Tailstorm: A Secure and Fair Blockchain for Cash Transactions

Patrik Keller ✉
Universität Innsbruck, Austria

Ben Glickenhaus
University of Massachusetts Amherst, MA, USA

George Bissias
University of Massachusetts Amherst, MA, USA

Gregory Griffith
Bitcoin Unlimited

───── **Abstract** ─────

Proof-of-work (PoW) cryptocurrencies rely on a balance of security and fairness in order to maintain a sustainable ecosystem of miners and users. Users demand fast and consistent transaction confirmation, and in exchange drive the adoption and valuation of the cryptocurrency. Miners provide the confirmations, however, they primarily seek rewards. In unfair systems, miners can amplify their rewards by consolidating mining power. Centralization however, undermines the security guarantees of the system and might discourage users.

In this paper we present Tailstorm, a cryptocurrency that strikes this balance. Tailstorm merges multiple recent protocol improvements addressing security, confirmation latency, and throughput with a novel incentive mechanism improving fairness. We implement a parallel proof-of-work consensus mechanism with k PoWs per block to obtain state-of-the-art consistency guarantees [29]. Inspired by Bobtail [9] and Storm [4], we structure the individual PoWs in a tree which, by including a list of transactions with each PoW, reduces confirmation latency and improves throughput. Our proposed incentive mechanism discounts rewards based on the depth of this tree. Thereby, it effectively punishes information withholding, the core attack strategy used to reap an unfair share of rewards.

We back our claims with a comprehensive analysis. We present a generic system model which allows us to specify Bitcoin, $\mathcal{B}_k$ [29], and Tailstorm from a joint set of assumptions. We provide an analytical bound for the fairness of Tailstorm and Bitcoin in *honest* networks and we confirm the results through simulation. We evaluate the effectiveness of *dishonest* behaviour through reinforcement learning. Our attack search reproduces known optimal strategies against Bitcoin, uncovers new ones against $\mathcal{B}_k$, and confirms that Tailstorm's reward discounting makes it more resilient to incentive layer attacks. Our results are reproducible with the material provided online [30].

Lastly, we have implemented a prototype of the Tailstorm cryptocurrency as a fork of Bitcoin Cash. The client software is ready for testnet deployment and we also publish its source online [23].

1   Introduction

Proof-of-work (PoW) cryptocurrencies can be thought of as stacked systems comprising four layers. The PoW layer moderates access of weakly identified parties using a mining puzzle: proposing a new block requires finding a small hash. The consensus layer allows all participants to agree on a specific block ordering. The application layer maintains a distributed ledger by writing cryptocurrency transactions into the blocks. Finally, the incentive layer motivates participation in the PoW layer by minting new cryptocurrency for successful miners.

The circular dependencies between the layers result in interdependent failures. Consensus faults are inevitable when individual miners gain too much control [19]. Unreliable consensus enables double spending and erodes confidence in the cryptocurrency. Devaluation of the currency renders the mining rewards worthless as well. Lastly, misaligned incentives encourage centralization among the miners, eventually allowing the strongest one to break consensus.

In this paper, we introduce and analyze Tailstorm, a new cryptocurrency that strengthens two layers of the stack. We employ an innovative incentive mechanism as well as a state-of-the-art consensus mechanism, while retaining the PoW and application layers of Bitcoin. We draw inspiration from the organization of delta blocks in the Storm protocol [4] as well as the use of partial PoW in the Bobtail [9] and $\mathcal{B}_k$ protocols [29].

Bitcoin's consensus [39] uses a *sequential* PoW mechanism where each block references a single parent block. The blocks form a tree and the participants mine new blocks that extend the longest branch according to the *longest chain rule*. Blocks off the longest branch are discarded. Evidently, attackers who possess more than 50 % of the hash rate pose a threat to the system: they can execute double-spend attacks by mining their own branch until it eventually becomes the longest. But even less mining power might suffice since honest participants also discard the blocks of other honest miners if they are not on the same branch. This happens naturally in realistic networks due to propagation delays. If an attacker can induce and exploit communication delays, all discarded blocks may benefit the attacker; effectively increasing their strength [24].

In contrast, Tailstorm implements a *parallel* PoW consensus mechanism that largely avoids discarding blocks. For this, we closely follow the approach taken by Keller and Böhme at AFT '22 [29]. Their protocol, $\mathcal{B}_k$, confirms each block with *k votes*. $\mathcal{B}_k$ blocks do not require a PoW, only votes do. Notably, votes confirming the same block can be mined in parallel because they do not depend on each other. Discarding only occurs when there are more than k votes for the same block. Even then, individual discarded votes only account for $1/k$ of a block's PoW. As Keller and Böhme [29] argue, this makes consensus more robust against consensus layer attacks.

But it is futile to analyze the consensus without considering incentives. Ideally, rewards are distributed fairly, which means that a miner's expected reward is proportional to its hash rate. Arnosti and Weinberg [3] show that even small inequalities in reward allocation encourage substantial centralization of hash rate which ultimately poses a threat to consensus and the cryptocurrency itself [19].

Unfairness arises from natural network delays and dishonest behavior. Both of these factors affect Bitcoin. In latent networks, two blocks mined around the same time may refer to the same parent, even if both miners follow the longest chain rule. One of the blocks will be discarded, the other rewarded. The stronger miner has more hash rate to support their own block, hence the weaker miner is worse off. Network-level attackers can additionally exploit latency to manipulate impartial participants in their favor. But even without delays,
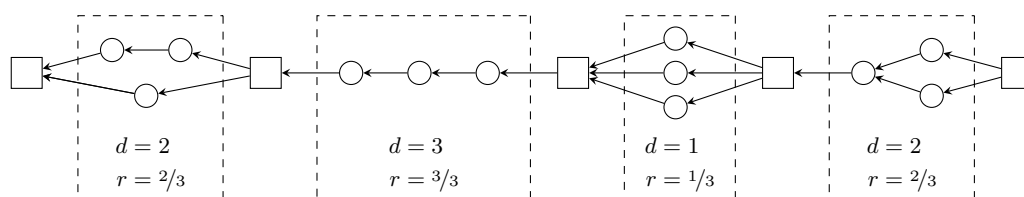
Figure 1 Example of a Tailstorm blockchain with $k = 3$ subblocks per summary. Squares are summary blocks, circles are subblocks, arrows indicate hash-references. Dashed rectangles mark subblock trees with depth d and discounted subblock reward r. Reduced depth implies lower rewards.

Bitcoin miners with more than one third of the hash rate can reap an unfair share of rewards by temporally withholding blocks instead of acting honestly [16, 21, 40, 47]. As we will demonstrate, $\mathcal{B}_k$ suffers from the same problem, partly due to its leader election mechanism.

Tailstorm addresses the unfairness problem through an innovative reward scheme that punishes withholding. The Tailstorm blockchain consists of *subblocks* and *summary* blocks. Similar to $\mathcal{B}_k$, summaries do not require a PoW, but subblocks do. Assembling a new summary requires k subblocks that confirm the same parent summary. To preserve the security properties of $\mathcal{B}_k$, subblocks confirming the same summary are conflict-free, and hence can be mined in parallel. Taking inspiration from Bobtail [9], subblocks *optionally* refer to another subblock instead of a summary, and hence form a tree. With Tailstorm, we propose to discount rewards based on the depth of this tree, as depicted in Figure 1. Mining subblocks in private causes branching of the tree, reduces its depth, and ultimately leads to lower rewards.

We support our claims with a comprehensive analysis and make the following contributions.

1. We formulate a generic system model for PoW cryptocurrencies. The models abstracts PoW and communication, defines a joint set of assumptions, and enables valid comparisons between different consensus protocols and incentive mechanisms.

2. We specify Bitcoin, $\mathcal{B}_k$, and Tailstorm in the joint model. To isolate the effect of Tailstorm's discount reward scheme and $\mathcal{B}_k$'s leader election mechanism, we additionally specify a hybrid protocol, TS/const, modelling Tailstorm without discounting and $\mathcal{B}_k$ without leader election.

3. We provide an upper bound for the orphan rate of Tailstorm in honest networks. Compared to Bitcoin [43], Tailstorm creates less orphans and hence presents less opportunity for unfairness.

4. We implement the system model as a simulator and show that Tailstorm is more fair than Bitcoin in honest but realistic networks with propagation delays. We confirm Bitcoin's inherent tradeoff: while short block intervals are desirable for fast confirmations and a less volatile stream of rewards, they also bias rewards in favour of strong miners. In Tailstorm, these concerns are largely separated by configuring long *summary* block intervals for fairness and short *subblock* intervals for fast confirmations and frequent rewards.

5. We evaluate multiple hard-coded attack strategies against the specified protocols, finding that attacks which are profitable against $\mathcal{B}_k$ are less profitable against Tailstorm, with the TS/const protocol lying in between.

6. We follow Hou et al. [26] and search optimal attack strategies using reinforcement learning. Our search reproduces optimal strategies against Bitcoin [21, 47] and generally matches or outperforms the hard-coded strategies. The regularity of our results indicates that we indeed found near-optimal strategies against all protocols and enables the conclusion that Tailstorm is less susceptible to incentive layer attacks than the other protocols.

7. We describe the Tailstorm application layer, which implements a cryptocurrency on top of the proposed consensus protocol. It preserves Bitcoin's transaction logic while enabling faster confirmations. Transactions are stored in subblocks in much the same way that the Storm protocol [4] stores transactions in delta blocks.

8. Lastly, we implement a prototype of Tailstorm which is ready for testnet deployments and make the code available online [23].

We structure the paper in order of our contributions. Section 2 defines the system model and Section 3 presents the specification of Tailstorm. In Section 4, we evaluate the protocols in an honest network with propagation delays. In Section 5, we evaluate hard-coded attack strategies, and in Section 6 we conduct the search for optimal policies with reinforcement learning. Section 7 presents the Tailstorm cryptocurrency and our prototype implementation. In Section 8, we discuss related work, limitations and future work. Section 9 concludes.

2   System Model

Recall the layered view on PoW cryptocurrencies presented in the introduction: The PoW layer moderates access using a mining puzzle, the consensus layer establishes a specific block ordering, the application layer writes cryptocurrency transactions into the blocks, and the incentive layer mints new cryptocurrency for successful miners. We now present a system model that abstracts PoW and communication to enable concise specification of the consensus layer. Application and incentives are considered in later sections.

In practice, PoW consensus protocols are executed as distributed systems, where independent nodes communicate over a P2P network. Messages exchanged between the nodes may be subject to natural or potentially malicious delays. To facilitate specification, we abstract this distributed system and model it algorithmically. We define a virtual environment that emulates distributed protocol execution in a single thread of computation. Within the virtual environment, nodes are represented as numbers, and blocks are represented as vertices in a directed acyclic graph (DAG). Mining is simulated as a loop with random delays, while communication is modeled by restricting the visibility of blocks to a subset of the nodes. The behaviour of nodes is defined by functions, which can be customized to model different protocols.

We define the virtual environment in Algorithm 1. The environment maintains a DAG where each *vertex* represents one block. Each block b has an associated list of parent blocks that constitute the outgoing *edges* in the DAG. We denote this list `parents`(b). In practice, edges arise from hash references pointing to other blocks in the blockchain. We say that block b is a *descendant* of block b', if b' is either a parent of b, or is connected transitively by the `parents` relationship. In this case, we say b' is an *ancestor* of b. Each block has *properties* which are assigned as the protocol unfolds. For example, the virtual environment uses the Boolean property `pow`(b) to track whether block b has a PoW or not.

We label the participating nodes with integers ranging from 1 to n. For each node, the virtual environment maintains a local view of the DAG and a preferred tip of the chain. In Lines 10 and 21, we restrict the local view of node i to blocks where `visibility`(b, i) was set to true. Initially, local views are empty and new blocks are not visible to any node. We denote as `tip`(i) the preferred *tip* of node i, the block to which a new block from i will point. We describe the behaviour of nodes as pure functions. These functions are called by the environment to obtain instructions from the node which the environment then interprets according to our assumptions. This makes all modifications of the DAG and all

■ **Algorithm 1** Virtual Environment.

```
 1  t ← Root();                                              // Root: protocol's genesis block template
 2  b ← block obtained from appending template t to the block DAG;                    // reification
 3  for i = 1, . . . , n do                                                    // n: number of nodes
 4    │  tip(i) ← b;                                                       // tip(i): state of node i
 5    │  visibility(b, i) ← true;                      // visibility(b, i): whether node i sees block b

 6  while true do                                                           // concurrent PoW loop
 7    │  sample delay d_pow ∼ Expon(rate λ);                                       // λ: PoW rate
 8    │  sample node i ∼ Discrete(1, . . . , n with weights κ_1, . . . , κ_n); // κ_1, . . . κ_n: relative hash rates
 9    │  wait for d_pow seconds, handling other tasks concurrently;
10    │  with local view of node i do                    // partial and immutable view on block DAG
11    │    │  tmpl ← Extend(tip(i));                                // Extend: protocol's mining rule
12    │  b ← block obtained from appending template tmpl to the block DAG;          // reification
13    │  pow(b') ← true;                                                      // mark block as mined
14    │  visibility(b, ·) ← false;                                      // block not yet propagated
15    │  if Validate(b) then Deliver(b, i);                      // Validate: protocol's chain structure

16  function Deliver(block b, node i) is                          // Turn block visible for node . . .
17    │  for p ∈ parents(b) do                                         // . . . in topological order . . .
18    │    │  wait until visibility(p, i) is true;
19    │  if not visibility(b, i) then                                      // . . . and at most once.
20    │    │  visibility(b, i) ← true;
21    │    │  with local view of node i do               // partial and immutable view on block DAG
22    │    │    │  (tip(i), share, append) ← Update(tip(i), b); // Update: protocol's state update rule

23    │    │  for b' ∈ share and j = 1, . . . , n, j ≠ i do          // handle block broadcast requests
24    │    │    │  pick message delay d_net according to network assumptions;
25    │    │    │  Deliver(b', i) in d_net seconds;

26    │    │  for tmpl ∈ append do                    // handle requests to append blocks without PoW
27    │    │    │  b' ← block obtained from appending template tmpl to the block DAG; // reification
28    │    │    │  pow(b') ← false;                                          // block has no PoW
29    │    │    │  visibility(b', ·) ← false;
30    │    │    │  if Validate(b') then Deliver(b', i);        // Validate: protocol's chain structure
```

communication explicit in Algorithm 1. In particular, nodes do not directly append blocks to the DAG; they return block templates, which the environment then reifies by appending a new block to the DAG.

A protocol is fully specified through four functions: Root and Validate define the structure of the blockchain, while Update and Extend define the behavior of honest nodes.

The Root function takes no argument and returns a single block, which we call *genesis*. Initially, the genesis is the only block in the DAG. Validate takes a block as argument and returns true if the block is valid and false otherwise. E. g., Bitcoin's Validate checks the pow property and that there is exactly one parent. The environment enforces block validity during the reification of blocks, while deployed protocols would reject invalid blocks in the communication layer. The genesis is not subject to the validity rule.

The Update function specifies how nodes react to newly visible blocks, after they are mined locally with PoW, appended locally without PoW, or received from the network. The function takes two arguments: the node's currently preferred tip and the new block. The function returns the new preferred tip, a list of blocks the node intends to *share* with other nodes, and a list of block templates it wants to *append* to the chain *without* PoW. On the other hand, the function Extend defines how nodes grow the chain *with* PoW. It takes a single argument, a node's currently preferred tip, and returns a template for the block that the node intends to mine.

■ **Algorithm 2** Tailstorm: Chain Structure.

```
 1  function Root() is
 2      return block template b with summary(b) = true and height(b) = depth(b) = 0;

 3  function Validate(block b) is
 4      if summary(b) then
 5          p ← last summary before b;
 6          S ← subblocks between b and p;
 7          return (|S| = k) ∧ (depth(b) = 0) ∧ (height(b) = height(p) + 1);
 8      else                                                              // b is subblock
 9          p ← first parent of b;
10          return
              pow(b) ∧ (|parents(b)| = 1) ∧ (depth(b) = depth(p) + 1) ∧ (height(b) = height(p));
```

We follow related work [51, 43, 15, 37, 29] and model the mining process in continuous time. The virtual environment generates independent mining delays from the exponential distribution $\texttt{Expon}(\lambda)$, with rate λ measured in expected number of proofs-of-work per second. Accordingly, the expected value of the distribution, $1/\lambda$, is called the *mining interval* and is measured in seconds. After each mining delay, the environment randomly selects a successful miner, obtains a block template from $\texttt{Extend}$, and reifies the block by appending it to the DAG. We support arbitrary hash rate distributions among the nodes by setting the weights $\kappa_1, \ldots, \kappa_n$ in Line 8 accordingly.

The $\texttt{Deliver}$ function captures the process of making blocks visible to nodes. The specified protocols have in common that block validation requires knowledge of all referenced blocks. We avoid a lot of boilerplate code in the specification, by ensuring that parent blocks are delivered before their children. Upon delivery, the virtual environment first invokes the $\texttt{Update}$ function to obtain the node's new preferred tip, a list of blocks the node wants to share, and a list of block templates the node intends to append without PoW. It then handles the node's requests to share and append. Communication is modelled through delayed delivery, while appends happen immediately.

3   The Tailstorm Protocol

This section specifies the Tailstorm consensus protocol and reward mechanism using the algorithmic model described in Section 2. The specification serves as the basis for our theoretical analyses, network simulations, and attack search in subsequent sections. We first describe Tailstorm's chain structure in Section 3.1. We then specify the behaviour of honest nodes in Section 3.2. As a point of reference, we also specify Bitcoin and $\mathcal{B}_k$ protocols in the extended version of the paper. In this section, we assume that the application layer implements a cryptocurrency which we can use to pay rewards. We defer the description of Tailstorm's application layer to Section 7. Throughout this section, we focus on honest miners who follow the protocol as intended. Later sections will consider dishonest behaviour.

3.1   Chain Structure

Algorithm 2 defines Tailstorm's chain structure. Each block is either a summary or a subblock. Subblocks must have PoW and they must have exactly one parent. Summaries do not require PoW and reference k subblocks, each confirming the same ancestor summary.

Each block b has two integer properties, $\texttt{height}(b)$ and $\texttt{depth}(b)$. The genesis has height and depth zero. Subblocks inherit the height of their parent and they increment the depth by one. Summaries increment the height and reset the depth to zero. Figure 1 in Section 1

◼ **Algorithm 3** Tailstorm: Node.

---

 1 **function** Preference(summary s, summary b) **is**
 2   hs, hb ← height(s), height(b);
 3   ns, nb ← size of subblock tree confirming s and b;
 4   rs, rb ← reward obtained from (s, b);
 5   **return** $(hb > hs) \lor (hb = hs \land nb > ns) \lor (hb = hs \land nb = ns \land rb > rs)$;

 6 **function** Extend(tip t) **is**   // t: currently preferred block
 7   **if** t has children **then** p ← maximum-depth subblock after t **else** p ← t;
 8   **return** subblock template b with
 9   parents(b) = $[p]$, height(b) = height(p), and depth(b) = depth(p) + 1;

10 **function** Update(tip t, block b) **is**   // t: currently preferred block // b: new block
11   *pref, share, append* ← t, $[b]$, $[]$; // new preferred tip, blocks to broadcast, templates to append
12   **if** summary(b) **then**
13   **if** Preference(*pref*, b) **then** *pref* ← b;
14   **else**   // b is subblock
15   p ← last summary before b;
16   **if** Preference(*pref*, p) **then** *pref* ← p;
17   S ← all subblocks in subblock tree confirming p;
18   **if** $|S| \geq k$ **then**
19   L ← result of Algorithm 4: Subblock Selection;
20   create summary template s with parents(s) = L;
21   *append* ← $[s]$;

22   **return** *pref, share, append*;

---

illustrates a valid Tailstorm chain for $k = 3$. Note that the subblocks confirming the same summary form a tree and that the depth property tracks the depth of this tree. The height property on the other hand counts the number of trees that have been summarized.

To incentivize participation, Tailstorm allocates rewards to the miners of subblocks. The reward size is proportional to the depth of the subblock tree: let b be a summary block, and S be the set of subblocks in the corresponding subblock tree. Then all subblocks in S are allocated the same reward

$$\mathtt{discount}(b) = \frac{c}{k} \cdot \max_{x \in S} (\mathtt{depth}(x)), \tag{1}$$

where c represents a tunable upper limit on the subblock reward. In Figure 1 we show the rewards for $c = 1$. Note that the reward scheme punishes non-linearities in the blockchain, and this punishment affects all included subblocks equally.

3.2 Honest Nodes

Algorithm 3 specifies the behaviour of honest nodes. The algorithm revolves around a preference order (ln. 1-5) that ranks summaries first by height, then by number of confirming subblocks, and finally by potential personal reward for the individual node. Nodes set the highest ranked summary as their preferred summary (ln. 13+16) and they mine subblocks (ln. 6-9) that confirm their preferred summary. To maximize the depth of the subblock tree, nodes append their subblocks to the longest existing branch.

Whenever nodes learn about a new block (ln. 10-22), they share it with the other nodes and they update their preference. As soon as there are k subblocks (ln. 18) confirming the preferred summary, nodes assemble the next summary. When there are more than k subblock candidates for the next summary, nodes choose the subblocks to maximize their own rewards. We present a greedy algorithm for subblock selection in Algorithm 4. Note that nodes may

◼ **Algorithm 4** Tailstorm: Subblock Selection.

1 $R \leftarrow \emptyset$;	// selected subblocks
2 **while** $|R| < k$ **do**	// select one subblock per iteration
3 $\quad C \leftarrow S \setminus R$;	// S: candidate subblocks
4 $\quad$ **for** $x \in C$ **do**	
5 $\qquad B_x \leftarrow x$ and all ancestors of x in S;	
6 $\qquad B'_x \leftarrow B_x \setminus R$;	// newly referenced blocks
7 $\qquad r_x \leftarrow$ number of node's own subblocks in B'_x;	// reward
8 $\quad C \leftarrow \{x \in C : |R| + |B'_x| \leq k\}$;	// enforce tree size k
9 $\quad y \leftarrow \arg\max_{x \in C} r_x$;	// select candidate with highest reward
10 $\quad R \leftarrow R \cup B'_y$;	// track referenced subblocks
11 **return** leaves in subblock tree R;	// invariant: $|R| = k$

choose different subblocks and thereby create conflicting summaries. Such conflicts create temporary forks of the blockchain which are resolved quickly according to the preference order described above. We analyze the implications on fairness in Section 4.

3.3 Difficulty Adjustment

A major goal for most blockchains, including Tailstorm, is for the blockchain itself to grow at a constant rate so as to maintain constant transactional throughput. However, in any deployed blockchain, the puzzle solving rate λ changes over time because nodes may come and go or may add or remove mining hardware. The changes in solving rate lead to changes of the growth rate. Adjusting for the fluctuations requires feedback from the consensus to the PoW layer. Typically, blockchains adjust the puzzle solving *difficulty* depending on the observed chain growth using a dynamic *difficulty adjustment algorithm* (DAA).

There exists a rich body of prior work concerning DAA design and analysis [8, 18, 25, 27, 35], but a deep investigation of ideal DAAs for the Tailstorm protocol is beyond the scope this paper. We note, however, that existing DAAs for Bitcoin can be adapted to Tailstorm by counting the number of subblocks where Bitcoin DAAs use the length of the blockchain. For any Tailstorm block b (summary or subblock), we define

$$\texttt{progress}(b) = k \cdot \texttt{height}(b) + \texttt{depth}(b), \tag{2}$$

which counts the number of PoWs included in the chain. Any Tailstorm DAA should adjust the puzzle solving difficulty such that $\texttt{progress}$ grows at a constant rate.

3.4 Protocol Variant With Constant Rewards

Tailstorm discounts rewards proportional to the depth of the subblock tree. We see the discounting mechanism as a core contribution of this paper. To isolate the effect of discounting, we introduce a protocol variant without discounting, which we call TS/const. While Tailstorm pays out *at most c* units of reward per subblock, the TS/const protocol pays out *exactly c* units of reward per subblock. In all other aspects, TS/const is identical to Tailstorm.

Note, that the TS/const protocol does not use the subblock tree structure, neither for consensus nor for incentives. In that regard, TS/const resembles the parallel PoW protocol $\mathcal{B}_k$ [29] where all subblocks refer to a summary, never another subblock. The only difference with $\mathcal{B}_k$ is that TS/const does not implement leader election: while $\mathcal{B}_k$ restricts the creation of the next summary to the miner of the subblock with the smallest hash, any TS/const node may (re-)create valid summaries locally.

4   Fairness Under Protocol Compliance

A fair PoW protocol rewards miners in proportion to the amount of work they do. Dispro-
portionate or inconsistent allocation discriminates against weak miners and encourages the
formation of pools and centralization. In this section, we explore the causes of unfairness in
Tailstorm under the assumption that all miners are honest. We measure work in number of
hashes evaluated. The system is fair if rewards are proportional to the miners' hash rates.

The root cause of unfairness is the inherent asymmetry in reward loss when multiple PoW
solutions are discovered in short order. In Tailstorm, if more than k subblocks are produced,
all having the same height, then all but k of them will be discarded. The discarded blocks
are commonly called *orphans* and receive no rewards. Typically, miners do not orphan their
own blocks. Since miners with a high hash rate are more often able to choose which blocks
will be orphaned, miners with a relatively low hash rate lose a disproportionate amount of
rewards. In Bitcoin, this effect is amplified because orphaning can occur for each block (set
$k = 1$ in the above argument).

4.1   Analytical Orphan Rate Analysis

In this section, we develop an analytical model for the orphan rate and use it to compare
the fairness of Bitcoin and Tailstorm. Let B be a random variable denoting the number of
subblocks orphaned during the production of a summary block in Tailstorm or the number
of orphans per block in Bitcoin. The orphan rate is given by $\rho = E[B]/k$, which represents
the expected number of PoW solutions orphaned for every PoW solution confirmed. In the
remainder of this section, we will derive a bound on ρ.

Following Rizun's orphan rate analysis for Bitcoin [43], we model block propagation
delays as

$$\tau(\tau_0, z, Q) = \tau_0 + zQ, \tag{3}$$

where τ_0 represents network latency (seconds), z represents bandwidth (bytes per second),
and Q represents block size (bytes). In order to adapt this expression for Tailstorm, we
assume that the transactions included in a summary block are spread evenly across the k
subblocks, so the subblock size is Q/k. The expected summary block interval is T and the
subblock mining rate is $\lambda = k/T$.

▶ **Theorem 1.** *For network parameters τ_0, z, and Q, summary block interval T, and subblock
count k, Tailstorm's expected orphan rate ρ is bounded from above by:*

$$\rho(\tau_0, z, Q, T) \leq \frac{\tau_0}{T} + \frac{zQ}{kT}.$$

Proof. Assume there are $k + X$ subblocks, all pointing to the same summary block. Then, k
of these subblocks will be included in the next summary and X will be orphaned. Recall
from Algorithm 3 that honest miners proceed to the next summary as soon as they learn
of k subblocks. So, if X subblocks are orphaned, they must have been mined before the
kth subblock has propagated to all nodes. Additionally, subblocks k through $k + X$ must
originate from different miners, since no miner orphans its own subblock.

Let Y_t be a random variable representing the number of subblocks generated during an
arbitrary time interval $t > 0$ under the assumption, introduced in Section 2, that mining
intervals are exponentially distributed with rate $\lambda = k/T$. A standard result from the theory
of Poisson processes shows that $Y_T \sim \texttt{Poisson}(k)$ [46, ch. 5]. The same theory shows that

■ **Table 1** Upper bounds on orphan rate obtained from Theorem 1 assuming various combinations of k and expected block (Bitcoin) or summary block (Tailstorm) intervals T.

	Bitcoin	Tailstorm		
block interval	$k = 1$	$k = 5$	$k = 10$	$k = 15$
$T = 75$ seconds	10.08 %	7.35 %	7.01 %	6.89 %
$T = 150$ seconds	5.04 %	3.67 %	3.50 %	3.45 %
$T = 300$ seconds	2.52 %	1.84 %	1.75 %	1.72 %
$T = 600$ seconds	1.26 %	0.92 %	0.88 %	0.86 %

$Y_t \sim \texttt{Poisson}(t\frac{k}{T})$ for arbitrary $t > 0$. According to our network model, the kth subblock (as well as the $k - 1$ before) is fully propagated after $\tau(\tau_0, z, Q/k)$ seconds. Thus, X is equal to $Y_{\tau(\tau_0,z,Q/k)}$ in distribution. $E[X]$ bounds the expected number of subblocks orphaned per summary block. The orphan rate, that is subblocks orphaned per subblocks included, is bounded by

$$\rho(\tau_0, z, Q, T) \leq \frac{E[X]}{k} = \frac{\tau(\tau_0, z, Q/k)\frac{k}{T}}{k} = \frac{\tau_0}{T} + \frac{zQ}{kT}. \tag{4}$$

◀

Table 1 shows how the upper bound varies depending on the expected summary block interval T and the subblock count k. The orphan rate for Bitcoin [43] appears in the column for $k = 1$. In this analysis we assume that $Q = 32\,\text{MB}$, $z = 100\,\text{MB/s}$, and $\tau_0 = 5\,\text{s}$. Generally, we can see that the orphan rate decreases with both expected summary block interval T and k. This suggests that, when summary block intervals are constant across protocols, Tailstorm with high k increases fairness over Bitcoin and Tailstorm with lower k.

4.2   Measuring Fairness in Simulation

Section 4.1 provides an analytical bound for Tailstorm's orphan rate. This bound is only tight if all subblocks originate from different miners, which is unlikely to happen in practice. To obtain a more realistic analysis, albeit in a more specific setting, we now consider strong miners who are likely to produce multiple blocks within a single propagation delay. Since the mathematical analysis for this scenario is complex, we rely on simulation instead. Additionally, instead of using the orphan rate metric, itself only a proxy for fairness, we directly measure a miner's deviation from their fair share of rewards.

We implement the virtual environment as described in Section 2, Algorithm 1 and measure how the choice of protocol – Bitcoin and Tailstorm under various parameterizations – affects rewards. The network is configured with one *weak* and one *strong* miner operating with 1 % and 99 % of the total hash rate, respectively. All messages are delayed for 6 seconds.

We simulate one million PoW solutions per configuration. We split the simulation into *independent observations*, each representing one day of protocol execution. Specifically, with T denoting the expected summary block interval, we terminate individual executions when $\lfloor 24 \times 3600 \times k/T \rceil$ PoWs are mined. For each execution, we identify the longest chain of blocks and calculate the accumulated rewards for both miners. Since each observation covers one day of mining, the variance represents daily volatility of rewards.

A miner's fair share of reward equals its relative hash rate. To measure fairness, we compare actual rewards to the fair share. Figure 2 shows the weak miner's relative reward as a percentage of its fair share. The leftmost facet shows the Bitcoin protocol for expected
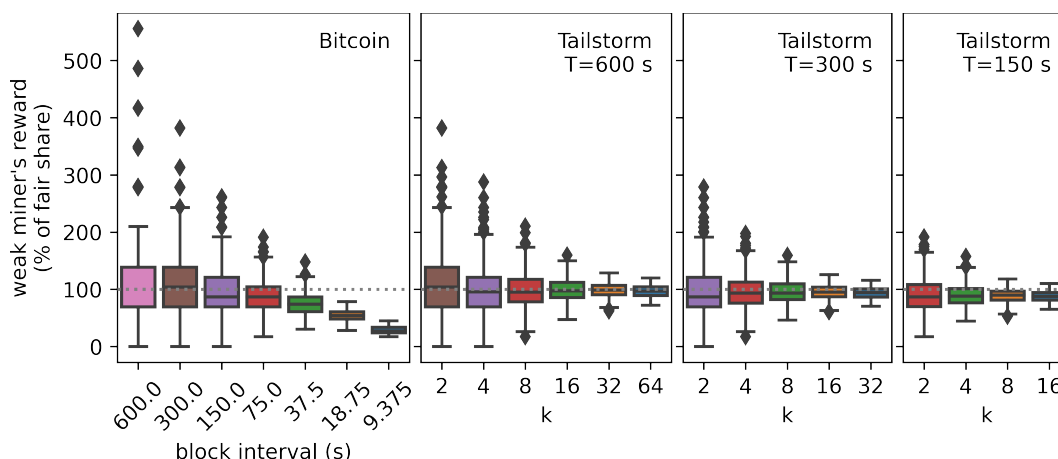
■ **Figure 2** Observed fairness for different configurations of Bitcoin (leftmost facet) and Tailstorm (right facets) with expected summary interval T ranging from 600 to 150 seconds. Colors represent expected subblock mining intervals.

block intervals ranging from roughly 9 seconds to 600 seconds. The remaining three facets show Tailstorm with *summary* block intervals T of 600, 300, and 150 seconds for k ranging from 2 to 64. Like colors represent such configurations where *subblocks* are generated at the same frequency $\lambda = k/T$. We omit the configurations where the expected subblock interval T/k is lower than the network propagation delay of 6 seconds.

The Figure illustrates the tradeoff between reward fairness and volatility in Bitcoin and Tailstorm. In the leftmost facet, it shows how increasing the expected block interval improves fairness but also increases volatility in Bitcoin. In contrast, the right facets show how Tailstorm can be configured to independently control both reward fairness (by adjusting the summary block interval T) and volatility (by adjusting k). In particular, choosing higher k leads to more frequent rewards and lower volatility, while longer summary block intervals tend to increase fairness. The latter observation aligns with the results in Table 1, which also shows that fairness tends to increase with the summary block interval.

5 Attack Evaluation

We now extend the virtual environment defined by Algorithm 1 in Section 2 to model adversarial behavior. We then specify and evaluate several attack strategies against Tailstorm.

To account for the possibility of collusion among multiple dishonest parties, we pessimistically assume that they indeed collude. Therefore, we model a single but strong node deviating from the protocol. Throughout the remaining sections we refer to the single dishonest node as the *attacker* and to the honest nodes as *defenders*. The attacker *observes* the virtual environment's state and reacts to any updates accordingly. We describe attacks as *policies*, which determine the *action* to take based on the observed state.

5.1 Network

We deploy a virtual environment comprising n nodes. The first node represents the attacker and is assigned fraction α of the total hash rate. The remaining hash rate is distributed evenly among the $n-1$ defenders. Henceforth, we refer to α as the *attacker's strength*.

In Bitcoin, blocks form a linear chain. If there are two blocks with the same height, only one will be included in the blockchain long term. Since honest Bitcoin nodes prefer and mine on the block first received, nodes with better connectivity obtain an advantage. The same *block race* situation can arise in Tailstorm. Here, there can be only one *summary* at each height and the nodes' preference of summaries (Alg. 3) likewise depends on the order they are received. Attackers able to send blocks more quickly might be able to manipulate the defenders' preference to suit their needs.

We follow Eyal and Sirer [16] and Sapirshtein et al. [47] who model the *block race advantage* with a single network parameter γ, which defines the proportion of defenders, weighted by hash rate, that will opt for the attacker's block in the case of a block race. We pessimistically assume that the attacker receives all blocks without delay. Defender nodes can send blocks to one another with a negligible delay of ε. When the attacker sends a block to a defender, we introduce a random delay, which we draw independently from a uniform distribution on the interval $[0, \frac{n-2}{n-1}\frac{\varepsilon}{\gamma}]$.

Note that we chose these delays to enable comparison with related work [16, 47] on selfish mining against Bitcoin. Other constructions with different distributions are possible but redundant: $\gamma = 0$ implies an inferior network level attacker who wins *no* block races, $\gamma = 1$ models a superior attacker who wins *all* block races, and intermediate attackers can be modeled by choosing γ accordingly. The effect of non-negligible delays among the defenders was analyzed in Section 4.

▶ **Theorem 2.** *Let $\gamma \in [0,1]$ be given. For any choice of $\varepsilon > 0$ and $n > \frac{1}{1-\gamma} + 1$, our definition of γ is equal to the attacker's block race advantage in [16, 47].*

Proof. Let A be the event that a randomly chosen honest miner accepts the attacker block over honest and A_i the event that a specific miner accepts the attacker block. By the law of total probability we have $P[A] = \frac{1}{n-1} \sum_{i=1}^{n-1} P[A_i]$. Note also that if we assume w.l.o.g. that A_{n-1} was the miner of the competing block, then $P[A_{n-1}] = 0$. Since defenders communicate with each other with delay ε while the attacker communicates with defenders with uniform random delay up to time $\frac{n-2}{n-1}\frac{\varepsilon}{\gamma}$, it must be the case that

$$P[A_i] = \varepsilon \left(\frac{n-2}{n-1}\frac{\varepsilon}{\gamma} \right)^{-1}, \quad \forall i < n-1, \tag{5}$$

provided that $\frac{n-2}{n-1}\frac{\varepsilon}{\gamma} > \varepsilon$. Choosing $n > \frac{1}{1-\gamma} + 1$ satisfies this constraint. Equality in the constraint is not valid because it forces $P[A_i] = 1$ for all γ. Therefore,

$$P[A] = \frac{1}{n-1} \sum_{i=1}^{n-1} P[A_i] \tag{6}$$

$$= \frac{1}{n-1} \sum_{i=1}^{n-2} P[A_i] \tag{7}$$

$$= \frac{n-2}{n-1} \frac{\varepsilon}{\frac{n-2}{n-1}\frac{\varepsilon}{\gamma}} \tag{8}$$

$$= \gamma. \tag{9}$$

◀

Constraint $n > \frac{1}{1-\gamma} + 1$, from Theorem 2, is fundamental; if n is smaller, then it is not possible for an honest miner to communicate a block to another honest miner faster than the attacker can. The constraint implies that $\gamma < \frac{n-2}{n-1}$. Hence, we can model $\gamma = 1$ only in the
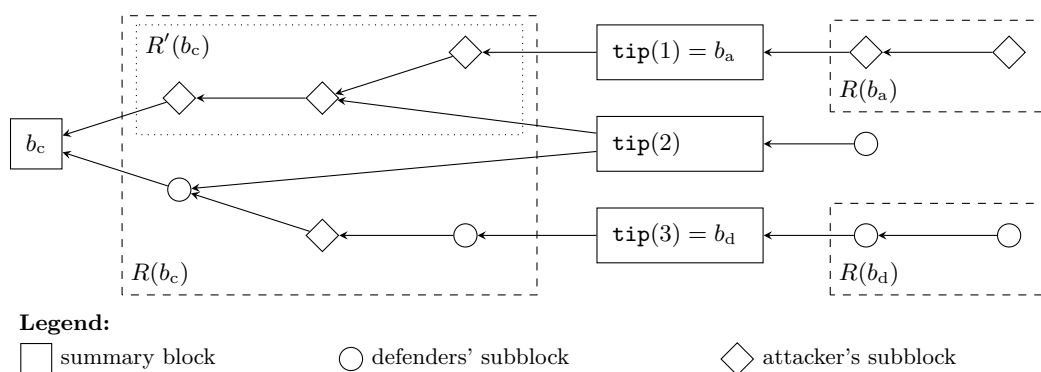
Figure 3 Notation in Section 5.2 Observation Space. This example uses three nodes and $k = 3$.

limit that the number of miners approaches infinity, which is not possible in practice. Other authors [16, 47] have directly considered $\gamma = 1$, but we feel that the restriction to $\gamma < 1$ is natural. Given that the miner of the defending block in a block race will never adopt the attacker's competing block, $\gamma = 1$ implies that individual miners have zero hash rate, which is not realistic.

5.2 Observation Space

The virtual environment maintains complex state. It tracks the full history of blocks, delayed messages, and the partial views of all nodes. Some of this information is not available to attackers in practice, other parts are not relevant. To enable concise policies, we follow Sapirshtein et al. [47] and restrict what the attacker can see.

Figure 3 illustrates our notation. At any given time, let $b_a = \mathtt{tip}(1)$ denote the attackers preferred summary block, and let b_d denote the best block (according to `Preference` in Alg. 3) among the preferred blocks of the defenders: $\mathtt{tip}(2), \ldots, \mathtt{tip}(n)$. We use b_c to refer to the best summary block among the common ancestors of b_a and b_d. In other words, b_c is the latest block that *all* nodes agree upon. For any summary block b, we use $R(b)$ to identify the subblocks which are both a descendant of b and have the same height as b. Note that Tailstorm's chain structure (Alg. 2) implies that $R(b)$ and b together span a tree. Section 3 defines the *depth* of a subblock to be its depth within this tree. We define $R'(b)$ as the largest subset of $R(b)$ such that all subblocks are mined by the attacker and $R'(b) \cup \{b\}$ is still connected. In the following, we use $|S|$ to refer to the cardinality of a set S and $\mathtt{depth}[S]$ to refer to the maximum depth among all blocks in S. The attacker observes

h_a: the attacker's height advantage, $\mathtt{height}(b_a) - \mathtt{height}(b_c)$,

h_d: the defenders' height advantage, $\mathtt{height}(b_d) - \mathtt{height}(b_c)$,

s_a: the attacker's inclusive subblock count, $|R(b_a)|$,

s'_a: the attacker's exclusive subblock count, $|R'(b_a)|$,

s_d: the defenders' subblock count, $|R(b_d)|$,

d_a: the attacker's inclusive depth, $\mathtt{depth}[R(b_a)]$,

d'_a: the attacker's exclusive depth, $\mathtt{depth}[R'(b_a)]$, and

d_d: the defenders' depth, $\mathtt{depth}[R(b_d)]$.

The example shown Figure 3 implies $(h_a, h_d, s_a, s'_a, s_d, d_a, d'_a, d_d) = (1, 1, 2, 2, 2, 2, 2, 2)$.

> ◼ **Algorithm 5** Tailstorm: Attacker.
>
> 1 **function** Update(tip t, block b) **is**   // t: currently preferred block // b: new block
> 2 $pref, share, append \leftarrow t, [], []$;   // new preferred tip, blocks to broadcast, templates to append
> 3   **if** b is own summary block **then** $pref \leftarrow b$;
> 4 $W \leftarrow$ set of withheld blocks (= blocks not shared before);
> 5 $withhold, extend \leftarrow \mathcal{P}(h_\mathrm{a}, h_\mathrm{d}, s_\mathrm{a}, s'_\mathrm{a}, s_\mathrm{d}, d_\mathrm{a}, d'_\mathrm{a}, d_\mathrm{d})$;   // $\mathcal{P}$: attack policy
> 6   **switch** $withhold$ **do**
> 7   **case** Adopt **do** $pref \leftarrow b_\mathrm{d}$ // b_d: defender's block, see Sec. 5.2 ;
> 8   **case** Match **do** $share \leftarrow \{x \in W \mid \mathtt{progress}(x) \leq \mathtt{progress}(b_\mathrm{d})\}$;
> 9   **case** Override **do** $share \leftarrow \{x \in W \mid \mathtt{progress}(x) \leq \mathtt{progress}(b_\mathrm{d}) + 1\}$;
> 10   **case** Wait **do** nothing;
> 11   **switch** $extend$ **do**
> 12   **case** Inclusive **do**
> 13   **if** $|R(pref)| \geq k$ **then**   // $R(b)$: full subblock tree confirming b, see Sec. 5.2
> 14 $append \leftarrow$ summary template using $R(pref)$;
> 15   **case** Exlusive **do**
> 16   **if** $|R'(pref)| \geq k$ **then**   // $R'(b)$: partial subblock tree confirming b, see Sec. 5.2
> 17 $append \leftarrow$ summary template using $R'(pref)$;
> 18   **return** $pref$, $share$, $append$;

5.3 Action Space

The virtual environment (Alg. 1) calls a node's Update function whenever this node learns about a new block: after it was mined locally with PoW, appended locally without PoW, or received from the network. We implement the attacker's *dishonest* Update function in Algorithm 5. We allow for generic attacks by letting the attacker choose from a set of potentially dishonest actions. We assume that there is a policy $\mathcal{P}$ which maps observations (see Sect. 5.2) to action tuples of the form (*withhold*, *extend*). The *withhold* action type controls the preferred tip of the chain and the withholding of blocks. We hereby follow closely the actions used by Sapirshtein et al. [47] for selfish mining against Bitcoin [16].

Wait Continue mining on b_a and withhold new blocks.
Match Release just enough blocks to induce a block race between b_d and an attacker block.
Override Release just enough blocks to make the defenders discard their block b_d.
Adopt Abort attack, prefer defenders' block b_d, and discard b_a.

Recall from Section 2 that "blocks" refers to vertices in the block DAG. This makes the *withhold* action protocol-agnostic. For Tailstorm, blocks can be subblocks or summaries. Note that while Adopt and Wait are always feasible, Match and Override are not. If the attacker's chain is too short, the defenders will not consider adopting it. In such cases, Match and Override, as implemented in Algorithm 5, fall back to releasing all withheld blocks.

The second action type, *extend*, is specific to Tailstorm. It controls how the attacker assembles new summary blocks, more specifically, which subblocks it considers for selection with Algorithm 4.

Inclusive Use all available subblocks to create new summaries.
Exclusive Use only subblocks which were mined by the attacker.

Note that the attacker never delays the next summary block longer than necessary. As soon as there are enough subblocks for an inclusive or exclusive summary (according to the chosen action), this summary will be created.
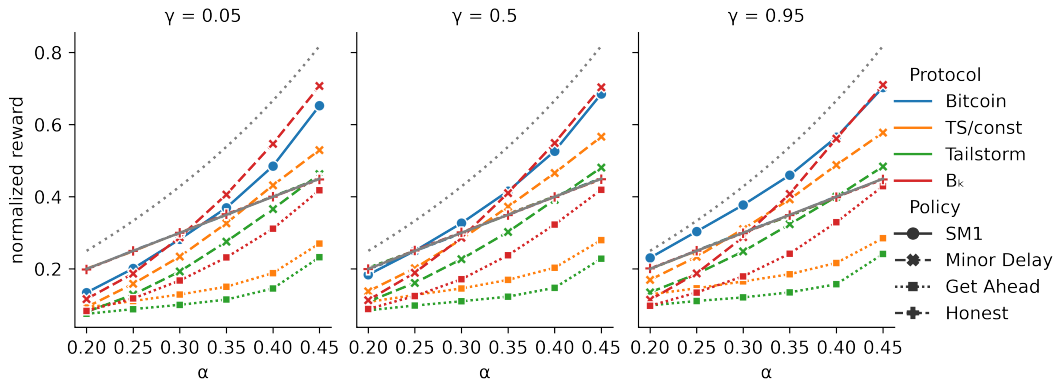
Figure 4 Observed normalized reward as a function of attacker strength α for different protocols (color) and reference policies (line style and markers). The three facets represent the network assumptions $\gamma \in \{5, 50, 95\}\,\%$. We set protocol parameter $k = 8$ where applicable. TS/const is Tailstorm without reward discounting. We add a gray reference line for the fair reward α (overlaps with honest policy) and a gray dotted curve for the known upper bound $\alpha/(1-\alpha)$.

5.4 Reference Policies

We evaluate the following policies. In each, the attacker assembles `Inclusive` summaries.

Honest Emulate Algorithm 3: adopt the longest chain and release all blocks:
    `Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. Otherwise `Override`.
Get Ahead Withhold own subblocks, release own summaries:
    `Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. `Override` if $h_\mathrm{d} < h_\mathrm{a}$. Otherwise `Wait`.
Minor Delay Withhold own subblocks, override defender summaries as they come out:
    `Adopt` if $h_\mathrm{d} > h_\mathrm{a}$. `Wait` if $h_\mathrm{d} = h_\mathrm{c}$. Otherwise `Override`.

To evaluate the policies above, we reuse the simulator from Section 4.2, configuring the network as described in Sections 5.1 to 5.3. We set block race advantage $\gamma \in \{5, 50, 95\}\,\%$ and allow the attacker strength α to range from 20 % to 45 %. We run one hundred simulations per configuration, and stop each as soon as the DAG contains 2048 blocks. At the end of each simulation we select the longest chain of blocks and calculate its rewards. We configure Tailstorm's discount reward scheme, as defined in Section 3.1, with $c = 1$ such that at most one unit of reward is minted per unit of chain progress.

Some policies cause more orphans than others. Assuming effective difficulty adjustment according to Section 3.3, the amount of simulated time depends on the attacker's behaviour. The attacker's mining cost is proportional to the time spent on the attack. To facilitate comparison across policies, we normalize rewards with respect to simulated time. Formally, we define the *normalized reward* as the attacker's reward up to block b divided by `progress`(b). Note that relative reward, a metric commonly used for Bitcoin [55, 16, 47, 54], is not sufficient to account for Tailstorm's reward discounting.

Figure 4 reports the average normalized reward (across 100 simulations) on the y-axis for the varying α on the x-axis and with γ varying by facet. The green curves represent the different reference policies against Tailstorm. We also evaluate the reference policies against the TS/const protocol as defined in Section 3.4 (orange curves) and against $\mathcal{B}_k$ (red curves). In addition, we evaluate the SM1 policy against Bitcoin (blue curve) as described by Sapirshtein et al. [47]. For orientation we include their upper bound $\alpha/(1-\alpha)$ as a gray dotted curve and add a solid gray line for α, the expected reward of honest behaviour.

The figure supports multiple conclusions. First, the curves for the *Honest* policy coinciding with the line for α indicates that the policy indeed replicates honest behavior in all evaluated protocols. Second, comparing the measurements for Tailstorm with the ones for TS/const shows clearly that the discounting of rewards reduces efficacy of all dishonest reference policies. Third, *Minor Delay* is generally the most effective reference policy. Forth, *Minor Delay* produces less reward in Tailstorm than SM1 does in Bitcoin. Fifth, for low γ, *Minor Delay* is more profitable against $\mathcal{B}_k$ than SM1 is against Bitcoin.

Note however, that *Minor Delay* might not be the best strategy against Tailstorm, just like SM1 is not always optimal for Bitcoin [47]. This motivates to search for optimal policies.

6   Attack Search

Previous research has identified optimal attacks against Bitcoin and Ethereum through the use of Markov Decision Processes (MDP) and exhaustive search [55, 21, 47]. However, for more complex protocols, the state space of MDPs can become prohibitively large, and exhaustive search becomes impractical [26, 29, 54]. As a result, many authors in the past have resorted to evaluating hard-coded attacks instead of searching for optimal policies [9, 17, 29, 34, 13].

We adopt the approach introduced by Hou et al. [26] and employ reinforcement learning (RL) to search for attacks. The observation and action spaces described in Section 5 readily define a partially observable MDP. To search for optimal attacks, we replace the hard-coded policy P with an RL agent that learns to select actions based on past observations.

We utilize off-the-shelf RL tooling by first exposing our simulator and attack space as an OpenAI Gym [10]. Next, we deploy Proximal Policy Optimization (PPO) [48] as our agent. To support reproduction and future research, we release the Gym as Python package [28] and include all training scripts in our open source repository [30].

The modularity of our simulator allows us to apply the same RL pipeline to different protocols. We insert the protocol specifications along with their associated observation and action spaces, while reusing the virtual environment and network assumptions across protocols. For Bitcoin, we adopt the attack space defined by Sapirshtein et al. [47]. Details about the attack spaces for Bitcoin and $\mathcal{B}_k$ are provided in the extended version of this paper.

We use Bitcoin as a reference protocol to measure the completeness of our approach. Previous research by Sapirshtein et al. [47] has yielded the optimal policy. As we will demonstrate, our search mechanism reproduces these results closely.

We search for optimal policies for all possible combinations of attacker strength $\alpha \in \{20, 25, 30, \ldots 45\}\%$, and block race advantage $\gamma \in \{5, 50, 95\}\%$. We evaluate four protocols: Bitcoin, $\mathcal{B}_k$, Tailstorm, and TS/const with $k = 8$ where applicable. In total, this amounts to $7 \cdot 3 \cdot 4 = 84$ different learning problems.

For each learning problem, we conduct multiple training runs with varying hyperparameters. For the objective function we choose *normalized reward*, as evaluated for the reference policies in Figure 4. From the training we obtain 1292 policies.

We select the best trained policy for each problem by simulating 100 independent protocol executions for each and proceeding as follows. As in Section 5.4, we stop individual executions as soon as there are 2048 blocks. We then select the longest chain of blocks and observe the attacker's normalized reward. This reward is averaged over the 100 observations per policy to determine the policy with the highest average reward. For reference, we apply the same filtering method to select the best policy among the reference policies from Section 5.4.
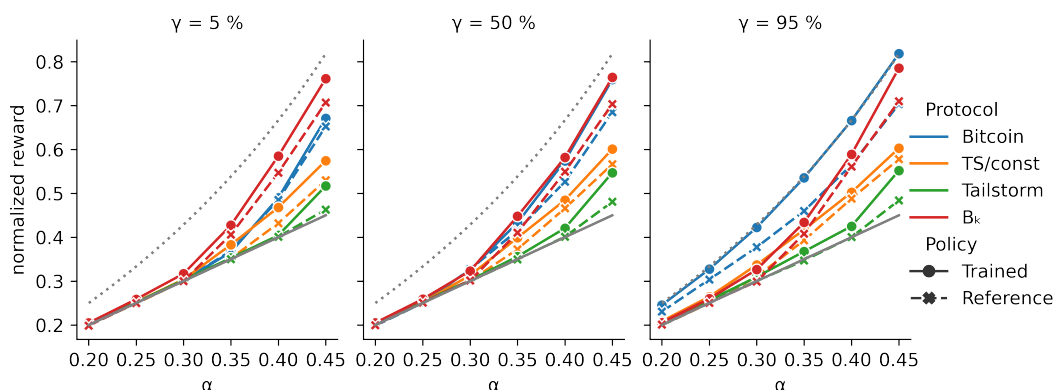
■ **Figure 5** Observed normalized reward of the *best known policy* with the axes set up like in Figure 4. Solid curves with ●-markers represent trained reinforcement learning models. Dashed curves with ×-markers represent the reference policies defined in Section 5.4.

Figure 5 shows the performance of the selected policies, with γ varying by facet, α on the x-axis, and average normalized reward on the y-axis. The solid colored curves represent the best trained policies, while the dashed colored curves represent the best reference policies. As in Figure 4, we include gray reference curves for Bitcoin's known upper bound $\alpha/(1 - \alpha)$ (dotted) and fair share α (solid). Note that for $\gamma = 50\%$, Nakamato and $\mathcal{B}_k$ visually overlap.

In comparing the performance of different protocols, we say that protocol A performs *better* (or *worse*) than B in the event that A returns fewer (alternatively *more*) rewards to the attacker than does B. Figure 5 supports the following conclusions. First, $\mathcal{B}_k$ performs worse than Bitcoin for low γ, but better than Bitcoin for high γ. The break-even point seems to be at $\gamma = 50\%$. Second, Tailstorm consistently outperforms Bitcoin and $\mathcal{B}_k$, even with constant rewards (TS/const). Moreover, discounting of rewards consistently makes Tailstorm less susceptible to selfish mining and similar incentive layer attacks. Third, the learned policies consistently match or outperform the hard-coded reference policies. This finding is important because anything else would indicate deficiencies in the training. Forth, the learned policies are close to optimal for Bitcoin. This can be seen in two ways. Firstly, Sapirshtein et al. showed that the SM1 policy is close to optimal for $\gamma = 0$ [47, Fig. 1a]. Our figure for $\gamma = 5\%$ shows that the learned policy matches SM1 (the dashed blue curve with ×-markers). Secondly, the authors showed that the optimal policy reaches the upper bound $\alpha/(1 - \alpha)$ for $\gamma = 1$ [47, Fig. 1c]. Our figure for $\gamma = 95\%$ shows that the learned policy matches this bound as well.

Until now, we have evaluated the efficacy of policies in absolute terms. We now take a different perspective and ask: how strong must an attacker be for dishonest behaviour to pay off? Recall that honest behaviour implies an expected normalized reward of α (solid gray line in Figures 4 and 5). To answer the question, we calculate break-even points, which represent the minimum relative hash rate α, such that following the policy produces more than α of normalized reward. We start from the optimal policies presented in Figure 5 and select only those policies that feature dishonest behavior. Recall that each of the remaining policies was trained on a fixed relative hash rate α. For each protocol and choice of γ, we select the dishonest policy trained for the lowest α. We then evaluate this policy against alternative α values ranging from 5% to 50%. Our simulator provides noisy observations and the reward distribution varies stochastically. Hence, we use Bayesian optimization to minimize the difference between the observed reward and α, i.e., to get as close to the

Table 2 Break-even points: minimal α (in %) where dishonest behaviour (Figure 5) pays off.

γ	Trained Policy			Reference Policy		
	5 %	50 %	95 %	5 %	50 %	95 %
Bitcoin	33.6	26.5	≤ 5	32.7	24.5	≤ 5
TS/const	28.9	28.6	21.7	37.9	32.6	28.4
Tailstorm	41.6	35.8	35.8	43.7	41.0	39.3
$\mathcal{B}_k$	22.2	27.8	25.3	31.3	31.1	30.7

break-even point as possible. Table 2 reports the results. Observe that for the trained policies, the break-even points are consistently either lower or close to the break-even points of the hard-coded reference policies. $\mathcal{B}_k$, TS/const, and Tailstorm are also less sensitive to changes in γ than Bitcoin. Tailstorm has the highest break-even points among all protocols, indicating that it is most resilient to incentive layer attacks.

7   Tailstorm Cryptocurrency

So far, we have focused on Tailstorm's consensus and incentive mechanisms. Throughout our analysis, we have assumed a cryptocurrency on the application layer that facilitates the creation and distribution of rewards to participants of the consensus layer. In this section, we describe and discuss the Tailstorm cryptocurrency and our prototype implementation.

7.1   Transaction Handling

Tailstorm implements the unspent transaction output (UTXO) model as it is used in Bitcoin [39]: Each UTXO represents a designated amount of cryptocurrency. Transactions consume and create UTXOs, but they never create more cryptocurrency than they consume. Consumed UTXOs cannot be consumed again. Ownership and transfer of value follows from restricting the consummation of UTXOs to the holders of specific cryptographic keys.

Tailstorm uses the same public key cryptography as Bitcoin, and it also supports Bitcoin's UTXO scripting facility. The dissemination and processing of transactions follows Bitcoin's approach as well. However, as we describe next, Tailstorm deviates from Bitcoin in how transactions are recorded in the blockchain.

Taking inspiration from Storm [4], each subblock contains a list of transactions. Let $\mathtt{tx_a}$ be a transaction listed at position $\mathtt{idx_a}$ in subblock $\mathtt{sub_a}$ summarized in summary block $\mathtt{sum_a}$ and let $\mathtt{tx_b}$, $\mathtt{idx_b}$, $\mathtt{sub_b}$, and $\mathtt{sum_b}$ be defined similarly. Assume $\mathtt{tx_a}$ and $\mathtt{tx_b}$ are incompatible, e. g., because they spend the same UTXO twice. If $\mathtt{tx_a}$ and $\mathtt{tx_b}$ are listed in the same subblock, i.e., $\mathtt{sub_a} = \mathtt{sub_b}$ and $\mathtt{idx_a} \neq \mathtt{idx_b}$, then this subblock is marked invalid. We proceed similarly if $\mathtt{tx_a}$ and $\mathtt{tx_b}$ are summarized by different summaries: due to Tailstorm's chain structure (see Alg. 2) $\mathtt{sum_a}$ and $\mathtt{sum_b}$ must differ in height and we mark the higher one as invalid. Honest nodes ignore invalid blocks and all their descendants at the consensus layer. Hence, under the above circumstances, incompatible transactions are not persisted in the blockchain, as is the case in Bitcoin.

However, special attention is required for incompatible transactions in different subblocks summarized by the same summary block, i.e., $\mathtt{sub_a} \neq \mathtt{sub_b}$ and $\mathtt{sum_a} = \mathtt{sum_b}$. A critical assumption of our analyses as well as parallel PoW in general [29] is that subblocks confirming the same summary are compatible and thus can be mined independently. We thus cannot

mark the summary invalid in this case and the incompatible transactions are persisted in the blockchain. To resolve this conflict, Tailstorm executes only one of the transactions at the application layer and ignores the other according the following rules. Let $\mathtt{dst_a}$ denote the distance of $\mathtt{sub_a}$ to $\mathtt{sum_a}$ in the block DAG, and let $\mathtt{dst_b}$ be defined similarly. If $\mathtt{dst_a} < \mathtt{dst_b}$, then $\mathtt{tx_a}$ is ignored. Ties are broken by the PoW hash function, i.e., if $\mathtt{dst_a} = \mathtt{dst_b}$ and $\mathtt{hash(sub_a)} > \mathtt{hash(sub_b)}$, then $\mathtt{tx_a}$ is ignored. If $\mathtt{tx_a}$ is ignored then all dependent transactions (i.e. those spending the UTXOs produced by $\mathtt{tx_a}$) are ignored as well. Note that even though incompatible transactions are persisted in the blockchain, the double-spending semantics are equivalent to Bitcoin: offending transactions off the longest branch are ignored.

7.2   Fast Confirmations

To reap the full consistency guarantees of parallel PoW [29], prudent cryptocurrency users should wait for one summary block confirmation before accepting their transactions as final. For example, if their transaction is included in subblock $\mathtt{sub_a}$ and first summarized in $\mathtt{sum_a}$, then they should wait for another valid summary $\mathtt{sum_b}$ with $\mathtt{height(sum_b)} = \mathtt{height(sum_a)}+1$. Assuming a 10 minute summary block interval and large k, the full confirmation will likely occur in 10 to 20 minutes, depending on whether the transaction was included early in the subblock tree or later.

If time is short and the transacted value is low, e.g. if the user is selling a cup of coffee to go, they can consider waiting for a number of subblock confirmations instead. For example, consider Tailstorm with $k = 60$ and a 10 minute summary block interval. Subblocks will be mined every 10 seconds in expectation. The heuristics are similar to a fast version of Bitcoin: If the seller waits for 6 subblock confirmations, i.e., for a subblock $\mathtt{sub_b}$ with $\mathtt{depth(sub_b)} = \mathtt{depth(sub_a)} + 6$, the settlement will take about one minute. Invalidation of the payment, e.g. due to a double spend, implies a fork of the subblock tree of length 6 or more. Tailstorm discounts mining rewards according to the depth of the subblock tree; whereas the tree could have achieved depth 60, it now can achieve depth at most 54. One tenth of the minted rewards is lost, and the cost of the coffee is dwarfed.

7.3   Tailstorm Prototype

We have implemented Tailstorm and make the code available online [23]. We started from a fork Bitcoin Unlimited's implementation of Bitcoin Cash. Our fork implements the Tailstorm consensus layer and incentive mechanism as specified in Section 3. The prototype uses $k = 3$, but is easily configurable by changing a compile-time flag. The DAA is calibrated for a target summary block interval of ten minutes. Subblocks are expected to arrive every 200 seconds.

To minimize propagation delays, we have implemented several network compression mechanisms. First, we adapt the Graphene protocol [41] to avoid redundant transmission of transactions. Second, we adapt the Compact Block protocol [12] to reduce the size of summary blocks. Third, the transaction list is encoded bit-efficiently in each subblock.

We leave the implementation of Tailstorm's application layer, i.e. the transaction handling as described in Section 7.1, for future work. Finally, we note that our implementation has minimal testing and, being a prototype, is not ready for production use.

8   Discussion

Tailstorm inherits the consistency guarantees of parallel PoW consensus [29]. To increase fairness, we discount rewards based on the tree structure of subblocks which itself is inspired from Bobtail [9]. To obtain fast confirmations, we write transactions into subblocks like

Storm [4]. Alongside these foundational influences, we incorporate insights from a variety of related works, which we here can only list partially. For more details we refer to the surveys of Garay and Kiayias [19], describing the different ways of defining the consensus problem, and Bano et al. [6], focusing on the different solution approaches and protocols.

Tailstorm improves fairness by avoiding orphans and discounting rewards. The former is not new [2, 50, 51], however, the motivation differs. Sompolinsky and Zohar [51] propose to improve Bitcoin's *consistency* by referencing blocks that would otherwise be orphaned (called *uncles*) and then counting both blocks and uncles (GHOST-rule) whereas Bitcoin counts only the blocks (longest chain rule). However, the authors explicitly avoid rewarding uncles and hence do not improve *fairness*. Follow-up work [2, 50] does not discuss rewards at all. In Ethereum PoW, a deployed variant of GHOST [51], and Kaspa, a deployed variant of GHOSTDAG [50], uncles receive *partial* rewards and the blocks that include uncles get *additional* rewards. The unfair dynamic of Bitcoin, where included blocks receive the full reward and orphans none, is largely preserved. We think that GHOST-like protocols can become more fair by applying Tailstorm's ideas: discounting of rewards, applied equally to all involved miners.

Pass and Shi [42] explicitly set out to increase fairness in PoW. Their Fruitchains protocol uses two kinds of blocks like Tailstorm. Blocks record fruits and fruits record transactions. Unlike in Tailstorm, both fruits and blocks require PoW. Applying the *2-for-1 trick* of Garay et al. [20], both are mined with the same hash puzzle observing different bits of the output. Fruits have a lower difficulty and thus are created more frequently. The rewards are distributed evenly across recent fruits while blocks receive nothing. According to Zhang and Preneel [54], Fruitchains are more vulnerable to incentive layer attacks than Bitcoin for block race advantage $\gamma = 0$, and thereby less fair. Furthermore, Fruitchains suffer from a tradeoff between fairness and transaction confirmation time [54]. A related line of research [5, 53] extends the *2-for-1* into an *n-for-1* trick and makes the miners work on multiple chains in parallel. The chains are then interleaved to form a single coherent transaction ledger. Unlike Fruitchains, however, Prism [5] and OHIE [53] focus on consistency, liveness, and throughput. We leave for future work to investigate whether and how the fairness of these protocols can be improved by applying Tailstorm's reward discounting.

Kiffer and Rajaraman [33] propose to discount rewards inversely proportional to the overall hash rate participating in the system. This reduces centralization in their model, but we worry that motivating lower hash rates might harm the primary goals of consensus: consistency and liveness. Their mechanism poses an alternative to Bitcoin's halving mechanism, which statically reduces block rewards by 50 % roughly once every 4 years, to avoid long term inflation of the cryptocurrency. In contrast, Tailstorm's reward discounting is focused on the short term and punishes non-linearities within a single subblock tree.

But our approach is not without limitations. In Section 4, we analyze the impact of message propagation delays on fairness. We assume that these delays are uniform, affecting all miners equally. However, in practice, propagation delays depend on the connectivity of individual miners. Some inequalities arise from economies of scale, larger miners can invest more in low latency connections, others from the underlying physics of information propagation: miners located in the same region or on the same continent implicitly form are cartel, whereas joining the network from a distant location puts them at a disadvantage. We leave for future work to analyze how Tailstorm is affected by more realistic network topologies [1, 14, 38, 44, 45].

In Section 6, we perform a search for optimal attack strategies. Our search algorithm is based on reinforcement learning (RL), which means it is not exhaustive. It is possible that the RL agent did not discover the optimal strategy against certain protocols. This

challenges our conclusion that Tailstorm is less susceptible to incentive layer attacks than the other protocols, Bitcoin and $\mathcal{B}_k$. To encourage further exploration we release the RL Gym environment on PyPI [28], allowing others to discover more effective attacks. Assuming the absence of better strategies, our results can be confirmed by encoding our attack space as a Markov Decision Process (MDP) and employing exhaustive search techniques, as has been done for Bitcoin [21, 47], Ethereum [55] and other longest chain protocols [54]. However, to the best of our knowledge, such techniques have not yet been successfully applied to DAG-based protocols like Tailstorm.

Another limitation is that our action space might not cover all possible attacks. We closely follow the modeling of the selfish mining attack space against Bitcoin [47], where we observe consent that the four actions `Adopt`, `Match`, `Override`, and `Wait` are indeed complete. We are convinced that adding two actions for summary formation, `Inclusive` and `Exclusive`, is enough to represent the most profitable attacks. However, our conclusions can be challenged by presenting a more effective incentive layer attack against Tailstorm, which cannot be expressed in our attack space. We encourage this endeavour by making all analytical code available online [30]. After modifying the attack space, new strategies can be drafted, evaluated, and optimized, while the virtual environment, protocol specifications, and RL attack search can be reused without changes.

On a separate note, Carlsten et al. [11] demonstrate that selfish mining becomes more profitable when considering transaction fees in addition to mining rewards. They present a strategy targeting Bitcoin which leverages transaction fees to outperform honest behavior for any $\alpha > 0$ and $\gamma < 1$. Similar attacks are likely feasible against all PoW cryptocurrencies, including Tailstorm, however they also exceed the scope of this paper.

Lastly, Arnosti and Weinberg [3] show that even small cost imbalances or economies of scale lead to highly centralized PoW mining ecosystems. While, Tailstorm reduces the imbalances compared to other PoW protocols we studied, it cannot fully mitigate centralization. Our notion of fairness revolves around rewarding miners in proportion to their hash rate. This definition is not new [1, 7, 31, 42, 54]. However, from an economic standpoint, to prevent centralization, miners should be rewarded in proportion to their operational mining costs [3]. Unfortunately, achieving this goal seems to be impossible. Mining costs primarily depend on the price and the availability of energy and specialized mining hardware. Purchasing energy in large quantities tends to be more cost-effective, and strong miners may even consider operating their own power plants. Scaling up operations also enables miners to develop and deploy more efficient mining hardware, and then selling it to weak miners after it has become outdated [52]. Given that these unfair scaling effects do not only affect Tailstorm, but PoW cryptocurrencies in general, one might be lead to consider alternative approaches like proof-of-stake [22, 32, 49]. But even then, it remains questionable whether any permissionless system can fully avoid centralization [36].

9 Conclusion

Tailstorm integrates parallel PoW [29], partial transaction confirmation [4, 9], and a novel incentive mechanism – reward discounting – into a PoW cryptocurrency that provides fast and secure confirmations for its users and fair rewards for its miners. We thereby solve a long standing issue of longest chain protocols, where the operator has to choose between either a short block interval with fast but less reliable confirmations and unfair rewards, or a long block interval with slow confirmations and more fair but infrequent rewards. Our prototype demonstrates that Tailstorm can replace Bitcoin not only theoretically but also in practice.

Contribution Statement

George originally developed the idea of reward discounting as a way to combat withholding attacks in Bobtail. George and Gregory refined that idea into an early version of Tailstorm suitable for a hard fork of Bitcoin Cash. They also contributed to the prototype implementation. Patrik contributed the protocol specification, simulation-based analyses, and reinforcement learning interface. George contributed the analytical orphan rate analysis. George and Patrik contributed the hard-coded reference policies. Ben and Patrik conducted the attack search with reinforcement learning. Guided by earlier stages of the analyses, Patrik provided improvements to Tailstorm consensus and its transaction handling. Patrik and George wrote this paper.

References

1 Mohamed Alzayat, Johnnatan Messias, Balakrishnan Chandrasekaran, Krishna P. Gummadi, and Patrick Loiseau. Modeling coordinated vs. P2P mining: An analysis of inefficiency and inequality in proof-of-work blockchains. *CoRR*, abs/2106.02970, 2021. `arXiv:2106.02970`.

2 Ignacio Amores-Sesar, Christian Cachin, and Anna Parker. Generalizing weighted trees: a bridge from bitcoin to GHOST. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 156–169. ACM, 2021. `doi:10.1145/3479722.3480995`.

3 Nick Arnosti and S. Matthew Weinberg. Bitcoin: A natural oligopoly. *Manag. Sci.*, 68(7):4755–4771, 2022. `doi:10.1287/mnsc.2021.4095`.

4 awemany. Storm. `https://github.com/awemany/storm-sim/blob/master/whitepaper/`, 2019.

5 Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 585–602. ACM, 2019. `doi:10.1145/3319535.3363213`.

6 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 the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 183–198. ACM, 2019. `doi:10.1145/3318041.3355458`.

7 Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J. Marmolejo Cossío. Fairness and efficiency in dag-based cryptocurrencies. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2020. `doi:10.1007/978-3-030-51280-4_6`.

8 George Bissias. Radium: Improving dynamic pow targeting. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2020 International Workshops, DPM 2020 and CBT 2020, Guildford, UK, September 17-18, 2020, Revised Selected Papers*, volume 12484 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2020. `doi:10.1007/978-3-030-66172-4_24`.

9 George Bissias and Brian Neil Levine. Bobtail: Improved blockchain security with low-variance mining. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. URL: `https://www.ndss-symposium.org/ndss-paper/bobtail-improved-blockchain-security-with-low-variance-mining/`.

10 Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. `arXiv:1606.01540`.

11 Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 154–167. ACM, 2016. `doi:10.1145/2976749.2978408`.

12 Matt Corallo. BIP152: Compact Block Relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, 2016.

13 Francisco J. Marmolejo Cossío, Eric Brigham, Benjamin Sela, and Jonathan Katz. Competing (semi-)selfish miners in bitcoin. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 89–109. ACM, 2019. `doi:10.1145/3318041.3355471`.

14 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 Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 550–566. Springer, 2019. `doi:10.1007/978-3-030-32101-7_32`.

15 Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 859–878. ACM, 2020. `doi:10.1145/3372297.3417290`.

16 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014. `doi:10.1007/978-3-662-45472-5_28`.

17 Chen Feng and Jianyu Niu. Selfish mining in ethereum. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, pages 1306–1316. IEEE, 2019. `doi:10.1109/ICDCS.2019.00131`.

18 Daniel Fullmer and A. Stephen Morse. Analysis of difficulty control in bitcoin and proof-of-work blockchains. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 5988–5992. IEEE, 2018. `doi:10.1109/CDC.2018.8619082`.

19 Juan A. Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, volume 12006 of *Lecture Notes in Computer Science*, pages 284–318. Springer, 2020. `doi:10.1007/978-3-030-40186-3_13`.

20 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. `doi:10.1007/978-3-662-46803-6_10`.

21 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 Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 3–16. ACM, 2016. `doi:10.1145/2976749.2978341`.

22   Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017. `doi:10.1145/3132747.3132757`.

23   Greg Griffith and George Bissias. Tailstorm node implementation. `https://gitlab.com/georgebissias/BCHUnlimited/-/tree/tailstorm_prototype`, 2023.

24   Dongning Guo and Ling Ren. Bitcoin's latency-security analysis made simple. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 244–253. ACM, 2022. `doi:10.1145/3558535.3559791`.

25   Thomas M. Harding. Real-time block rate targeting. *Ledger*, 5, 2020. `doi:10.5195/ledger.2020.195`.

26   Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramèr, Giulia Fanti, and Ari Juels. Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. URL: `https://www.ndss-symposium.org/ndss-paper/squirrl-automating-attack-analysis-on-blockchain-incentive-mechanisms-with-deep-reinforcement-learning/`.

27   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, 2017. `doi:10.4173/mic.2017.4.1`.

28   Patrik Keller. PyPI release of our reinforcement learning environment as OpenAI Gym. `https://pypi.org/project/cpr-gym/0.7.0/`, 2023.

29   Patrik Keller and Rainer Böhme. Parallel proof-of-work with concrete bounds. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 1–15. ACM, 2022. `doi:10.1145/3558535.3559773`.

30   Patrik Keller and Ben Glickenhaus. Source code for our simulator, evaluations, and attack search. `https://github.com/pkel/cpr/tree/aft23`, 2023.

31   Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 365–382. ACM, 2016. `doi:10.1145/2940716.2940773`.

32   Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017. `doi:10.1007/978-3-319-63688-7_12`.

33   Lucianna Kiffer and Rajmohan Rajaraman. Happy-mine: Designing a mining reward function. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 250–268. Springer, 2021. `doi:10.1007/978-3-662-64331-0_13`.

34   Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 729–744. ACM, 2018. `doi:10.1145/3243734.3243814`.

35   Daniel Kraft. Difficulty control for blockchain-based consensus systems. *Peer-to-Peer Netw. Appl.*, 9(2):397–413, 2016. `doi:10.1007/s12083-015-0347-x`.

36   Yujin Kwon, Jian Liu, Minjeong Kim, Dawn Song, and Yongdae Kim. Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 110–123. ACM, 2019. `doi:10.1145/3318041.3355463`.

37   Jing Li, Dongning Guo, and Ling Ren. Close latency-security trade-off for the nakamoto consensus. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 100–113. ACM, 2021. `doi:10.1145/3479722.3480992`.

38   Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stütz, and Bernhard Haslhofer. All that glitters is not bitcoin - unveiling the centralized nature of the BTC (IP) network. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–9. IEEE, 2020. `doi:10.1109/NOMS47738.2020.9110354`.

39   Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008.

40   Kevin Alarcón Negy, Peter R. Rizun, and Emin Gün Sirer. Selfish mining re-examined. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2020. `doi:10.1007/978-3-030-51280-4_5`.

41   A. Pinar Ozisik, Gavin Andresen, Brian Neil Levine, Darren Tapp, George Bissias, and Sunny Katkuri. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In Jianping Wu and Wendy Hall, editors, *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, pages 303–317. ACM, 2019. `doi:10.1145/3341302.3342082`.

42   Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 315–324. ACM, 2017. `doi:10.1145/3087801.3087809`.

43   Peter R. Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016. URL: `https://ledgerjournal.org/ojs/index.php/ledger/article/view/40`.

44   Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 199–213. ACM, 2019. `doi:10.1145/3318041.3355469`.

45   Elias Rohrer and Florian Tschorsch. Blockchain layer zero: Characterizing the bitcoin network through measurements, models, and simulations. In *46th IEEE Conference on Local Computer Networks, LCN 2021, Edmonton, AB, Canada, October 4-7, 2021*, pages 9–16. IEEE, 2021. `doi:10.1109/LCN52139.2021.9524930`.

46   Sheldon M. Ross. *Introduction to Probability Models*. Elsevier, 2014. `doi:10.1016/C2012-0-03564-8`.

47   Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 2016. `doi:10.1007/978-3-662-54970-4_30`.

48   John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. `arXiv:1707.06347`.

49   Jakub Sliwinski and Roger Wattenhofer. Asynchronous proof-of-stake. In Colette Johnen, Elad Michael Schiller, and Stefan Schmid, editors, *Stabilization, Safety, and Security of Distributed Systems - 23rd International Symposium, SSS 2021, Virtual Event, November*

17-20, 2021, Proceedings, volume 13046 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2021. `doi:10.1007/978-3-030-91081-5_13`.

50 Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 57–70. ACM, 2021. `doi:10.1145/3479722.3480990`.

51 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015. `doi:10.1007/978-3-662-47854-7_32`.

52 Aviv Yaish and Aviv Zohar. Pricing ASICs for cryptocurrency mining. In *Proceedings of the 5th ACM Conference on Advances in Financial Technologies, AFT 2022, Princeton, NJ, USA, October 21-25, 2023*. LIPIcs, 2023.

53 Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. OHIE: blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 90–105. IEEE, 2020. `doi:10.1109/SP40000.2020.00008`.

54 Ren Zhang and Bart Preneel. Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 175–192. IEEE, 2019. `doi:10.1109/SP.2019.00086`.

55 Roi Bar Zur, Ittay Eyal, and Aviv Tamar. Efficient MDP analysis for selfish-mining in blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 113–131. ACM, 2020. `doi:10.1145/3419614.3423264`.