPoW parallelizes not only puzzle solutions but also the phases of the commit logic.

Another advantage of the gamma distribution per block is a reduction in the variance of block delays compared to the exponential distribution implied by the puzzle. While

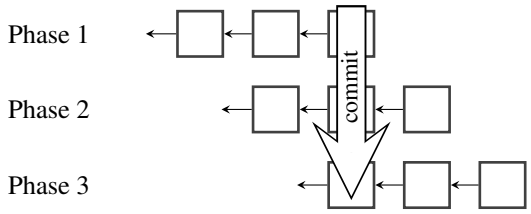


Figure 3: Pipelined three-phase commit on a blockchain in HotStuff and HotPoW.

the commit pipeline gives us finality after three blocks, the reduced variance translates this into a reliable time to commit. The theory in the following section shows formally how all this is related to the quorum size, HotPoW's new security parameter.

3 Proof-of-Work Quorums

Quorums are central to the design and analysis of BFT protocols. The typical Byzantine setting assumes a set of $n = 3f+1$ identified nodes, of which at most f deviate from the protocol. A set of $2f+1$ votes for the same value is called a quorum. If correct nodes vote at most once, quorums imply a majority decision and thus are unique. The uniqueness may be violated in two situations.

BFT-1 More than n nodes vote.

BFT-2 More than f nodes vote more than once.

Practical systems avoid BFT-1 using preset identities for all nodes and rule out BFT-2 by assumption.

Proof-of-work enables systems where agents can join and leave at any time without obtaining permission from an identity provider or gatekeeper [48]. This difference is often implied in the terms "permissioned" and "permissionless". In the permissionless case one must distinguish between *agents* and *nodes*. Agents are entities participating in a distributed system. An agent can operate any number of nodes. Colluding parties are interpreted as a single agent.

We introduce the notion *proof-of-work quorum* for a set of votes where each vote requires a solution to a proof-of-work puzzle. Since the puzzle solving time is probabilistic, the uniqueness of quorums cannot be absolute. In contrast to the Byzantine setting, we have to consider three failure modes:

PoW-1 The total compute power of the network is higher than assumed.

PoW-2 The adversary controls more than the assumed fraction of compute power.

PoW-3 A random bad realization happens.

The failure modes PoW-1 and PoW-2 correspond to the Byzantine failure modes BFT-1 and BFT-2. Our goal is to understand the new failure mode PoW-3 and how it affects the potential ambiguity (violation of uniqueness) of quorums.

Definition 1 (Proof-of-work process)**.** A proof-of-work process is a stochastic count process where each event assigns one *ability to vote* (ATV) to one agent. Each ATV can be used by the agent it is assigned to, to vote once for one value.

We adopt the notion of a quorum from the BFT literature [45, 65] except that we will apply it to votes from ATVs rather than identified nodes.

Definition 2 (k-quorum)**.** A set of k votes for the same value x is called a k-quorum for x.

Observing a k-quorum implies that at least k ATVs have been used, hence the proof-of-work process must have assigned at least k ATVs. This connects to time.

Definition 3 (Optimistic quorum time)**.** The time at which the proof-of-work process assigns the k-th ATV is called optimistic k-quorum time. For a proof-of-work process P and quorum size k it is formally defined by the random variable

$$T_{P,k} := \inf\{t \in \mathbb{R}_{\geq 0} \mid P(t) \geq k\}\,.$$

$T_{P,k}$ is the earliest point in time at which a k-quorum is feasible. A k-quorum is only possible at exactly $T_{P,k}$, if all assigned ATVs are used to vote for the same value.

A quorum for x is ambiguous if there is another quorum for $y \neq x$. Since each ATV can be used for at most one value, ambiguous k-quorums are only possible when the proof-of-work process has assigned at least $2k$ ATVs.

Definition 4 (Probability of ambiguity)**.** For a proof-of-work process P and quorum size k we define the *probability of ambiguity* (POA) as

$$\mathrm{poa}_{P,k}(t) := \mathbf{Pr}\big[P(t) \geq 2k\big]\,.$$

For puzzles where the best known solving algorithm is independent trial and error, the stochastic process is instantiated by the Poisson process P_λ. This is because if each puzzle solution generates one ATV, the time between consecutive ATVs is exponentially distributed with rate λ.

Lemma 1. *The POA for the Poisson process P_λ is given by*

$$\mathrm{poa}_{P_\lambda,k}(t) = 1 - e^{-\lambda t} \sum_{i=0}^{2k-1} \frac{(\lambda t)^i}{i!}\,.$$

Proof. See Appendix A. □

Lemma 2. *The optimistic k-quorum time for the Poisson process is Erlang distributed with shape parameter k and rate parameter λ, in short*

$$T_{P_\lambda,k} \sim \mathrm{Erlang}(k,\lambda)\,.$$

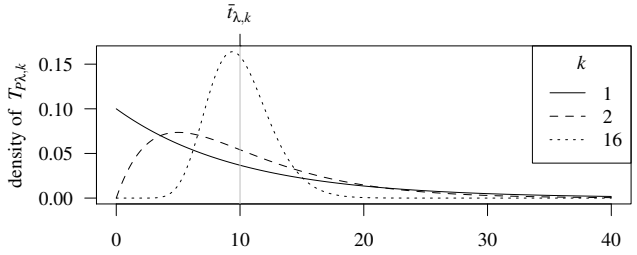


Figure 4: The density of the distribution of the optimistic k-quorum time based on P_λ with rate $\lambda = k/10$ (minutes).

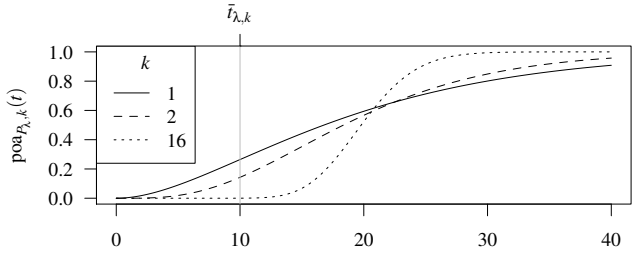


Figure 5: The probability of ambiguity as a function of time for quorum sizes $k = 1, 2$, and 16 and $\lambda = k/10$ (minutes).

Proof. See Appendix A. □

Corollary 1. *The expected optimistic k-quorum time for the Poisson process is*

$$\bar{t}_{\lambda,k} := \mathbf{Ev}\big[T_{P_\lambda,k}\big] = k/\lambda\,.$$

Proof. The statement follows from Lemma 2 and the definition of the Erlang distribution [63, p. 146]. □

Figure 4 illustrates the distribution of the optimistic k-quorum time for $k \in \{1, 2, 16\}$ based on the Poisson process. In order to compare quorum sizes greater than one to an ideal Bitcoin ($k = 1$, $\bar{t}_{\lambda,k} = 10$ minutes), we choose $\lambda = k/10$.

Figure 5 shows the POA for different quorum sizes as a function of time. Again, we adjust the rate such that the expected optimistic k-quorum time is 10 minutes. Observe that the POA increases over time as the number of ATVs grows. More importantly, the POA at the expected optimistic quorum time decreases in the quorum size k.

In order to isolate the effect of k, we evaluate the POA at fixed time $\bar{t}_{\lambda,k}$, which lends itself to a closed form.

Corollary 2. *For the Poisson process, the POA at expected optimistic k-quorum time is given by*

$$\mathrm{poa}_{P_\lambda,k}(\bar{t}_{\lambda,k}) = 1 - e^{-k} \sum_{i=0}^{2k-1} \frac{k^i}{i!}\,.$$

Proof. By inserting Corollary 1 into Lemma 1. □

Observe that the POA at expected optimistic quorum time is independent of λ. This is useful as λ may measure the total compute capacity in proof-of-work networks, which is not necessarily known to each agent.

Since ambiguity causes failure, and the probability of ambiguity vanishes as k grows, k becomes a security parameter. In order to relate it to other security parameters, such as the key size, we adopt the common definition of negligibility from cryptography (i. e., asymptotic decline faster than any polynomial) and state the following theorem.

Theorem 1. *For the Poisson process, the probability of ambiguity at the expected quorum time is negligible in the quorum size k.*

Proof. See Appendix A. □

Remark (Validation on Bitcoin). For Bitcoin parameters ($k = 1, \lambda = 0.1$), the POA at $\bar{t}_{\lambda,k}$ is $p = 0.2642$. This part of the theory can be validated on historical data. We estimate the expected block delay by averaging the differences between consecutive block time stamps over 2017–2018.[2] The estimated average block delay is $\hat{t} = 9.52$ minutes. The ratio of cases with more than two blocks arriving within $\hat{t}$ is $\hat{p} = 0.2606$. This estimate should be slightly below p because our historic data does not contain orphaned blocks. Since $p \approx \hat{p}$, we conclude that the theory applies to Bitcoin.

The implication of this theory for protocol design is that larger quorums reduce the probability of ambiguity. The (close to) exponential decay makes it conceivable to choose parameters such that quorums are practically unique. This allows us to use a notion of quorum uniqueness with ephemeral identities generated by proof-of-work.

4 HotPoW

Now we specify HotPoW, a distributed log protocol secured by a proof-of-work process (Def. 1) and k-quorums (Def. 2).

We present HotPoW using pseudocode and a mixture of event-driven and imperative programming. A less ambiguous implementation in OCaml is provided online.[3]

4.1 Prerequisites

We assume interfaces to the network and application layers (Fig. 6), and the availability of cryptographic primitives.

4.1.1 Broadcast Network The proposed protocol requires a (potentially unreliable) network broadcast. We abstract from the exact implementation and assume that scheduling an event $\langle$ send $\mid m \rangle$ results in the message m being sent to (most of) the other HotPoW nodes. On the receiving side, the implementation delivers message m' by scheduling $\langle$ receive $\mid m' \rangle$.

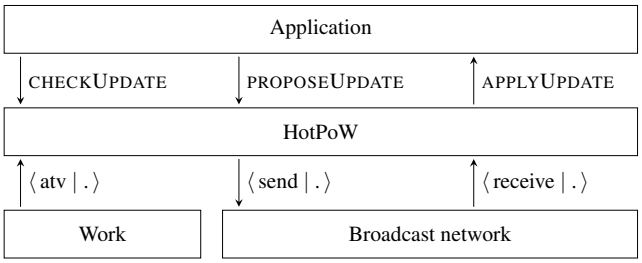


Figure 6: Interaction between the protocol (HotPoW), the application, the proof-of-work process, and the network. Arrows denote information flows and not necessarily call directions.

4.1.2 Application HotPoW implements a distributed log which may serve as a base for different applications [1, 43, 60]. For example, a simple cryptocurrency could append lists of transactions which jointly form a ledger. More advanced applications could add scalability layers that only record key decisions in the distributed log while handling other state updates separately [23, 41, 54].

We abstract from the application logic using three procedures HotPoW can call. CHECKUPDATE takes an application state and a state update as arguments and returns true if the state update is valid. APPLYUPDATE takes an application state and a state update and returns an updated state. PROPOSEUPDATE takes an application state and returns a valid state update. We are agnostic about direct access of the application to the broadcast network. For example, cryptocurrencies share transactions provisionally before they are logged in blocks.

4.1.3 Cryptography HotPoW uses cryptographic hash functions for the hash-linked list and the proof-of-work process. We separate these two concerns and use two different hash functions, $\mathcal{H}_{\text{list}}$ and $\mathcal{H}_{\text{pow}}$. While it is sufficient that $\mathcal{H}_{\text{list}}$ is cryptographically secure, HotPoW requires the same stronger assumptions for $\mathcal{H}_{\text{pow}}$ as Bitcoin [1]. Since this difference is not central, the reader can safely assume $\mathcal{H}_{\text{list}} = \mathcal{H}_{\text{pow}} = \text{SHA3}$.

HotPoW also requires a digital signature scheme [39, Def. 12.1, p. 442]. We assume a secure implementation is given by the three procedures GENERATEKEYPAIR, CHECKSIGNATURE, and SIGN. Every node holds an asymmetric key pair (me, secret).

4.2 Protocol

4.2.1 Local Block Store HotPoW nodes maintain a local tree of hash-linked blocks and a reference to the preferred chain (head). They store blocks together with the associated

[2] We choose this time range because the block time stamps were less accurate in the more distant past as the data field was used for other purposes.

[3] https://github.com/pkel/hotpow/tree/arxiv_v3

application state, the block height, and a set of corresponding votes (see Listing 4.1). The block storage is indexed by $\mathcal{H}_{\text{list}}$.

Listing 4.1 Local Block Store

```
1:  procedure STORE(block B)
2:      h ← H_list(B)
3:      if h ∉ blocks then
4:          parent ← blocks[B.parent]
5:          blocks[h].parent ← parent
6:          blocks[h].state ← APPLYUPDATE(parent.state, B.payload)
7:          blocks[h].height ← parent.height + 1
8:          blocks[h].votes ← ∅
9:          blocks[h].block ← B
10:         UPDATEHEAD(h)
```

4.2.2 Votes As mentioned in Section 2, a vote in HotPoW is a triple (r, p, s), where r is a reference to a previous block, p is the public key of the voter, and s is a puzzle solution. A vote (r, p, s) is valid if $\mathcal{H}_{\text{pow}}(r, p, s) \leq t_{\text{v}}$, where t_{v} denotes the proof-of-work threshold and represents HotPoW's difficulty parameter. HotPoW nodes maintain a set of valid votes for each block. The procedure COLLECT (Listing 4.2) adds a valid vote (r, p, s) to the block referenced by r and, if necessary, updates the preferred chain (see Sect. 4.2.9 below).

Listing 4.2 Collection of Votes

```
11: procedure COLLECT(r, p, s)
12:     if H_pow(r, p, s) ≤ t_v then
13:         blocks[r].votes ← blocks[r].votes ∪{(p, s)}
14:         UPDATEHEAD(r)
```

4.2.3 Quorums As defined in Section 3, a k-quorum is a set of k votes for the same reference. We represent such quorums as lists. Since the reference is the same for all votes, we omit it from the list. A list $L = \{(p_i, s_i)\}$ represents a valid k-quorum for r, if the following conditions hold:

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

The first condition enforces the quorum size. The second condition ensures that all votes are valid. The third condition imposes a canonical order which we use for leader election. We intentionally allow single nodes providing multiple votes. Sibyl attacks are mitigated by the scarcity of votes.

4.2.4 Leader Election A quorum can only be formed at optimistic quorum time (Def. 3) if all nodes vote for the same block. We facilitate coordination by electing a leader who is responsible for proposing a new block. This election is based on the proof-of-work quorum: the leader is identified by the smallest vote. According to Section 4.2.3 Condition 3, this vote is also the first element of the quorum. Leaders authenticate their proposals for the next block using SIGN and their private key. Everyone verifies proposals with the first public key in the quorum.

4.2.5 Blockchain The global data structure of the protocol is a hash-linked list of blocks. Each block consists of a hash reference to its predecessor (parent), a proof-of-work quorum for this predecessor, a payload, and a proof of leadership (signature). The references to parent blocks are established by the collision-resistant hash function $\mathcal{H}_{\text{list}}$. The payload is a state update to the application implemented on top of the distributed log (see Sect. 4.1.2).

With quorums, leader election, and state updates defined, we are in the position to present HotPoW's block validity rule in Listing 4.3. The loop iterates over the quorum, counts the votes, verifies them, and checks their canonical order. The boolean conjunction in line 22 verifies the remaining condition of the quorum, leadership, and the validity of the proposed state update.

Listing 4.3 Block Validity

```
15: procedure VALIDBLOCK(block B)
16:     (c, h) ← (0, 0)
17:     for all (p, s) in B.quorum do
18:         h' ← H_pow(B.parent, p, s)                    ▷ predecessor!
19:         if h' > t_v then return false                 ▷ quorum condition 2, Sect. 4.2.3
20:         if h' < h then return false                   ▷ quorum condition 3, Sect. 4.2.3
21:         (c, h) ← (c + 1, h')
22:     return
            c = k ∧                                       ▷ quorum condition 1, Sect. 4.2.3
            CHECKSIGNATURE(B.quorum.[0].p, B) ∧
            CHECKUPDATE(blocks[B.parent].state, B.payload)
```

A key difference to Nakamoto consensus is that the proof-of-work solutions in the quorum are bound to the previous block and not to the state update of the proposed block (see line 18). This implements the separation of puzzle solutions from block proposals and enables parallel puzzle solving (see Sect. 2).

4.2.6 Proposing Nodes assume leadership whenever possible. If so, the procedure PROPOSEIFLEADER (Listing 4.4) obtains a state update from the application, integrates it into a new valid block, and shares it with the other nodes.

Listing 4.4 Block Proposals

```
23: procedure PROPOSEIFLEADER(r)
24:     if ∃ valid k-quorum Q ⊂ blocks[r].votes where Q[0].p = me then
25:         B.parent ← r
26:         B.quorum ← Q
27:         B.payload ← PROPOSEUPDATE(blocks[r].state)
28:         B.signature ← SIGN((B.parent, B.quorum, B.payload), secret)
29:         STORE(B)
30:         schedule ⟨ send | block B ⟩
31:         return true
32:     else return false
```

4.2.7 Commit Proposals become final after the three-phase commit. Each subsequent block carries a quorum that completes one phase, like in HotStuff (see Sect. 2). Consequently, the most recent application state can be retrieved from the local block store as shown in Listing 4.5.

Listing 4.5 Reading Application State

```
33: procedure READSTATE
34:     return blocks[head].parent.parent.parent.state
```

4.2.8 Conflict Resolution The commit becomes effective after three blocks, but we have to consider conflicting block proposals at the uncommitted frontier. For example, when more than k votes exist, the leader election is not unique. Moreover, a malicious leader can send different proposals without solving additional proof-of-work puzzles. Nodes resolve such conflicts based on the progress towards the *next* quorum.

4.2.9 Block Preference When learning of a new block or vote, nodes update their preferred chain according to a modified version of Nakamoto's longest chain rule. HotPoW adapts it to include information on quorum progress (Sect. 4.2.8) and reject changes to already committed state (Sect. 4.2.7). Procedure UPDATEHEAD (Listing 4.6) takes a candidate block reference and updates the preferred chain if necessary.

Listing 4.6 Block Preference

```
35: procedure UPDATEHEAD(r)
36:     H ← blocks[head]
37:     R ← blocks[r]
38:     d ← R.height − H.height
39:     if d > 0 ∨ (d = 0 ∧ |R.votes| > |H.votes|) then
40:         while d > 0 do (R, d) ← (R.parent, d − 1)
41:         if H.parent.parent.parent.block = R.parent.parent.parent.block then
42:             head ← r
```

4.2.10 Main Program Listing 4.7 shows the set of event handlers that tie everything together and define a HotPoW node. The execution is initiated by scheduling the ⟨ init ⟩ event. The listing shows how nodes assume leadership upon completing a suitable quorum with an ATV of their own (line 50), or votes received from others, either directly (line 54) or as part of a block proposal (line 58). In the last case, if more than k votes exist, it can happen that a node replaces the leader. It proposes a block of its own by reusing votes contained in the received proposal. This is possible because votes in HotPoW reference the previous block and not the current proposal. The possibility of reusing votes reduces wasted work compared to orphans in Nakamoto consensus, a problem that has been studied separately [61]. It also provides robustness against leader failure (see Sect. 5.1.3).

Line 48 handles ATVs. If the node cannot lead a quorum, it broadcasts the vote. The last missing part is how ATVs can be scheduled, which we discuss next.

Listing 4.7 The HotPoW Protocol

```
43: upon ⟨ init ⟩ do
44:     me, secret ← GENERATEKEYPAIR( )
45:     head ← genesis                          ▷ hard-coded magic value
46:     blocks[genesis].state ← S_0             ▷ application's initial state
47:     blocks[genesis].height ← 0
48: upon ⟨ atv | s ⟩ do
49:     COLLECT(head, me, s)
50:     if not PROPOSEIFLEADER(head) then
51:         schedule ⟨ send | vote (head, me, s) ⟩
52: upon ⟨ receive | vote (r, p, s) ⟩ do      ▷ sent by other node in line 51
53:     COLLECT(r, p, s)
54:     PROPOSEIFLEADER(r)
55: upon ⟨ receive | block B ⟩ do      ▷ sent by other node in line 30 (Sect. 4.2.6)
56:     for all (p, s) in B.quorum do
57:         COLLECT(B.parent, p, s)
58:     PROPOSEIFLEADER(B.parent)
59:     if VALIDBLOCK(B) then STORE(B)
```

4.2.11 Work Agents can participate in the quorum finding process by computing ATVs on their nodes. For completeness, Listing 4.8 shows the trial-and-error algorithm which schedules solutions suitable for votes ($\leq t_v$). Alternatively, agents can search ATVs with the help of other machines, possibly in parallel and using specialized hardware. Figure 6 reflects this by splitting the lower layer in network and work.

Listing 4.8 Puzzle Solving

```
60: procedure WORK
61:     draw random number n
62:     if H_pow(head, me, n) ≤ t_v then
63:         schedule ⟨ atv | n ⟩
64:     WORK
```

Figure A.1 in the appendix visualizes an execution of HotPoW by correct nodes and compares it to Nakamoto consensus.

4.3 Incentives

It is possible to motivate participation in HotPoW by rewarding puzzle solutions. This requires some kind of virtual asset that (at least partly) fulfills the functions of money [34, p. 1] and can be transferred to a vote's public key. Claiming the reward for (r, p, s) depends on the corresponding secret key.

HotPoW could adopt Bobtails's constant reward per vote [9]. Rewarding votes instead of blocks would ensure inclusiveness without compromising security (see Sect. 2). Votes occur k times more frequently than blocks. HotPoW's mining income would thus be less volatile than in Nakamoto consensus. This reduces the pressure to form mining pools.

However, it is not trivial to establish if constant rewards are incentive compatible because the utility of the reward *outside* the system may affect the willingness to participate *in* the system and thereby make λ endogenous [18, 56]. This implies that rewards must be treated jointly with the assumptions preventing the failure modes PoW-1 and PoW-2. We are unaware of protocol analyses that solve this problem convincingly.

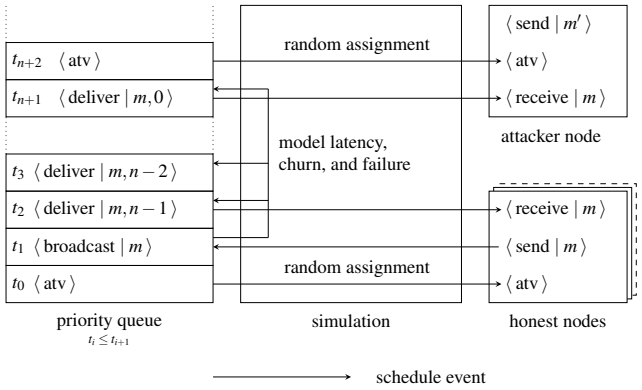


Figure 7: Schematic overview of the discrete event simulation.

On a more general note, designing protocols like economic mechanisms by incentivizing desired behavior sounds attractive because there is some hope that the assumption of honest nodes can be replaced by a somewhat weaker assumption of rational agents [26, 31]. In this spirit, Badertscher et al. [6] present positive results for Bitcoin in a discrete round execution model and under assumption of a constant exchange rate. However, many roadblocks remain. Agents' actions are not fully observable (e. g., information withholding) and preference orders are not fully knowable, hence rationality is not precisely defined. Side-payments (bribes), which cannot be ruled out, pose an insurmountable challenge for mechanism design [10, 12, 37]. For distributed logs, which work inherently sequential, this approach may even be thwarted by negative results on the existence of unique equilibria in repeated games [24]. For these reasons, we skip the mechanism design aspects and limit our contribution to transferring Byzantine consensus to proof-of-work scenarios. In other words, HotPoW supports incentives for inclusiveness, but its security intentionally does not rely on incentives.

5 Evaluation

We implement HotPoW in OCaml and evaluate it in a network of 1000 nodes using a discrete event simulation. We average over 100 independent executions of the first 500 blocks. All results are reproducible with the code provided online.[4]

The simulation maintains state for all simulated nodes separately. Events are stored in a priority queue, with keys representing points in time. Events are scheduled by inserting them into the queue. There are three types of simulation events: ⟨ atv ⟩, ⟨ broadcast ⟩ and ⟨ deliver ⟩. The simulation's main loop takes the first event from the queue and handles it by interacting with the nodes in the following way (also see Fig. 7).

[4] https://github.com/pkel/hotpow/tree/arxiv_v3

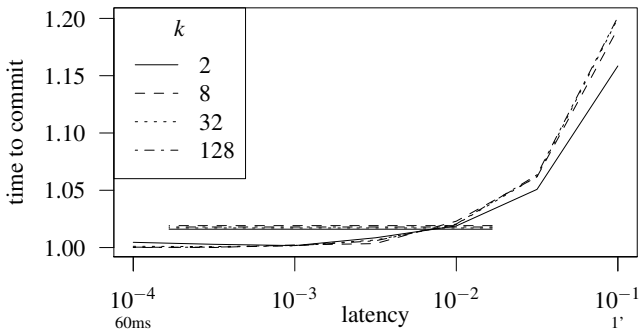


Figure 8: The effect of latency on the time to commit. The latency is stated relative to the optimistic quorum time (in small print for a quorum time of 10'). The horizontal lines show a realistic scenario (10s for blocks, 100ms for votes).

Proof-of-Work When taking an ⟨ atv ⟩ event from the queue, the simulation randomly and independently assigns an ATV to a node. The simulation executes the assignment by invoking the ⟨ atv ⟩ event handler on the receiving node. Then, it schedules the next ATV with a random, exponentially distributed time delta. This simulates a proof-of-work process according to Def. 1. The simulation does not perform actual work by setting the vote threshold t_v to the maximum; meaning puzzles are trivial to solve.

Broadcast Nodes invoke the broadcast logic by scheduling local ⟨ send | . ⟩ events. The simulation translates them to global ⟨ broadcast | . ⟩ events. For each broadcast event, the simulation schedules ⟨ deliver | . ⟩ events for each node except the sender. During this step, the simulation injects latency and simulates churn and leader failure. Delivery events are handled by invoking the ⟨ receive | . ⟩ handler on the receiving node.

5.1 Robustness

We evaluate the robustness in terms of latency, churn, and leader failure. In all simulation runs we check for inconsistent committed state, which did not occur.

5.1.1 Latency We model the effect of latency by injecting a random time delay between broadcast send and message delivery. We draw delays from an exponential distribution with fixed expectation, independently for each node and delivery. Latency causes temporal state inconsistencies. In these periods, nodes spend their ATVs on extending superseded blocks, or even produce temporal forks. We observe that largely independent of the quorum size k, expected latencies below 1 % of the expected block time (Bitcoin: 6 seconds) have marginal impact, while latencies in the order of 10 % of the expected block time (Bitcoin: 60 seconds) delay the commit by about 20 %. Figure 8 visualizes these results.

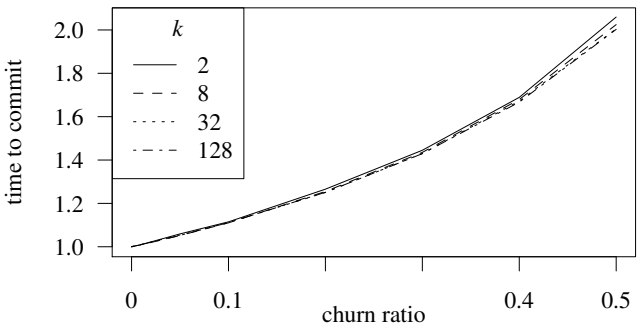


Figure 9: The effect of churn on the time to commit.

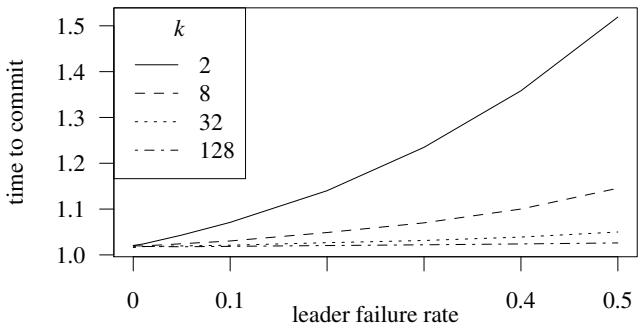


Figure 10: The effect of leader failure on the time to commit.

Empirical measurements [15, 16, 29] suggest that the propagation time of Bitcoin blocks ($\approx$ 500 KB) is about 9 seconds on the Internet. If we take this as an upper bound, we can argue that HotPoW tolerates practical latencies. Moreover, most of HotPoW's messages are votes. They are multiple orders of magnitude smaller (72 B; see Sect. 5.3), fit into a single packet, and are much easier to verify than Bitcoin blocks. Results of a simulation with different latencies for blocks (10s) and votes (100ms) suggest that HotPoW can run at Internet scale with lower expected block time than 10 minutes.

5.1.2 Churn We simulate churn by muting a fraction (churn ratio) of random nodes for 10 times the expected block time. Muted nodes can receive ATVs but do not send or receive messages. Accordingly, the ATVs assigned to muted nodes represent lost work. We expect that the time to commit is inversely proportional to the churn ratio: if 50 % of the nodes are muted, the time to commit is twice as long, independent of the quorum size. Figure 9 supports this claim.

5.1.3 Leader Failure Leaders may fail to propose blocks. We model such failures by dropping block proposals randomly with constant probability (leader failure rate).

In Nakamoto consensus, lost proposals imply a full block worth of wasted work. HotPoW 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, with increasing quorum size the robustness to leader failure should improve. The results in Figure 10 (with realistic 10s/100ms latency) and Figure A.2 (without latency to isolate effects) 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 nodes at discretion with 50 % success probability. Still, for large quorum sizes the time to commit is not longer than under the extreme latencies discussed in Section 5.1.1.

The robustness against churn and leader failure emerges from HotPoW's novel approach to form short-lived committees from ephemeral identities. This maintains liveness even under the threat of powerful network-level attacks. We move on to the discussion of attacks on the protocol layer.

5.2 Security

The security evaluation draws on the framework by Zhang and Preneel [66]. It distinguishes the security aspects proof-of-work blockchains should fulfill: chain quality, incentive compatibility, subversion gain, and censorship susceptibility.

The authors suggest Markov Decision Processes (MDP) as method and apply it to several variants of Nakamoto consensus. However, state explosion prevented them from modeling Bobtail,[5] because it ranks proof-of-work solutions by magnitude. Since HotPoW adopts this ranking for the leader election (Sect. 4.2.4), it does not seem readily amenable to MDPs, either. We thus resort to informal reasoning and simulation.

Following the convention in the literature, we assume two agents. Let λ be the total compute power. The attacker has $\alpha \cdot \lambda$ compute power, the honest agent controls the rest. The honest agent operates correct nodes, while the attacker operates a single node that may deviate from the protocol specification.

5.2.1 Subversion Gain The canonical example for subversion gain in cryptocurrencies is double spending: the attacker wants at least one of the honest nodes (the merchant) to act on inconsistent state. HotPoW supports commits, hence we neither need to consider the possibility of history rewriting nor the double spending of *un*committed transactions.[6] Nakamoto consensus suffers from these problems [2, 12, 29, 33, 38].

The only remaining strategy is splitting the network so that the recipients of at least two different double-spend transactions commit to different states. This loss of consistency would materialize in permanent forks that require out-of-band resolutions (triggered by an else-branch after code line 42).

[5] Zhang and Preneel [66] were aware of Bobtail and chose not to model it. This is confirmed in private communication with the authors of Bobtail [9].

[6] Sound applications on a system with finality wait until the commit. HotPoW can be parametrized to acceptable commit times for economic exchanges between humans. (High-frequency trading needs other architectures.)

In order to understand how HotPoW ensures consistency, it is instructive to recall the block preference rule in Sect. 4.2.9. Assume counterfactually that nodes never update their value according to received votes. Then, an attacker who becomes the leader could send different proposals to each node. This would fragment the honest nodes' compute power and give the attacker time to form six quorums, three per conflicting state. The probability of the attacker becoming leader is at least α in each round. This would be a catastrophic attack.

The actual block preference rule selects the value with the highest progress among all known proposals. Therefore, as soon as the first vote is received from an honest node, all honest nodes converge to a single value. As a result, the attacker would have to form six complete quorums in the time the honest nodes get assigned a single ATV and broadcast the corresponding vote. Since $\frac{6k}{\alpha} \gg \frac{1}{1-\alpha}$, such an attack becomes infeasible for large quorum sizes k and $\alpha < 1/2$.

5.2.2 Censoring In the censoring scenario, the attacker wants to control the values on which consensus is achieved for some time. This means he has to be elected as leader in multiple (m) consecutive blocks.

We start with the probability of an attacker becoming the leader in a single round. Without deviating from the protocol, he leads with probability α. This means he could successfully censor HotPoW for m consecutive blocks with probability α^m.

However, naively following the protocol is not the best censoring strategy. Taking inspiration from the work on selfish mining [22, 40, 59], we argue that an attacker can do better by withholding information. A selfish miner in Nakamoto consensus withholds complete blocks, such that other miners work on an irrelevant part of the chain. HotPoW has a more granular type of information: an attacker might withhold his votes. A censoring attacker would release his votes only when the release implies leadership. In practice, this means that a censoring attacker does not share votes, he only proposes blocks. Using this strategy, the attacker can delay the next quorum until the honest nodes can form one without the attacker's votes. This time window increases the attacker's odds of becoming the leader.

We implement this *censor* strategy and instantiate it in a special attacker node of the simulation environment (see Fig. 7). We bias the assignment of ATVs towards this node such that it posesses computational power α. We routinely check for forks, but do not find any. We count how many of the committed blocks are proposed by the attacker in order to estimate the probability of leadership per round. Figure 11 shows this estimate as a function of the quorum size for different attacker strengths α. Using the described withholding strategy, an $\alpha = 1/3$ attacker contributes roughly 42 % ($\alpha = 1/2$: 64 %) of the blocks. For comparison, the upper bound for block withholding strategies for the same attacker on Nakamoto consensus is 50 % ($\alpha = 1/2$: 100 %) [59].

We additionally validate the results on the censor strategy

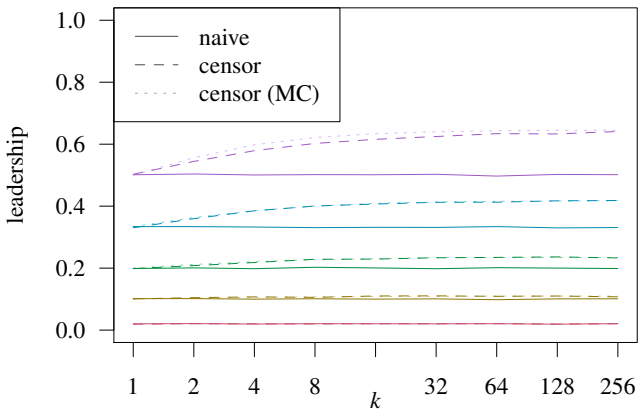


Figure 11: The attacker's share of committed block as a function of quorum size for $\alpha \in \left\{\frac{1}{50}, \frac{1}{10}, \frac{1}{5}, \frac{1}{3}, \frac{1}{2}\right\}$ (bottom-up) in two independent simulations (network and MC).

using an independent Monte Carlo (MC) simulation. (See Appendix B for details.) As depicted in Figure 11, the MC analysis confirms the network simulation.

5.2.3 Chain Quality and Incentive Compatibility The prevalent strategy for increasing the own share of blocks and rewards is selfish mining [22, 50, 59]. This attack is inherently connected with incentives. Its basic idea is to withhold and strategically release blocks in order to create an information asymmetry that allows to reap a disproportional amount of rewards for the invested share of work. This idea is not directly transferrable from Nakamoto consensus to HotPoW for three reasons. First, the finality after three blocks substantially limits the horizon of the selfish miner. Second, block proposals are less valuable. They are not significant sources of reward. Third, block proposals are less critical. In fact, block withholding reduces to the situation of leader failure. Since votes can be reused, honest nodes can replace missing proposals very fast (see Section 5.1.3). This makes proposals less rare events than in Nakamoto consensus, limiting the strategic advantage of withholding them.

However, as we have argued in Section 5.2.2, it is a valid strategy to *withhold votes*. Therefore, we analyze the effect of vote withholding on the distribution of rewards, assuming a constant reward per committed vote, like in Bobtail [9]. The naive strategy yields a share of α of the votes. The attacker's goal is to maximize the number of votes he contributes to each quorum. Since only the leader can decide which votes are included in a proposed quorum, the first step of optimal vote withholding is to increase the odds of becoming the leader. This, in turn, can be achieved by withholding votes! The circularity indicates that the attack can be approximated with the censoring strategy discussed in Section 5.2.2.

Figure 12 shows simulation results on how the strategy, α, and the quorum size affect the share of attacker votes com-

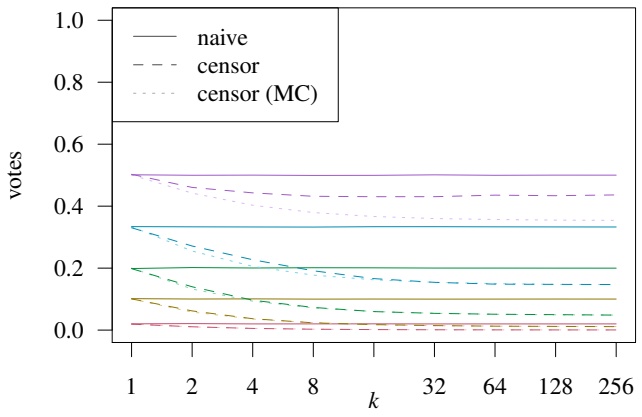


Figure 12: The attacker's share of committed votes as a function of quorum size for $\alpha \in \left\{\frac{1}{50}, \frac{1}{10}, \frac{1}{5}, \frac{1}{3}, \frac{1}{2}\right\}$ (bottom-up) in two independent simulations (network and MC).

mitted to the chain. Interestingly, the censor receives fewer rewards than honest nodes and naive attackers, indicating a dilemma between paying for becoming the leader and capitalizing the power of leadership. The tradeoff is visible by comparing Figures 11 and 12. A similar tradeoff appears for the so-called "proof withholding" strategy in Bobtail [9], which resembles the censoring strategy in HotPoW.

Again, we compare the protocol implementation in the network simulation with the idealized MC model described in Appendix B.

5.3 Overhead

Nakamoto consensus requires one message broadcast per block, namely the block itself, independent of the number of participating nodes. HotPoW adds k message broadcasts per block—one for each vote. Votes are much smaller than blocks. Under the conservative assumptions of 256 bits for block reference and public key, and 64 bits for the puzzle solution, a vote is 72 B.[7]

The number of messages is constant in the number of nodes, like in Bitcoin. However, block headers grow. HotPoW must store the complete quorum with k puzzle solutions. This overhead matters because the header is replicated in all nodes that want to verify the blockchain in the future.

Assuming the same vote size and the most robust case analyzed ($k = 256$), the storage overhead is about 10 kB per block. This is less than 1 % of Bitcoin's average block size in 2019. With this choice of k, falsely accepting a quorum as unique is much less likely than guessing a 128-bit key in one attempt. Table A.1 (in the appendix) shows the storage overhead per block and the associated probability of ambiguity at expected optimistic quorum time (Corollary 2) for different choices of k. We argue that the benefits of the protocol outweigh its storage costs and leave the exploration of compression techniques to future work.

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

Table 1: A comparison of related distributed log protocols.

	PBFT	HotStuff	Proof-of-Stake	Nakamoto	Bitcoin-NG	Byzcoin	Bobtail	HotPoW
# nodes	10	10^2	10^3	10^3	10^3	10^3	10^3	10^3
committee			✓	(✓)	(✓)	✓	✓	✓
permissioned								
- network	✓	✓						
- committee			✓					
resource binding (see Fig. 13)								
- BTP				✓			✓	
- BTI					✓	✓		✓
sidechain					✓	✓		
finality	✓	✓	✓			✓		✓

(✓): Bitcoin and Bitcoin-NG use single-node committees.

6 Discussion

6.1 Relation to Other Distributed Logs

New distributed log protocols are proposed almost every month. We do not claim to know all of them and we do not attempt to provide a complete map of the design space, since other researchers have specialized on this task [7, 13]. Instead, we compare HotPoW to some of its closest relatives along selected dimensions (see Table 1).

6.1.1 Number of Nodes Early BFT protocols were designed for a small number of nodes. PBFT [14], for example, is proven secure under the Byzantine assumptions BFT-1 and BFT-2. It requires multiple rounds of voting to reach consensus on a single value. The communication complexity of $O(n^2)$ renders it impractical for more than a dozen nodes n.

HotStuff [65] ensures safety under the same assumptions, but increases the rate of confirmed values to one per round of voting. Its key idea is to pipeline the commit phases of iterative consensus (recall Fig. 3). Moreover, it reduces communication complexity to $O(n)$ by routing all communication through a leader. These two changes make HotStuff practical for larger networks. However, all correct nodes actively participate (send messages) for each block.

6.1.2 Committee Protocols designed for even larger scale reduce communication complexity further by electing committees. Only committee members participate actively. All

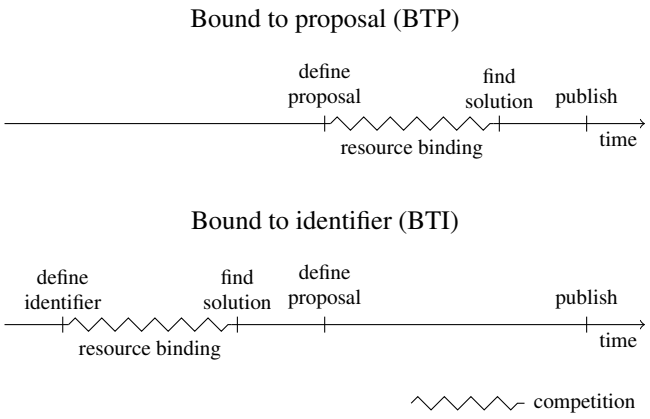


Figure 13: Resources can be bound to concrete proposals or to identifiers, which are later used to sign proposals.

other nodes wait until they become part of a committee.

In Nakamoto consensus, write-access to the ledger is controlled by a proof-of-work puzzle. In each round, one node – the finder of the block – broadcasts a message. Consequently, successful miners can be interpreted as single-node committees. In Bobtail [9] and HotPoW, multiple proof-of-work puzzles are solved per block. Consequently the committee size is greater than one. The committee approach is also followed by proof-of-stake protocols. Here, committee membership is tied to the possession of transferable digital assets (stake).

6.1.3 Permissioned As stated earlier (Sect. 3), assumption BFT-1 can only be satisfied by restricting access to the network based on identities assigned by an external identity provider or gatekeeper. Consequently, protocols relying on this assumption are permissioned on the network layer.

Proof-of-stake internalizes the gatekeeping functionality by restricting access to the committee based on the distribution of stake. While participating as a node is possible without permission, access to the committee is still permissioned.

In proof-of-work systems any agent can join and leave the network and has a (fair) chance of becoming committee member without obtaining permission from a gatekeeper.[8]

6.1.4 Resource Binding Proof-of-work can be seen as a commitment of resources to a value. Typically, these values are chosen locally on each node. Freshness is guaranteed by including a reference to recent puzzle solutions in the value. We distinguish between resources bound to a proposal (BTP) for an upcoming state update and resources bound to an identifier (BTI) used for entering the committee.

Nakamoto consensus uses BTP. Nodes form a proposal for the next block locally and then start to solve a proof-of-work for this proposal. If they are successful in finding a puzzle solution, they share their proposal. This process is depicted in the upper half of Figure 13.

Bitcoin-NG [23] innovated by translating the concept of leader election from the BFT literature (e. g., [20, 28, 51]) to Nakamoto consensus. The miner of a block (elected leader) becomes responsible for appending multiple consecutive (micro) blocks until the next leader emerges with the next mined block. In our framework, Bitcoin-NG adds throughput by switching from BTP to BTI in Nakamoto consensus. A more elaborate BTI protocol is Byzcoin [41]. It forms a committee over the last k successful miners. This rolling committee is then responsible for appending micro blocks. Byzcoin uses PBFT to reach final consensus within each committee, thereby shifting control over the micro blocks from a single node (Bitcoin-NG) to multiple nodes.

HotPoW is a BTI protocol: nodes bind resources to identifiers by mining votes. If they happen to lead when the quorum is complete, they sign a block proposal with their secret key. The lower half of Figure 13 shows this order of events.

Bobtail extends HotPoW by binding a preliminary transaction list into the proof-of-work solution of each vote.[9] This BTP aspect of Bobtail adds significant complexity to the voting logic in order to prevent the reuse of votes for different competing proposals. As described in Section 5, HotPoW makes the reuse of votes a key feature.

6.1.5 Sidechain The sequences of micro blocks in Bitcoin-NG, Byzcoin, and also Thunderella [54] are often referred to as sidechains. Sidechains can serve several purposes, such as increasing throughput (Bitcoin-NG) or adding finality (Byzcoin). However, since different mechanisms are used to advance different chains, synchronization is a major problem. Bitcoin-NG tackles it with incentives, Thunderella focuses on an optimistic case, and Byzcoin leaves open which chain has priority. Sidechains often involve high protocol complexity because different consensus mechanisms are stacked onto each other: the protocols require a distributed log in order to provide a distributed log (with different properties). By contrast, HotPoW provides an improved distributed log directly from a broadcast network and proof-of-work.

6.1.6 Finality The lack of finality in Nakamoto consensus exposes it to many attacks [4, 10, 12, 29]. So far, according to conventional wisdom, eventual consistency has been accepted as the price of a truly permissionless system. Byzcoin challenged this view with a stacked solution involving sidechains. HotPoW achieves the same at lower protocol complexity using proof-of-work quorums. Their stochastic uniqueness allows us to transfer the commit process from the permissioned world to the permissionless.

[8] We ignore the role of the supply chain for puzzle solving equipment.

[9] Since Bobtail inspired HotPoW, a better frame is to see HotPoW as simplification of Bobtail rather than Bobtail as an extension to HotPoW.

6.2 Other Related Protocols

Not included in Table 1 are protocol proposals that replace the linear data structure of the distributed log with more general directed acyclic graphs (DAGs) [61, 62]. This promises higher scalability and faster first confirmation in latent networks at the cost of additional complexity on the application layer, which cannot rely on the total order and uniqueness of state updates anymore. Also Fruitchain [52] can be interpreted as a DAG: it recognizes solutions to hard and easy puzzles but hides the DAG's complexity from the application layer by not allowing 'fruits' to carry state updates.

An even more radical approach is to drop the distributed log completely and implement a digital asset directly on a secure (source-ordered) broadcast without consensus [32]. However, this approach restricts the versatility of the application layer. For example, arbitrary smart contract logic is not supported.

6.3 Limitations and Future Work

We have presented a protocol that achieves finality in a permissionless setting under axiomatic exclusion of the failure modes PoW-1 and PoW-2, and the acceptance of a negligible failure probability. The assumption on PoW-1 and PoW-2 are also made for security proofs of Nakamoto consensus [27, 55]. Nevertheless, it is worth discussing their suitability.

Excluding PoW-1 corresponds to assuming a fixed, network-wide compute power λ. But agents can add and remove nodes at their willing. Even if the number of nodes is fixed, the computational power of each node is not. We observe in practice that a control loop, known as difficulty adjustment (DA), can compensate changes of λ up to a certain degree. But ample literature shows that the deployed DA algorithms are not optimal [25, 36, 42, 47], especially in case of sudden changes of λ. We argue that proof-of-work quorums can support more precise difficulty adjustment algorithms. A higher quorum size implies more votes and hence more data points to inform the algorithm about changes of λ.

The same effect can be exploited for detecting network-level attacks, such as eclipse and splits, more accurately. (Appendix C provides additional details.) This is relevant in the context of the CAP theorem [30], which tells us that every distributed system has to sacrifice one out of consistency, availability and partition tolerance. HotPoW, as presented, favors availability over consistency. It does not implement a mechanism for detecting network splits, even though it is possible at high confidence for big quorum sizes. The trade-off could be changed in favor of consistency. If a split is detected, the protocol withholds commits (and may notify the application layer in order to trigger out-of-band resolutions).

The second failure mode, PoW-2, can be catastrophic and is hard to rule out. We are not aware of any argument that bounds α to a constant below 50 % for any proof-of-work system. In fact, >50 % attacks have been mounted against smaller instances of Nakamoto consensus in practice [21].

Our network simulation in Section 5 models exponentially distributed message propagation times. This distribution puts the system under pressure, but it is not very realistic. Future work might put the simulation on a more structured network topology. However, since the literature reports a significant discrepancy between observed topologies and what cryptocurrencies are designed for [17, 46], it is not obvious what an appropriate topology would look like.

Similarly, we leave unexplored how to disseminate HotPoW's smaller vote messages efficiently. Votes easily fit into single Internet packets and their verification requires only one hash evaluation. It might be possible to improve vote propagation times using UDP-based structured broadcast [58] instead of the gossip broadcast used in many cryptocurrencies.

Finally, we refrain from designing an incentive mechanism for HotPoW for the reasons stated in Section 4.3. A principled approach would be to explore reward-optimizing strategies (combined withholding of votes and blocks) automatically using Markov Decision Processes [59, 66] or even more sophisticated Reinforcement Learning techniques [35].

7 Conclusion

We understand HotPoW as a positive example to support our claim that it is possible to build permissionless distributed logs *with* finality *directly* from proof-of-work. The claim is tentatively supported (with analysis and simulations) until HotPoW is broken. We invite the community to prove our claim wrong, and provide running code online to facilitate this task.[10] It is not safe to use this code in systems dealing with real values.

Regardless of whether our claim is true or false, the identified conflict between inclusiveness and security is instructive, and the associated theory of quorums on stochastic processes may find applications elsewhere. Since it comprises Nakamoto consensus as a special case, it also contributes to a better understanding of the role of proof-of-work in known systems that "work in practice, but [so far] not in theory" [11].

If our claim holds, we have found a way to build permissionless distributed logs from proof-of-work that can serve many applications better than existing systems. However, proof-of-work is a very wasteful way of establishing consensus. It should be avoided whenever possible. Only if there is no alternative to proof-of-work, HotPoW should be considered as a replacement for Nakamoto consensus.

References

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

[10] https://github.com/pkel/hotpow/tree/arxiv_v3

[2] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *Proceedings of SP '17*. IEEE, May 2017.

[3] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, July 2005.

[4] Raphael Auer. Beyond the Doomsday Economics of "Proof-of-Work" in Cryptocurrencies. BIS Working Paper Series 765, January 2019.

[5] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. Technical report, Blockstream, 2014.

[6] Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But Why Does It Work? A Rational Protocol Design Treatment of Bitcoin. In *Proceedings of EUROCRYPT '18*. Springer, 2018.

[7] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. SoK: Consensus in the Age of Blockchains. In *Proceedings of AFT '19*. ACM, October 2019.

[8] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State Machine Replication in the Libra Blockchain. Technical report, Calibra, 2019.

[9] George Bissias and Brian N Levine. Bobtail: Improved Blockchain Security with Low-Variance Mining. In *Proceedings of NDSS '20*. Internet Society, February 2020.

[10] Joseph Bonneau. Why Buy When You Can Rent? In *Proceedings of FC '16 Workshops*. Springer, 2016.

[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Proceedings of SP '15*. IEEE, May 2015.

[12] Eric Budish. The Economic Limits of Bitcoin and the Blockchain. Working Paper 24717, National Bureau of Economic Research, June 2018.

[13] Christian Cachin and Marko Vukolic. Blockchain Consensus Protocols in the Wild (Keynote Talk). In *Proceedings of DISC '17*, 2017.

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

[15] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains. In *Proceedings of FC '16 Workshops*. Springer, 2016.

[16] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *Proceedings of P2P '13*. IEEE, September 2013.

[17] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions. In *Proceedings of FC '19*. Springer, 2019.

[18] Nicola Dimitri. Bitcoin Mining as a Contest. *Ledger*, 2: 31–37, September 2017.

[19] John R. Douceur. The Sybil Attack. In *Proceedings of IPTPS '02*. Springer, 2002.

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

[21] Attah Elikem. 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/, April 2019. Accessed 2020-02-13.

[22] Ittay Eyal and Emin Gün Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *Proceedings of FC '14*. Springer, 2014.

[23] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *Proceedings of NSDI '16*. USENIX, 2016.

[24] James W. Friedman. A Non-cooperative Equilibrium for Supergames. *The Review of Economic Studies*, 38 (1):1–12, 1971.

[25] Daniel Fullmer and A. Stephen Morse. Analysis of Difficulty Control in Bitcoin and Proof-of-Work Blockchains. In *Proceedings of CDC '18*. IEEE, December 2018.

[26] Juan Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational Protocol Design: Cryptography against Incentive-Driven Adversaries. In *Proceedings of FOCS '13*. IEEE, October 2013.

[27] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Proceedings of EUROCRYPT '15*. Springer, 2015.

[28] Hector Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, C-31 (1):48–59, January 1982.

[29] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of CCS '16*. ACM, 2016.

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

[31] Adam Groce, Jonathan Katz, Aishwarya Thiruvengadam, and Vassilis Zikas. Byzantine Agreement with a Rational Adversary. In *Proceedings of ICALP '12*. Springer, 2012.

[32] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The Consensus Number of a Cryptocurrency. In *Proceedings of PODC '19*. ACM, July 2019.

[33] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *Proceedings of Security '15*. USENIX, 2015.

[34] John R. Hicks. *Critical Essays in Monetary Theory*. Clarendon Press, 1967.

[35] Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramer, Giulia Fanti, and Ari Juels. SquirRL: Automating Attack Discovery on Blockchain Incentive Mechanisms with Deep Reinforcement Learning. *arXiv:1912.01798 [cs]*, December 2019.

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

[37] Aljosha Judmayer, Alexei Zamyatin, Nicholas Stifter, Artemios G. Voyiatzis, and Edgar Weippl. Merged Mining: Curse or Cure? In *Proceedings of CBT '17*. Springer, 2017.

[38] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending Fast Payments in Bitcoin. In *Proceedings of CCS '12*. ACM, 2012.

[39] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[40] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain Mining Games. In *Proceedings of EC '16*. ACM, 2016.

[41] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of Security '16*. USENIX, 2016.

[42] Daniel Kraft. Difficulty control for blockchain-based consensus systems. *Peer-to-Peer Networking and Applications*, 9(2):397–413, March 2016.

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

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

[45] Dahlia Malkhi and Michael Reiter. Byzantine Quorum Systems. *Distributed Computing*, 11(4):203–213, October 1998.

[46] Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stutz, and Bernhard Haslhofer. All that Glitters is not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network. *arXiv:2001.09105 [cs]*, January 2020.

[47] Dmitry Meshkov, Alexander Chepurnoy, and Marc Jansen. Short Paper: Revisiting Difficulty Control for Blockchain Systems. In *Proceedings of CBT '17*. Springer, 2017.

[48] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report, 2008.

[49] Arvind Narayanan and Jeremy Clark. Bitcoin's Academic Pedigree. *Communications of the ACM*, 60(12): 36–45, November 2017.

[50] Kevin Alarcon Negy, Peter Rizun, and Emin Gun Sirer. Selfish Mining Re-Examined. In *Proceedings of FC '20*. Preprint, 2020.

[51] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proceedings of ATC '14*. USENIX, 2014.

[52] Rafael Pass and Elaine Shi. FruitChains: A Fair Blockchain. In *Proceedings of PODC '17*. ACM, 2017.

[53] Rafael Pass and Elaine Shi. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *Proceedings of DISC '17*, 2017.

[54] Rafael Pass and Elaine Shi. Thunderella: Blockchains with Optimistic Instant Confirmation. In *Proceedings of EUROCRYPT '18*. Springer, 2018.

[55] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Proceedings of EUROCRYPT '17*. Springer, 2017.

[56] Julien Prat and Benjamin Walter. An Equilibrium Model of the Market for Bitcoin Mining. CESifo Working Paper Series 6865, 2018.

[57] Herbert Robbins. A Remark on Stirling's Formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

[58] Elias Rohrer and Florian Tschorsch. Kadcast: A Structured Approach to Broadcast in Blockchain Networks. In *Proceedings of AFT '19*. ACM, October 2019.

[59] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal Selfish Mining Strategies in Bitcoin. In *Proceedings of FC '16*. Springer, 2016.

[60] Fred B. Schneider. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[61] Yonatan Sompolinsky and Aviv Zohar. Secure High-Rate Transaction Processing in Bitcoin. In *Proceedings of FC '15*. Springer, 2015.

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

[63] William J. Stewart. *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. Princeton University Press, July 2009.

[64] Florian Tschorsch and Björn Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys Tutorials*, 18(3):2084–2123, 2016.

[65] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of PODC '19*. ACM, July 2019.

[66] Ren Zhang and Bart Preneel. Lay Down the Common Metrics: Evaluating Proof-of-Work Consensus Protocols' Security. In *Proceedings of SP '19*. IEEE, May 2019.

A Proofs, Figures, and Visualizations

Lemma 1 The POA for the Poisson process P_λ is given by

$$\mathrm{poa}_{P_\lambda,k}(t) = 1 - e^{-\lambda t} \sum_{i=0}^{2k-1} \frac{(\lambda t)^i}{i!}\,.$$

Proof. P_λ has the following properties [63, p. 389]:

1. $\mathbf{Pr}\big[P_\lambda(0) = 0\big] = 1$,
2. $P_\lambda(t) - P_\lambda(s) \sim \mathrm{Poisson}(\lambda \cdot (t-s))$ for all $s < t$, and
3. for $n \in \mathbb{N}$ and $0 < t_1 < \cdots < t_n$, the family of random variables

$$\{P_\lambda(t_i) - P_\lambda(t_{i-1}) \mid 2 \le i \le n\}$$

is stochastically independent.

According to Definition 4,

$$\begin{aligned}\mathrm{poa}_{P_\lambda,k}(t) =& \mathbf{Pr}\big[P_\lambda(t) \ge 2k\big] && (1)\\ =& 1 - \mathbf{Pr}\big[P_\lambda(t) \le 2k-1\big]\,. && (2)\end{aligned}$$

By setting $s = 0$ in property 2 of the Poisson process and using property 1, we conclude that $P_\lambda(t) \sim \mathrm{Poisson}(\lambda t)$. By evaluating the cumulative distribution function of the Poisson distribution

$$F_{\mathrm{Poisson}}(n;\lambda') = e^{-\lambda'} \sum_{i=0}^{\lfloor n \rfloor} \frac{\lambda'^i}{i!} \qquad (3)$$

for $n = 2k-1$ and $\lambda' = \lambda t$, we obtain the stated result. □

Lemma 2 The optimistic k-quorum time for the Poisson process is Erlang distributed with shape parameter k and rate parameter λ, in short

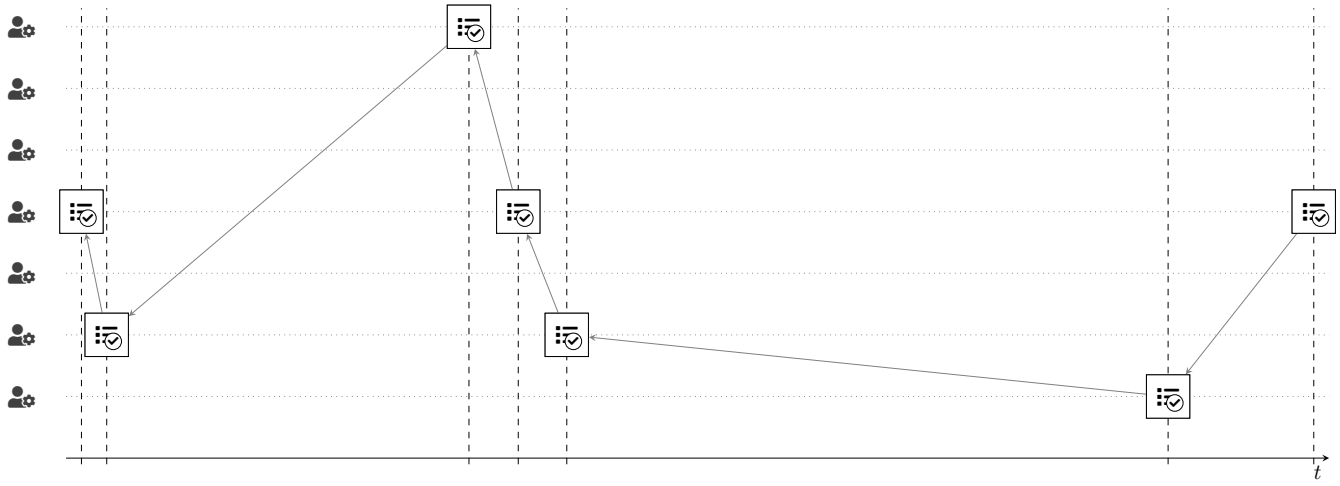
$$T_{P_\lambda,k} \sim \mathrm{Erlang}(k,\lambda)\,.$$

Proof. The time between two consecutive count events of P_λ is exponentially distributed with rate parameter λ. The times between any two consecutive count events are stochastically independent. The sum of k independent and identically distributed exponential random variables is Erlang distributed [63, p. 146] with shape parameter k and rate parameter λ. □

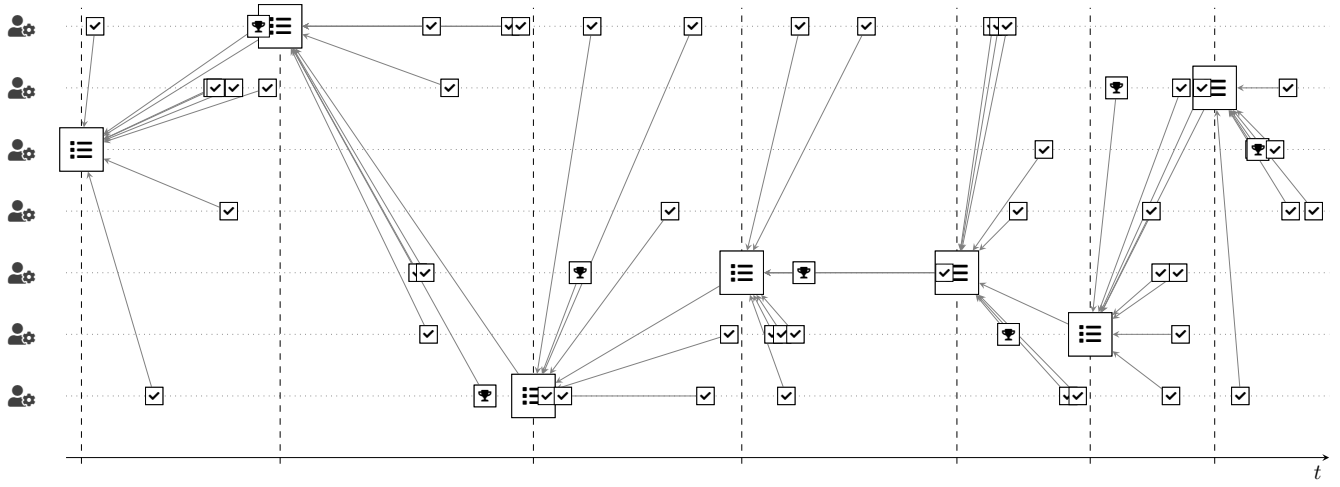
Theorem 1 For the Poisson process, the probability of ambiguity at the expected quorum time is negligible in the quorum size k.

Proof. Let

$$f(k) := \mathrm{poa}_{P_\lambda,k}(\bar{t}_{\lambda,k}) = 1 - e^{-k} \sum_{i=0}^{2k-1} \frac{k^i}{i!}\,. \qquad (4)$$



(a) In Bitcoin, puzzles are solved sequentially. Solutions are bound to block proposals, implying exponentially distributed block intervals.



(b) In HotPoW, smaller puzzles are solved in parallel. One of the solutions is chosen as leader, the corresponding miner collects the quorum ($k = 8$ votes) and proposes the next block. Thereby, HotPoW enables more regular block intervals and more frequent rewards for miners.

(c) Symbols and their meaning.

Figure A.1: Simulated executions of Bitcoin and HotPoW on $n = 7$ nodes (y-axis) over time (x-axis).

Our first observation is that $f(k)$ can be expressed in terms of the regularized incomplete Gamma function $P(\alpha,k)$. According to DLMF §8.4.E9[11],

$$f(k) = P(2k,k)\,. \tag{5}$$

Following the definition of the regularized incomplete Gamma function (see DLMF §8.2.E4), we obtain

$$f(k) = \frac{\gamma(2k,k)}{(2k-1)!}\,, \tag{6}$$

with the incomplete Gamma function (see DLMF §8.2.E1)

$$\gamma(\alpha,k) = \int_0^k t^{\alpha-1}e^{-t}dt\,. \tag{7}$$

We will prove the theorem by providing an (asymptotic) upper bound for $f(k)$ that decreases exponentially in k. Stirling's Approximation [57] provides a useful lower bound for the factorial in the denominator of Equation 6:

$$n! \geq \sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n} \tag{8}$$

We proceed with an upper bound for the enumerator as follows. Let $g(t) = t^{2k-1}e^{-t}$ be the function to integrate for $\alpha = 2k$. Like for integrals in general,

$$\gamma(2k,k) = \int_0^k g(t)\,dt \leq k \cdot \max_{t\in[0,k]} g(t)\,. \tag{9}$$

The derivative of g is $g'(t) = e^{-t}(2k-t-1)t^{2k-2}$. For $t \in [0,k]$ the derivative g' is greater than zero. Hence the function g is monotonically increasing, the maximum is reached at the end of the interval, and

$$\gamma(2k,k) \leq k^{2k}e^{-k}\,. \tag{10}$$

Applying Approximations 8 and 10 to Equation 6, yields

$$f(k) \leq \frac{k^{2k}e^{-k}}{\sqrt{2\pi}(2k-1)^{2k-\frac{1}{2}}e^{-2k+1}} \tag{11}$$

$$= \left(\frac{k\sqrt{e}}{2k-1}\right)^{2k}\sqrt{\frac{2k-1}{2\pi e^2}} \tag{12}$$

Observe that

$$\limsup_{k\to\infty} \frac{\left(\frac{k\sqrt{e}}{2k-1}\right)^{2k}}{\left(\frac{\sqrt{e}}{2}\right)^{2k}} = \limsup_{k\to\infty}\left(\frac{2k}{2k-1}\right)^{2k} = e < \infty\,. \tag{13}$$

Thus,

$$f(k) = O\left(\frac{e^k}{4^k}\sqrt{k}\right)\,. \tag{14}$$

[11] NIST Digital Library of Mathematical Functions

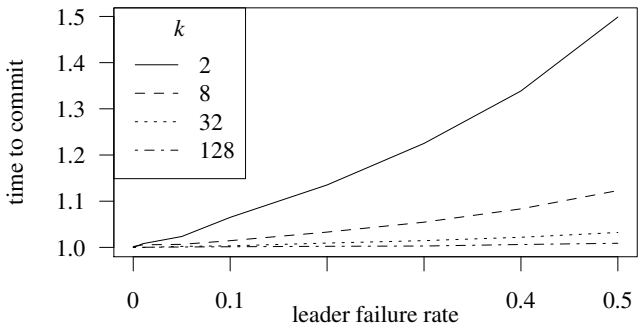


Figure A.2: Pure effect of leader failure, i. e., *without* latency. Supplement to Figure 10.

Table A.1: Storage overhead of HotPoW consensus.

quorum size k	probability of ambiguity at expected quorum time	block header (bytes)
1	0.2642	72
2	0.1429	112
16	0.0003	672
64	1.2×10^{-12}	2.6 k
256	4×10^{-45}	10 k

Since $\sqrt{k} < 1.25^k$ for $k > 1$, we can conclude

$$f(k) = O\left(\frac{e^k}{4^k}1.25^k\right) \tag{15}$$

$$= O\left(0.85^k\right)\,. \tag{16}$$

□

B Monte Carlo Simulation

We cross-check the implementation of the censor strategy and its behavior in the network simulation (see Sect. 5.2.2) using an independent Monte Carlo simulation. We model the formation of individual quorums using an (Absorbing) Markov Chain, but omit higher-level concepts such as blocks and their chaining. The censor strategy is to generally withhold votes until either the attacker can form a quorum as leader, or the defender forms a quorum without any of the attacker's (withheld) votes. In a protocol execution, the first case (SUCCESS) applies when the attacker proposes a block which the honest nodes accept. The second case (FAIL) applies when the honest nodes propose a block.

State representation and initialization We model the current state as a triple (a,d,l), where $a \in \mathbb{N}$ denotes the number of (withheld) attacker votes, $d \in \mathbb{N}$ (for defender) denotes the

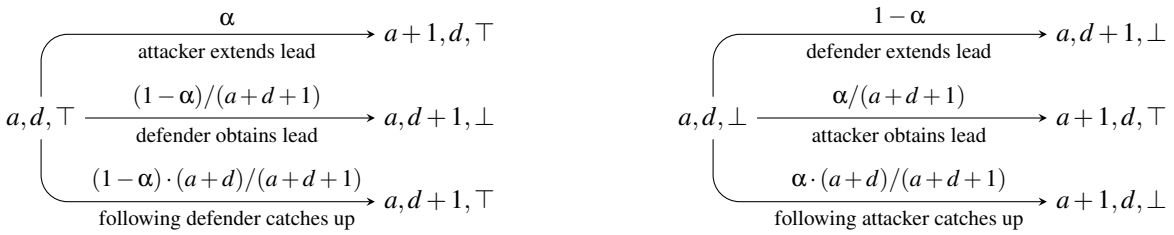


Figure B.1: Probabilistic state transitions in the Markov Chain model for the *censor* strategy.

number of votes of the honest nodes, and $l \in \{\bot, \top\}$ is true if the attacker holds the currently smallest vote. The initial state is $(1,0,\top)$ with probability α and $(0,1,\bot)$ otherwise.

State transition Figure B.1 shows an annotated state transition diagram. If $l = \top$, the next state is

$$\begin{aligned}
&(a+1,d,l) && \text{with probability } \alpha,\\
&(a,d+1,\bot) && \text{with probability } \frac{1-\alpha}{a+d+1}, \text{ and}\\
&(a,d+1,l) && \text{otherwise.}
\end{aligned}$$

If $l = \bot$, the next state is

$$\begin{aligned}
&(a,d+1,l) && \text{with probability } 1-\alpha,\\
&(a+1,d,\top) && \text{with probability } \frac{\alpha}{a+d+1}, \text{ and}\\
&(a+1,d,l) && \text{otherwise.}
\end{aligned}$$

Termination If $l \wedge a+d \geq k$, the simulation terminates in SUCCESS. If $\neg l \wedge d \geq k$, it terminates in FAIL. The simulation continues until one of these conditions is true.

Simulation We run the model 1 000 000 times for each combinations of $\alpha \in \left\{\frac{1}{50}, \frac{1}{10}, \frac{1}{5}, \frac{1}{3}, \frac{1}{2}\right\}$ and $k \in \{1,2,4,\ldots,256\}$. Figure 11 shows the fraction of cases where the simulation terminates in SUCCESS. Figure 12 shows the average number of attacker votes for the runs that end in SUCCESS.

Table C.1: Time until eclipse can be detected at confidence $p = 0.001$ (relative to the expected block time).

quorum size	1	2	4	8	16	32	64	128	256
time	6.91	3.45	1.73	0.86	0.43	0.22	0.11	0.05	0.03

C Detecting Attacks

Each vote is linked to one ATV. By assumption (Sect. 3), the time between two consecutive ATVs is exponentially distributed with rate λ. In an honest network, a node regularly receives votes (and own ATVs). A node can test the hypothesis of being eclipsed based on the arrival of votes. Table C.1 shows after how much time (relative to the block time) of not receiving a single vote a node can rule out a natural course of events with confidence $p = 0.001$.

Observe that larger quorums sizes increase the detectability of eclipse attacks. For quorum sizes greater than 8, eclipse attacks can be detected with confidence within a single expected block time. For plain Nakamoto consensus ($k = 1$), an equally powerful test requires an observation window of almost 7 times the expected block time.