Practical Settlement Bounds for Proof-of-Work Blockchains

Peter Gaži
IOG
peter.gazi@iohk.io

Ling Ren
University of Illinois at Urbana-Champaign
renling@illinois.edu

Alexander Russell
University of Connecticut
IOG
acr@uconn.edu

ABSTRACT

Nakamoto proof-of-work ledger consensus currently underlies the majority of deployed cryptocurrencies and smart-contract blockchains. While a long and fruitful line of work has succeeded to identify its exact security region—that is, the set of parametrizations under which it possesses *asymptotic* security—the existing theory does not provide concrete settlement time guarantees that are tight enough to inform practice.

In this work we provide a new approach for obtaining concrete and practical settlement time guarantees suitable for reasoning about deployed systems. We give an efficient method for computing explicit upper bounds on settlement time as a function of primary system parameters: honest and adversarial computational power and a bound on network delays. We implement this computational method and provide a comprehensive sample of concrete bounds for several settings of interest. We also analyze a well-known attack strategy to provide lower bounds on the settlement times. For Bitcoin, for example, our upper and lower bounds are within 90 seconds of each other for 1-hour settlement assuming 10 second network delays and a 10% adversary. In comparison, the best prior result has a gap of 2 hours in the upper and lower bounds with the same parameters.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; **Distributed systems security**.

KEYWORDS

Bitcoin, proof of work, blockchains

ACM Reference Format:
Peter Gaži, Ling Ren, and Alexander Russell. 2022. Practical Settlement Bounds for Proof-of-Work Blockchains. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559368

1 INTRODUCTION

Nakamoto proof-of-work consensus, introduced in the 2008 Bitcoin white paper [19], is the basic algorithmic framework supporting the sensational Bitcoin and Ethereum blockchains. This charmingly simple protocol has inspired a large body of analytic work which—after over a decade of attention—has finally determined the protocol's *security region*: specifically, two independent recent articles [6, 10] determine the exact conditions under which the protocol *eventually* achieves consensus. In greater detail, they identify the region of critical parameters (honest and adversarial hashing power, network delays) under which the probability of a consistency failure has the form $\exp(-\Omega(t))$, where t is the amount of time a given transaction has been included in the blockchain.

Despite offering interesting theoretical insights, such asymptotic guarantees tell us very little about concrete settlement times. In particular, their proof techniques are intentionally optimized for simplicity over precision, and they make no effort to achieve reasonable—or even explicit—constants in the results. This state of affairs is especially frustrating as it leaves conspicuously unanswered the most fundamental question faced by users of deployed cryptocurrencies and blockchains:

How long must I wait for a transaction to settle?

One prominent feature of Nakamoto consensus is that the settlement question has a parametric answer: a block (and its transactions) achieves higher certainty with longer waiting times. The ideal answer would thus determine the exact probability of a settlement failure as a function of the elapsed time or the number of subsequent blocks amassed on top of the block of interest.

Towards this end, the Bitcoin white paper [19] analyzed the transaction settlement time under a specific attack called the *private mining* attack. However, this approach is clearly unsatisfactory. The gold standard for security analysis calls for identification of a well-defined and widely accepted threat model that places limitations on the adversary without prescribing its concrete actions and, in the context of the model, a proof that the protocol remains secure against *any* adversary.

The model widely adopted for analyzing blockchains, which we also employ in this work, gives the adversary the ability to adaptively delay any messages sent by honest players and make corrupt parties deviate arbitrarily from the protocol. To make the adversary more powerful, corrupt parties are assumed to be connected by a zero-latency network so that they can act with perfect knowledge of each other's states—thus the adversary can be characterized by a single entity with the collective hashing power of all corrupt parties. The model additionally posits an upper bound on the fraction of hashing power controlled by corrupt parties. On the other hand, honest players are assumed to follow the protocol to the letter (see Section 2.1). Our model does not cover attacks exploiting rational behavior of parties, such as the selfish-mining attacks [8], beyond the simple expedient of considering such parties corrupt.

The main sticking point in coping with such a general model is accounting for possible network delays. Indeed, if one is content to assume an instantaneous network, two recent works [1, 6] provide exact analysis for proof-of-work blockchains. Most existing blockchain analysis with network delays [3, 6, 9, 10, 14, 16, 20, 21, 23] only give asymptotic bounds and do not directly speak to the

CCS '22, November 7–11, 2022, Los Angeles, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3559368

question. Some recent works derive concrete bounds by working out the constants in these asymptotic analyses [3, 16], but the resulting settlement bounds are very weak; for Bitcoin, the results come to thousands of hours—orders of magnitude larger than what is used in practice. Only recently, Li et al. [17] derived the first practically viable settlement time upper bounds; their results are still a few hours larger than corresponding lower bounds.

Our results. We lay out a new proof technique for analyzing consistency of proof-of-work blockchains with an eye toward explicit settlement times. Our method offers striking improvements over the best previous work—typically by a factor of 10 or more—in explicit settlement times for both the Bitcoin setting (with long interblock arrival times) and the Ethereum setting (with short interblock arrival times).[1] In both settings, the settlement times we obtain are within minutes of optimality.

1.1 An Overview of the Analysis

We capture the schedule of mining successes and the output of a concrete execution by a *characteristic string* and a *PoW-tree* respectively, two notions introduced for this purpose in the context of proof-of-stake [13] and adapted to proof-of-work (PoW) in [1, 10]. Our analysis departs almost immediately from [10] by shifting its focus to *serialization.* Given a sequence of mining successes, the longest-chain algorithm will produce a "hash tree" of blocks, where edges are given by the predecessor hashes in the generated blocks. The partial order assigned to blocks by this tree may be inconsistent with the "real" order in which they were generated due both to network delays and adversarial players postponing delivery of their blocks. But there is always a reordering of the block creation events for which the tree has a simple "temporally consistent" explanation—such a reordering is a *serialization.* Motivated by the fact that the analysis is more tractable in the lockstep-synchronous setting, we will first carry out the analysis there. We then study the serializations that can arise with network delays, and then rely on the lockstep results to analyze the serialized executions.

Ultimately, we are interested in studying the longest chain rule in a continuous-time model $C[\Delta_\mathrm{r}]$ where honest parties and the adversary both create proofs of work according to independent Poisson processes with rates r_h and r_a, respectively, and the adversary may selectively delay honest block delivery by up to Δ_r time ("r" stands for "real"). One way to capture this setting is to consider discrete slots corresponding to very short intervals of length t ($t \ll \Delta_\mathrm{r}$) and to appropriately adjust the network model so that honest parties are not guaranteed to see messages that were sent to them at most $\Delta \triangleq \lceil \Delta_\mathrm{r}/t \rceil$ slots ago. Call this model $\mathcal{D}[\Delta, t]$, where the two parameters record the maximum delay *in slots* and the duration of the slot, respectively. This is the approach taken in [10, 14, 20]; and taking $t \to 0$ makes this model approach $C[\Delta_\mathrm{r}]$ [10]. However, one difficulty in tackling this model is in keeping track of the complex delay patterns that can occur as each individual message can be delayed by anywhere between 0 to Δ_r/t slots.

The alternative approach that we propose in this work is to introduce two "adjacent" discrete models. The first model $\mathcal{D}[0, \Delta_\mathrm{r}]$ is a lockstep-synchronous model, dividing time into relatively long slots of length Δ_r. Honest players producing blocks in the same slot are not apprised of each other's blocks, but the network is assumed to deliver all created (honest) blocks at the end of each slot. Of course, the adversary always operates with full knowledge of all adversarial and honest blocks produced in any slot. The second model $\mathcal{D}[1, \Delta_\mathrm{r}]$ is similar, and a slot still represents a Δ_r-long time interval, but an honest block may now be delivered at the end of the *next* slot, i.e., the slot following the one in which the block was created; this effectively permits that some messages are delayed by up to $2\Delta_\mathrm{r}$.

It is easy to observe that $C[\Delta_\mathrm{r}]$ and $\mathcal{D}[\Delta, t]$ are "sandwiched" between these two models

$$\mathcal{D}[0, \Delta_\mathrm{r}] \preceq \mathcal{D}[\Delta, t], C[\Delta_\mathrm{r}] \preceq \mathcal{D}[1, \Delta_\mathrm{r}]$$

in the sense that any valid execution in a model on the left-hand side of $\preceq$ is also valid in the model on the right-hand side, as the restriction on delays gets more permissive as we move to the right. Therefore, an upper bound on the probability of settlement failure in a right-hand side model is also an upper bound in a left-hand side one; in other words, the $\mathcal{D}[0, \Delta_\mathrm{r}]$ model settles more quickly than the model of interest $\mathcal{D}[\Delta, t]$, while $\mathcal{D}[1, \Delta_\mathrm{r}]$ settles more slowly.

We first analyze consistency in $\mathcal{D}[0, \Delta_\mathrm{r}]$ in Section 3. The analysis follows essentially the same lockstep trajectory of [6] and can be given fairly succinctly. We then shift our attention in Section 4 to $\mathcal{D}[1, \Delta_\mathrm{r}]$ where the core technical difficulty lies. Our approach here is to show how to serialize an execution in the $\mathcal{D}[1, \Delta_\mathrm{r}]$ model *to an execution in* $\mathcal{D}[0, \Delta_\mathrm{r}]$. We can then rely on upper bounds obtained in the simpler $\mathcal{D}[0, \Delta_\mathrm{r}]$ model.

We evaluate numerically our results in Section 5.1. We then compare our upper bound results with lower bounds, which we obtain in Section 5.2 by analyzing the success probability of the private mining attack in the $\mathcal{D}[0, \Delta_\mathrm{r}]$ model. Recall that the $\mathcal{D}[0, \Delta_\mathrm{r}]$ model settles more slowly than the $\mathcal{D}[\Delta, t]$ model and the private mining attack may not be the most effective adversarial strategy. Therefore, this gives a lower bound on the consistency failure (or settlement times) of PoW blockchains.

In the rest of the paper, we reserve the symbol Δ to denote the maximum message delay *in slots.* Hence, in Section 3 (resp. 4), which employs the model $\mathcal{D}[0, \Delta_\mathrm{r}]$ (resp. $\mathcal{D}[1, \Delta_\mathrm{r}]$), we consider $\Delta = 0$ (resp. $\Delta = 1$). However, recall that in both these models, a slot itself has duration Δ_r.

We remark that our analysis does not consider difficulty adjustments that are present in PoW protocols. This is well justified by the fact that block settlement time is much shorter than the difficulty adjustment period (hours vs. weeks).

1.2 Sample Results Generated by our Method

As mentioned, our results provide very sharp estimates for Bitcoin in the region of practical interest: see Figure 1. With a 10% adversary and a bound of 10 seconds on network delay, we obtain a settlement error of no more than 4.489% after one hour. This can be directly compared with a lower bound of 4.261%, which is obtained by analyzing the private mining attack. Notably, these results are "only minutes" apart: 90 seconds before the one hour mark, the lower bound is 4.494%. So the upper bound is less than 90 seconds away from optimal. Alternatively, our results yield a 0.48% settlement

[1]A concurrent work obtains simple and close-form bounds that are tight for the Bitcoin setting but loose for the Ethereum setting [12].

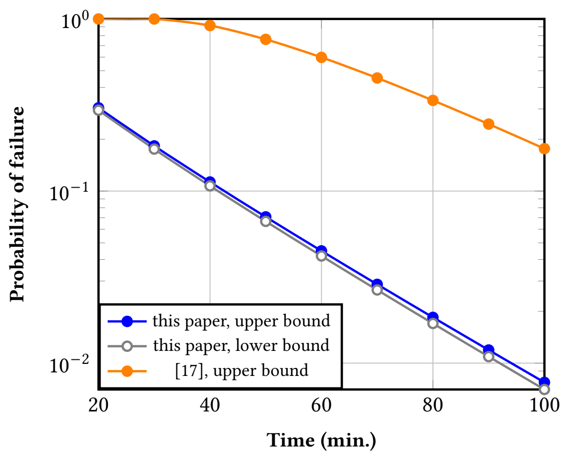

Figure 1: Our upper and lower bounds on settlement failure probability for Bitcoin with a 10% adversary and 10 second network delays; results from [17] included for comparison

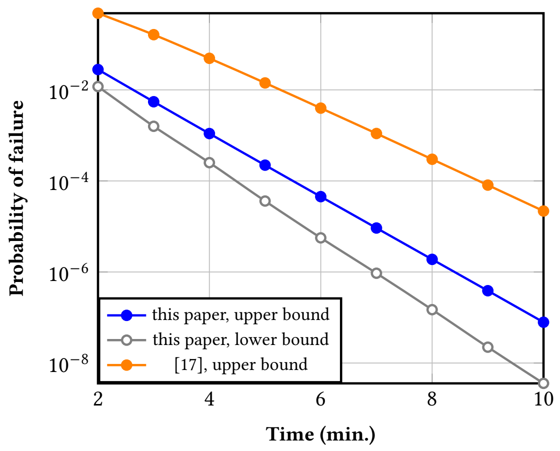

Figure 2: Our upper and lower bounds on settlement failure probability for Ethereum with a 10% adversary and 2 second network delays; results from [17] included for comparison.

error guarantee for the 6-block confirmation rule that is common used in practice.

In the case of Ethereum,[2] recall that blocks are comparatively small and have 13 second interblock time. With 2 second network delays and a 10% adversary, our methods bound settlement failure probability after four minutes within 0.1097% and 0.02518%. As expected, we observe a larger gap than in the "nearly lockstep-synchronous" Bitcoin case. However, the result is still less than one minute away from the optimum: at the five minute mark the upper bound has fallen to 0.02219%. These results improve the settlement failure estimates of previous work by well over an order of magnitude in the regime of interest. See Figure 2 for a representative example of our results for Ethereum and a comparison with [17]. A more comprehensive discussion of both time-based and block-based settlement appears in Section 5.1.

2 PRELIMINARIES

Notation. Throughout the paper, $\mathbb{N} = \{0, 1, 2, \dots\}$ denotes the set of natural numbers (including zero). For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \dots, n\}$ (hence $[0] = \emptyset$). For a set X, we let $\mathcal{P}(X)$ denote the power set of X.

For a word of length n over alphabet Σ, we use the notation $w = w_1 \dots w_n \in \Sigma^n$. We denote by $w_{i:j}$ its subword $w_i w_{i+1} \dots w_j$, and $\#_a(w)$ denotes the number of occurrences of the symbol $a \in \Sigma$ in w. We denote by $\|$ the concatenation of words and by $\circ$ the concatenation of languages, i.e., $L_1 \circ L_2 \triangleq \{w_1 \| w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$.

2.1 Modeling Proof-of-Work Blockchains with Network Delays

Our modeling of the protocol and its execution environment extends the model in [10], which we summarize here for completeness.

A PoW blockchain protocol is carried out by a set of parties of two types: *honest* parties follow the protocol and *adversarial* parties may diverge arbitrarily. All parties actively engage in searching for "proofs-of-work" (PoWs), which afford them the right to contribute to the ledger. We divide time into slots of length Δ_r and use a *characteristic string* to indicate a summary of the outcomes of the proof-of-work lottery in each slot.

More concretely, our main alphabet of interest in this paper will be $\Sigma_\infty \triangleq \{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N}$. Intuitively, a single symbol $(s, a) \in \{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N}$ from this alphabet captures the outcome of the proof-of-work lottery in a given time slot, at a level of precision that will be most convenient for our treatment. The natural number a simply captures the number of adversarial successes; the symbol $s \in \{0, \mathsf{h}, \mathsf{H}\}$ captures the number of honest successes as follows: 0 represents no honest successes, h represents one and only one honest success, and H denotes more than one honest successes in the considered slot.

Note that a characteristic string symbol does *not* capture the full outcome of the lottery in a given slot: it merely describes the numbers of successes and their attribution to party types, but not their *ordering* within a slot. Looking ahead, it will be clear that our analysis implicitly assumes that this ordering is the best possible for the adversary, in line with our effort to obtain upper bounds on error probabilities.

For notational convenience when treating our imprecise accounting of honest successes described above, we define the following helper "rounding" function $\mathsf{round}_\mathsf{H} : \mathbb{N} \to \{0, \mathsf{h}, \mathsf{H}\}$. Let h be the number of honest successes in a given time slot and define

$$\mathsf{round}_\mathsf{H}(h) \triangleq \begin{cases} 0 & \text{if } h = 0\,, \\ \mathsf{h} & \text{if } h = 1\,, \\ \mathsf{H} & \text{if } h \geq 2\,. \end{cases} \tag{1}$$

We consider characteristic strings w (i.e., words) drawn from the set Σ_∞^L; these describe the outcomes of the proof-of-work lottery over a period of L slots. We write

$$w = (w_1, \dots, w_L) = ((s_1, a_1), \dots, (s_L, a_L))\,.$$

Let ε denote the empty characteristic string (i.e., $L = 0$).

[2]Note that while Ethereum considers *uncle blocks* for difficulty recalculation and rewards distribution, these blocks do not affect its chain-selection rule, hence Ethereum is fully covered by our analysis.

The Bitcoin protocol calls for parties to exchange *blockchains*, each of which is an ordered sequence of blocks beginning with a distinguished "genesis block," known to all parties. Each proof-of-work success confers on that party the right to add exactly one block to an existing blockchain. (In fact, the party must identify the previous chain on which she wishes to build ahead of time, but this will not affect our analysis.) Honest parties follow the *longest-chain rule* which dictates that they always choose to add to the longest blockchain they have observed thus far and broadcast the result to all other parties. The basic dynamics of the system, with a particular characteristic string w and an adversary, can be described as follows.

Let C_t denote the collection of all blockchains created by time t and let $H(C_t)$ denote the subset of all chains in C_t whose last block was created by an honest party. Set $C_0 = \{G\}$, where G denotes the unique chain consisting solely of the genesis block. The genesis block is "honest"; thus $H(C_0) = C_0$. It is convenient to adopt the convention that $C_{-t} = H(C_{-t}) = \{G\}$ for any negative integer $-t < 0$. Then the protocol execution proceeds as follows. For each slot $t = 1, 2, \ldots$:

- Initiate $C_t := C_{t-1}$ and $H(C_t) := H(C_{t-1})$.
- If $w_t = (0, a)$, the adversary may repeat the following *adversarial iteration a* times: select a single blockchain C from C_t and add a block to create a new chain C', which is added to C_t. $H(C_t)$ remains unchanged.
- If $w_t = (\mathsf{h}, a)$, the same a adversarial iterations happen as above, but they are arbitrarily interleaved with a single *honest iteration* defined as follows: the adversary may select any collection of chains $\mathcal{V}$ for which $H(C_{t-1-\Delta}) \subseteq \mathcal{V} \subseteq C_t$. This is the "view" of the honest player, who applies the longest chain rule to $\mathcal{V}$, selects the longest chain $L \in \mathcal{V}$ where ties are broken by the adversary, and adds a new block to create a new chain L' that is added to C_t and also $H(C_t)$.
- If $w_t = (\mathsf{H}, a)$, then the execution of a adversarial iterations is arbitrarily interleaved with at least two honest iterations.

In each time step t we also maintain the set of Δ-*dominant* chains $\mathcal{D}_t \subseteq C_t$, determined entirely by C_t and $H(C_{t-1-\Delta})$: namely, $\mathcal{D}_t$ is the set of all chains in C_t that are at least as long as the longest chain in $H(C_{t-1-\Delta})$. The intuition behind the definition of Δ-dominant chains is that, in a time slot t, it is in principle possible for the adversary to manipulate an honest party into adopting any Δ-dominant chain, as the adversary is only obligated to deliver those chains in $H(C_{t-1-\Delta})$ and the chains in $\mathcal{D}_t$ are at least as long as those in $H(C_{t-1-\Delta})$.

Note that the synchrony assumption is reflected in the description of the honest iteration: the adversary is obligated to deliver all chains produced by honest players that are Δ slots old. Although we keep the presentation general, recall that as explained in the introduction, this work focuses on the two models $\mathcal{D}[0, \Delta_{\mathsf{r}}]$ and $\mathcal{D}[1, \Delta_{\mathsf{r}}]$, and hence always considers $\Delta \in \{0, 1\}$.

Considering that the adversary selects both the view $\mathcal{V}$ of each honest player and is empowered to break ties, the structure of the resulting sequence of chains (that is, the directed acyclic graph naturally formed by the blocks) is determined entirely by the adversary and the characteristic string.

We make two final remarks. First, we permit the adversary to have full view of the characteristic string during this process. Of course, in practice a Bitcoin adversary must make decisions "online," so our modeling only makes the adversary stronger. Second, we have placed an implicit constraint on the adversary: the only means of producing a new chain is to append a block (associated with a proof-of-work success) to an existing chain. In practice, this constraint is guaranteed with cryptographic hash functions.

In the context of ledger protocols, one is usually interested in preserving two properties, *consistency* and *liveness*, formulated in [9, 15, 22]. Consistency means that once a block (or equivalently, a transaction within it) is *settled*, then it remains settled forever. We consider two settlement rules in this paper: a time-based one and a block-based one.

- **Consistency for time-based settlement; with parameter τ.** A block B that is mined before time ℓ and contained in some chain in $\mathcal{D}_t$ where $t \geq \ell + \tau$ is contained in every chain $C \in \mathcal{D}_{t'}$ for all $t' \geq t$.
- **Consistency for block-based settlement; with parameter k.** A block B that is k blocks deep in some chain in $\mathcal{D}_t$ is contained in every chain $C \in \mathcal{D}_{t'}$ for all $t' \geq t$.

Intuitively, the above settlement rules state that, when an honest player examines the longest chain to its knowledge at time t, it considers all blocks mined at least τ time earlier (in the case of time-based settlement) or buried k blocks deep (in the case of block-based settlement) settled. The focus of this paper is to bound the error probability (from both above and below) as a function of the settlement delay, i.e., of the parameters τ or k in the above two settlement rules, respectively.

We also remark that our definitions of consistency and its error probability above are applicable to individual blocks. One can also phrase the consistency as a global property of the protocol by requiring the above to hold for all blocks. However, the error probability of such a global consistency property will depend on the total running time of the blockchain protocol and is hard to characterize accurately.

For completeness, we also mention the liveness property [10].

- **Liveness; with parameter u.** For any two slots $t_1, t_2 > 0$ with $t_1 + u \leq t_2$, and any chain $C \in \mathcal{D}_{t_2}$, there is a time $t' \in \{t_1, \ldots, t_1 + u\}$ and a chain $C' \in H(C_{t'}) \setminus H(C_{t'-1})$ such that C' is a prefix of C.

2.2 Proof-of-Work Blocktrees

We formally capture the above protocol dynamics by the combinatorial notion of a *PoW Δ-tree*. It is a variant of the "fork" concept first considered for the proof-of-stake case in [2, 5, 13] and more recently also employed for PoW-analysis [1, 10].

Definition 2.1 (PoW Δ-tree). Let $\Delta, L \in \mathbb{N}$. A PoW Δ-*tree* for the string $w \in \Sigma_\infty^L$ is a directed, rooted tree $F = (V, E)$ (in the graph-theoretic sense) with a pair of functions

$$\mathsf{l}_\# : V \to \{0, \ldots, L\} \qquad \text{and} \qquad \mathsf{l}_{\mathsf{type}} : V \to \{\mathsf{h}, \mathsf{a}\}$$

satisfying the axioms below. Edges are directed "away from" the root so that there is a unique directed path from the root to any vertex. The value $\mathsf{l}_\#(v)$ is referred to as the *label* of v. The value

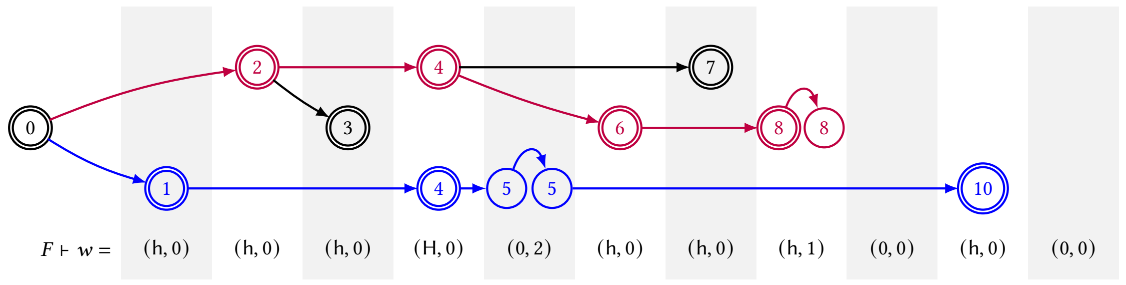

Figure 3: A PoW 1-tree F for the characteristic string w. Honest vertices are shown with double-struck boundaries, while adversarial vertices are simple circles. Vertices are labeled with $\mathsf{l}_\#(\cdot)$. The tree indicates a successful double spend attack—given by the red and blue chains—in a circumstance where the simple private-chain attack does not succeed: in particular, the tree constructs two alternate chains with disjoint suffixes of length 5, while only three adversarial proofs of work are discovered over this period. We remark that $F \equiv F_{\lceil 1}$, since the last symbol of w is $(0, 0)$, and that $\overline{F_{\lceil 1}}$ is obtained by removing the adversarial vertex with label 8. Thus $\mathsf{len}(\overline{F_{\lceil 1}}) = 5$, this maximum length achieved by the blue chain. Note, then, that the two chains indicated in red and blue each have 1-advantage equal to zero, and both are 1-dominant. Considering that these chains share no vertices after the root, they witness $\beta_1^1(F) \geq 0$ for the tree F and hence for the characteristic string w.

$\mathsf{l}_{\mathsf{type}}(v)$ is referred to as the *type* of the vertex: when $\mathsf{l}_{\mathsf{type}}(v) = \mathsf{h}$, we say that the vertex is *honest*; otherwise it is *adversarial*.

(A1) the root $r \in V$ is honest and is the only vertex with label $\mathsf{l}_\#(r) = 0$;
(A2) the sequence of labels $\mathsf{l}_\#()$ along any directed path is non-decreasing;
(A3) if $w_i = (s_i, a_i)$, then the number h_i of honest vertices of F with the label i satisfies $\mathsf{round}_{\mathsf{H}}(h_i) = s_i$, and there are no more than a_i adversarial vertices of F with the label i;
(A4) for any pair of honest vertices v, w for which $\mathsf{l}_\#(v) + \Delta < \mathsf{l}_\#(w)$, $\mathsf{len}(v) < \mathsf{len}(w)$, where $\mathsf{len}()$ denotes the depth of the vertex.

We will often refer to PoW Δ-trees simply as *trees* whenever Δ is clear from the context. Unless explicitly stated otherwise, throughout the paper we reserve the term 'tree' for the above PoW-specific structure, as opposed to the underlying graph-theoretic notion.

A PoW Δ-tree abstracts a protocol execution with a simple but sufficiently descriptive discrete structure. Its vertices and edges stand for blocks and their connecting hash links (in reverse direction), respectively. The root represents the genesis block, and for each vertex v, $\mathsf{l}_\#(v)$ and $\mathsf{len}(v)$ denote the slot in which the corresponding block was created and the block's depth, respectively.

It is easy to see the correspondence between the above axioms and the constraints imposed in the protocol execution. In particular, (A1) corresponds to the trusted nature of the genesis block; (A2) reflects that the blocks' ordering in a chain must be consistent with the order of their creation; (A3) reflects that honest players produce exactly one block per PoW success, while the adversary might forgo a block-creation opportunity; finally (A4) reflects the fact that given sufficient time, as needed for block propagation in the network, an honest party will take into account the blocks produced by previous honest parties.

Definition 2.2 (Tree notation). We write $F \vdash_\Delta w$ to indicate that F is a Δ-tree for the string w. If $F' \vdash_\Delta w'$ for a prefix w' of w, we say that F' is a *subtree* of F, denoted $F' \sqsubseteq F$, if F contains F' as a consistently-labeled subgraph. Given a Δ-tree F, we denote by $\overline{F}$ the maximal subtree of F having all leaves honest. We call two trees F_1 and F_2 *equivalent*, denoted $F_1 \equiv F_2$, if their underlying graphs and the $\mathsf{l}_{\mathsf{type}}(\cdot)$ functions are identical. Note that equivalent trees may only differ in their $\mathsf{l}_\#(\cdot)$ functions; whenever useful, we indicate the tree to which a labeling function belongs by a superscript (e.g. $\mathsf{l}_\#^F(\cdot)$).

An individual blockchain constructed during the protocol execution is represented by the notion of a *chain*, defined next.

Definition 2.3 (Chains). A path in a tree F originating at the root is called a *chain* (note that chains do not necessarily terminate at a leaf). As there is a one-to-one correspondence between directed paths from the root and vertices of a tree, we routinely overload notation so that it applies to both chains and vertices. Specifically, we let $\mathsf{len}(T)$ denote the *length* of the chain, equal to the number of edges on the path; recall that $\mathsf{len}(v)$ also denotes the depth of a vertex. We sometimes emphasize the tree from which v is drawn by writing $\mathsf{len}_F(v)$. We further overload this notation by letting $\mathsf{len}(F)$ denote the length of the longest chain in a tree F. Likewise, we let $\mathsf{l}_\#(\cdot)$ apply to chains by defining $\mathsf{l}_\#(T) \triangleq \mathsf{l}_\#(v)$, where v is the terminal vertex on the chain T. We say that a chain is *honest* if the last vertex of the chain is honest. For a vertex v in a tree F, we denote by $F(v)$ the chain in F terminating in v.

For two chains T, T' of a tree F, we write $T \sim_\ell T'$ if the two chains share a vertex with a label greater or equal to ℓ.

Intuitively, $T \sim_\ell T'$ guarantees that the respective blockchains agree on the state of the ledger up to time slot ℓ. Looking ahead, the adversary can make two honest parties disagree on the state of the ledger up to time ℓ only if she makes them hold two chains $T \nsim_\ell T'$.

Definition 2.4 (Tree trimming; dominance). For a string $w = w_1 \ldots w_n$ and some $k \in \mathbb{N}$, we let $w_{\lceil k} = w_1 \ldots w_{n-k}$ denote the string obtained by removing the last k symbols. For a tree $F \vdash_\Delta w_1 \ldots w_n$ we let $F_{\lceil k} \vdash_\Delta w_{\lceil k}$ denote the tree obtained by retaining only those vertices labeled from the set $\{1, \ldots, n-k\}$. We say that a chain T in F is Δ-*dominant* if $\mathsf{len}(T) \geq \mathsf{len}(\overline{F_{\lceil\Delta}})$ and simply call it *dominant* if Δ is clear from the context.

Observe that honest chains appearing in $F_{\lceil\Delta}$ are those that are necessarily visible to honest players at the end of the last time slot of the characteristic string. Correspondingly, the notion of a Δ-dominant chain matches the use of this term in Section 2.1.

2.3 Advantage and Margin

Definition 2.5 (Advantage α_F^Δ). For a Δ-tree $F \vdash_\Delta w$, we define the Δ-*advantage* of a chain $T \in F$ as

$$\alpha_F^\Delta(T) = \mathsf{len}(T) - \mathsf{len}(\overline{F_{\lceil\Delta}}) \ .$$

Observe that $\alpha_F^\Delta(T) \geq 0$ if and only if T is Δ-dominant in F.

Definition 2.6 (Margin β_ℓ^Δ). For $\ell \geq 1$, we define the Δ-*margin* of a tree F as

$$\beta_\ell^\Delta(F) = \max_{\substack{T^* \nsim_\ell T \\ T^* \text{ is } \Delta\text{-dominant}}} \alpha_F^\Delta(T) \,,$$

this maximum extended over all pairs of chains (T, T^*) where T^* is Δ-dominant and $T \nsim_\ell T^*$. We call the pair (T^*, T) the Δ-*witness chains for* F if the above conditions are satisfied; i.e., T^* is Δ-dominant, $T^* \nsim_\ell T$, and $\beta_\ell^\Delta(F) = \alpha_F^\Delta(T)$. Note that there might exist multiple such pairs in F, but under the condition $\ell \geq 1$ there will always exist at least one such pair, as the trivial chain T_0 containing only the root vertex satisfies $T_0 \nsim_\ell T$ for any T and $\ell \geq 1$, in particular $T_0 \nsim_\ell T_0$. For this reason, we will always consider β_ℓ^Δ only for $\ell \geq 1$.

We overload the notation and let

$$\beta_\ell^\Delta(w) = \max_{F \vdash_\Delta w} \beta_\ell^\Delta(F) \,.$$

We call a tree $F \vdash_\Delta w$ a Δ-*witness tree for* w if $\beta_\ell^\Delta(w) = \beta_\ell^\Delta(F)$; again many Δ-witness trees may exist for a string w.

Intuitively, $\alpha_F^\Delta(T)$ captures the length advantage (or deficit) of the chain T against the longest honest chain created at least Δ slots before the upcoming slot, which is hence now known to all honest parties. Consequently, $\beta_\ell^\Delta(F)$ records the maximal advantage of any chain T in F that potentially disagrees with some Δ-dominant chain T^* about the chain state up to slot ℓ. A negative $\beta_\ell^\Delta(F)$ hence indicates that the adversary cannot make an honest party holding T^* switch to any T that would potentially cause a revision of its ledger state up to slot ℓ. This connection between margin and consistency/settlement is exploited in previous work, for the PoW case it was made formal in [10, Lemma 1] in which the following fact is implicit:

Lemma 2.7 ([10]). *Consider an execution of a PoW blockchain for L slots as described in Section 2.1, resulting in a characteristic string $w = w_1 \dots w_L$. Let B be a block produced in slot $\ell \in [L]$, and let $t > \ell$ be such that B is contained in some chain $C \in \mathcal{D}_t$. If for every $t' \in \{t, \dots, L\}$ we have $\beta_\ell^\Delta(w_{1:t'}) < 0$ then B is contained in every $C' \in \mathcal{D}_{t'}$ for all $t' \in \{t, \dots, L\}$.*

This statement motivates our effort to upper-bound $\beta_\ell^\Delta(w)$ in the following sections.

Remark. One can define and study an analogous notion of consistency for protocols with unbounded lifetimes and, in fact, the explicit upper bounds we compute later in the paper reflect this stronger notion. Specifically, for a characteristic string $w = w_1 w_2 \dots$ and a finite ℓ, this requires that $\beta_\ell^\Delta(w_{1:t'}) < 0$ for all $t' \geq \ell$.

3 THE LOCKSTEP-SYNCHRONOUS ANALYSIS

In this section we focus on the simpler, so-called lockstep-synchronous setting, where all messages are delivered at the end of the slot in which they were sent, this corresponds to $\Delta = 0$ (the $\mathcal{D}[0, \Delta_\mathsf{r}]$ model). Throughout the section, as no confusion can arise, we omit the index 0 and write $F \vdash w$, $\alpha_F()$, $\beta_\ell()$, in place of $F \vdash_0 w$, $\alpha_F^0()$, $\beta_\ell^0()$, respectively. Note that now $\alpha_F(T) = \mathsf{len}(T) - \mathsf{len}(\overline{F})$.

Our main goal will be to obtain a simple recursive description of the margin quantity $\beta_\ell(w)$ for a characteristic string $w \in \Sigma_\infty^*$. Looking ahead, we will obtain an exact characterization (Theorem 3.6) that will then serve us later in Section 4 when establishing bounds for the margin $\beta_\ell^1(w)$ in the case with delays $\Delta = 1$.

3.1 The Fully Serialized Setting ($\Sigma_\mathsf{ser} = \{\mathsf{h}, \mathsf{a}\}$)

We begin the analysis of the lockstep-synchronous setting by considering an additional simplifying assumption that block creations are fully serialized, i.e., exactly one block is created in each time slot. Specifically, we work with a reduced alphabet $\Sigma_\mathsf{ser} = \{(\mathsf{h}, 0), (0, 1)\}$ for characteristic strings, and use the abbreviations $\mathsf{h} = (\mathsf{h}, 0)$ and $\mathsf{a} = (0, 1)$; thus we treat characteristic strings over the alphabet $\{\mathsf{h}, \mathsf{a}\}$. The definition of tree remains unchanged.

The following exact characterization of β_ℓ in the lockstep (i.e., $\Delta = 0$), fully serialized (i.e., with alphabet Σ_ser) setting was given in prior work [1] and serve as an instructive starting point of our investigation. Recall that ε is the empty characteristic string.

Lemma 3.1 ([1, Lemma 1]). *Fix $\ell \geq 1$. We consider characteristic strings $w \in \Sigma_\mathsf{ser}^*$. By definition $\beta_\ell(\varepsilon) = 0$. We have*

$$\beta_\ell(w\mathsf{a}) = \beta_\ell(w) + 1 \,,$$

$$\beta_\ell(w\mathsf{h}) = \begin{cases} \beta_\ell(w), & \text{if } \beta_\ell(w) = 0 \text{ and } |w\mathsf{h}| < \ell, \\ \beta_\ell(w) - 1, & \text{otherwise.} \end{cases}$$

Thus, prior to slot ℓ, β_ℓ performs a biased *barrier walk* with a barrier at 0; *after* round ℓ, it performs a standard biased random walk (without any barriers).

3.2 The Multi-Honest Setting ($\Sigma_\mathsf{mh} = \{\mathsf{h}, \mathsf{H}, \mathsf{a}\}$)

We now slightly generalize the treatment of Section 3.1 and consider characteristic strings over an alphabet that allows for multiple honest (hence the "mh" subscript) successes in a single slot. Namely, we consider $\Sigma_\mathsf{mh} = \{(\mathsf{h}, 0), (\mathsf{H}, 0), (0, 1)\} \subset \Sigma_\infty$, and use the shorthands $\{\mathsf{h}, \mathsf{H}, \mathsf{a}\}$ for these three symbols, respectively. The definition of a tree again remains unchanged.

Lemma 3.2. *Fix $\ell \geq 1$. We consider characteristic strings $w \in \Sigma_{\mathsf{mh}}^*$. By definition $\beta_\ell(\varepsilon) = 0$. We have*

$$\beta_\ell(w\mathsf{a}) = \beta_\ell(w) + 1,$$

$$\beta_\ell(w\mathsf{h}) = \begin{cases} \beta_\ell(w), & \text{if } \beta_\ell(w) = 0 \text{ and } |w\mathsf{h}| < \ell, \\ \beta_\ell(w) - 1, & \text{otherwise,} \end{cases} \tag{2}$$

$$\beta_\ell(w\mathsf{H}) = \begin{cases} \beta_\ell(w), & \text{if } \beta_\ell(w) = 0, \\ \beta_\ell(w) - 1, & \text{otherwise.} \end{cases}$$

Informally, the reason why H has a different effect on β_ℓ than h after slot ℓ is as follows. If $\beta_\ell(w) = 0$, this means that there are two competing chains of the *same*, maximal length that can be served to honest parties; now the adversary can orchestrate things so that the two (or more) honest successes occurring in this slot contribute to both of these chains equally, and hence they don't improve the situation for the honest parties. We call this effect a *neutralization* of honest successes. Note that, in contrast, a unique honest success h improves the situation for the honest parties in the "tie" case of $\beta(w) = 0$, as it extends only one of the chains, creating a unique longest chain.

The proof of Lemma 3.2 is an extension of the proof of Lemma 3.1 that appeared in [1], accounting for presence of H symbols in the considered characteristic string, we defer it to the full version of this paper [11].

3.3 The General Case ($\Sigma_\infty = \{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N}$)

We finally consider the full alphabet $\Sigma_\infty = \{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N}$. Intuitively, our approach here is to assign to any "rich" characteristic string $w \in \Sigma_\infty^*$ a set of "possible serializations" $R_0(w) \subseteq \Sigma_{\mathsf{mh}}^*$ such that any tree over w can be interpreted (via relabeling) as a tree over one of these Σ_{mh}-serializations, and vice versa. This then allows to precisely characterize $\beta_\ell(w)$ in terms of $\beta_\ell()$ of these Σ_{mh}-serializations, which are already understood in Lemma 3.2.

Serialization of the general alphabet. We define a serialization mapping $R_0 \colon \Sigma_\infty \to \mathcal{P}(\Sigma_{\mathsf{mh}}^*)$ as follows:

$$R_0(0, k) = \left\{\mathsf{a}^k\right\},$$

$$R_0(\mathsf{h}, k) = \left\{r \in \{\mathsf{a}, \mathsf{h}\}^* \mid \#_{\mathsf{h}}(r) = 1 \land \#_{\mathsf{a}}(r) = k\right\},$$

$$R_0(\mathsf{H}, k) = \left\{r \in \{\mathsf{a}, \mathsf{h}, \mathsf{H}\}^* \mid \#_{\mathsf{a}}(r) = k \land (\#_{\mathsf{h}}(r) \geq 2 \lor \#_{\mathsf{H}}(r) \geq 1)\right\}.$$

Moreover, we naturally extend the mapping $R_0(\cdot)$ to strings $w = w_1 \dots w_n \in \Sigma_\infty^*$ by the convention

$$R_0(w) \triangleq R_0(w_1) \circ \cdots \circ R_0(w_n) \subseteq \Sigma_{\mathsf{mh}}^*.$$

Lemma 3.3. *Let $w \in \Sigma_\infty^n$ and $F \vdash w$. Then there is a characteristic string $w' \in R_0(w)$ and a tree $F' \vdash w'$ such that $F' \equiv F$.*

Proof. Consider the fragment of a PoW-tree

$$F \vdash w = w_1 \dots w_n \in \Sigma_\infty^n$$

induced by vertices attributed to a particular symbol $w_i \in \Sigma_\infty$. This is a (potentially disconnected) forest of trees. (The word "tree" here and throughout this proof is used in its standard graph-theoretic sense, as opposed to referring to a PoW tree.) Partitioning this forest according to depth—as measured in the original tree F—we write the vertices of the forest as a disjoint union $V_d \cup \cdots \cup V_D$, where d is the smallest depth appearing in the forest, D is the largest depth, and V_j contains those vertices of depth j. Now associate with each V_j the string

$$w^{(j)} = \begin{cases} \mathsf{a}^k & \text{if } V_j \text{ contains no honest vertices,} \\ \mathsf{h}\mathsf{a}^k & \text{if } V_j \text{ contains one honest vertex,} \\ \mathsf{H}\mathsf{a}^k & \text{if } V_j \text{ contains multiple honest vertices,} \end{cases}$$

where k is the number of adversarial vertices appearing in V_j. By construction, there is a straightforward labeling of each set V_j by the string $w^{(j)}$ that maintains the classification of vertices as adversarial or honest and satisfies axioms (A2) and (A3). Finally, let $w'_i = w^{(d)} \dots w^{(D)}$. Combining the labelings of each V_j induces a labeling of the trees by the string w'_i that likewise satisfies (A2) and (A3). It follows that F can be (re)labeled by the string $w'_1 \dots w'_n \in R_0(w)$ so as to satisfy all of the PoW tree axioms; this relabeling determines the PoW tree F', as desired. □

Lemma 3.4. *Let $w \in \Sigma_\infty^*$ and $w' \in R_0(w)$. Then for any tree $F' \vdash w'$ there exists a tree $F \vdash w$ such that $F \equiv F'$.*

Proof. Let v be a vertex in F' with $\mathsf{l}_\#(v) = j \in [|w'|]$, and let $i \in [|w|]$ be the index in $w = w_1 \dots w_{|w|}$ such that the j-th symbol in w' belongs to the expansion $R_0(w_i)$ of w_i. Then it suffices to set the label of v in F as $\mathsf{l}_\#^F(v) = i$. The correctness of this construction follows directly from the definition of R_0. □

Lemmas 3.3 and 3.4 immediately imply the following corollary.

Corollary 3.5. *Let $w \in \Sigma_\infty^*$. Then*

$$\beta_\ell(w) = \max_{w' \in R_0(w)} \beta_{\ell'}(w'),$$

where ℓ' is the appropriate index in w' corresponding to ℓ in w.

Proof. Let $F \vdash w$ be a witness tree, and let $w^* \in R_0(w)$ and $F^* \equiv F$ be such that $F^* \vdash w^*$ as guaranteed by Lemma 3.3. Let ℓ^* be the appropriate index in w^* corresponding to ℓ in w. We have

$$\beta_\ell(w) = \beta_\ell(F) = \beta_{\ell^*}(F^*) \leq \beta_{\ell^*}(w^*) \leq \max_{w' \in R_0(w)} \beta_{\ell'}(w')$$

where ℓ' is defined as in the statement of the lemma, establishing the first inequality.

For the opposite inequality, let

$$w^* \triangleq \arg \max_{w' \in R_0(w)} \beta_{\ell'}(w')$$

for ℓ' as defined in the statement, let ℓ^* be the respective value for w^*, and let $F^* \vdash w^*$ be its witness tree. Let $F \vdash w$ be the tree satisfying $F \equiv F^*$ as guaranteed by Lemma 3.4. Then

$$\max_{w' \in R_0(w)} \beta_{\ell'}(w') = \beta_{\ell^*}(w^*) = \beta_{\ell^*}(F^*) = \beta_\ell(F) \leq \beta_\ell(w)$$

as desired. □

Now we are ready to establish the main result of this section.

Theorem 3.6. *Fix $\ell \geq 1$. We consider characteristic strings $w \in \Sigma_\infty^* = (\{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N})^*$. By definition $\beta_\ell(\varepsilon) = 0$. We have*

$$\beta_\ell(w(0,a)) = \beta_\ell(w) + a,$$

$$\beta_\ell(w(\mathsf{h},a)) = \begin{cases} \beta_\ell(w) + a, & \text{if } \beta_\ell(w) = 0 \wedge |w| + 1 < \ell, \\ \beta_\ell(w) + a - 1, & \text{otherwise,} \end{cases}$$

$$\beta_\ell(w(\mathsf{H},a)) = \begin{cases} \beta_\ell(w) + a, & \text{if } -a \leq \beta_\ell(w) \leq 0, \\ \beta_\ell(w) + a - 1, & \text{otherwise.} \end{cases}$$

Proof. The statements are shown independently for each case, always applying Corollary 3.5, the definition of the mapping R_0, and Lemma 3.2. Concretely, in the simplest case we have

$$\beta_\ell(w(0,a)) = \max_{w' \in R_0(w(0,a))} \beta_{\ell'}(w') = \max_{w'' \in R_0(w)} \beta_{\ell'}(w''\mathsf{a}^a)$$
$$= \max_{w'' \in R_0(w)} \beta_{\ell'}(w'') + a = \beta_\ell(w) + a \ .$$

The other two cases are fully analogous, additionally taking into account subcases depending on the value of $\beta_\ell(w)$ and ℓ when invoking Lemma 3.2. □

4 THE ANALYSIS WITH DELAYS

We now move our attention to the case of $\Delta = 1$. Contrary to the previous section, we will not derive an exact description of $\beta_\ell^1(w)$; nonetheless, we will define an easy-to-compute recurrent function that we show can give us a good upper-bound on $\beta_\ell^1(w)$.

4.1 Weak Serialization via Deferrals

We start by defining the set $\mathcal{D}_1(w)$ of so-called *deferrals* of w that will play a somewhat similar role in this section as the set of serializations $R_0(w)$ in Section 3. The important difference is that while R_0 partially serialized the block-creation events captured in w, deferrals have a different goal: they account for the possible 1-slot delay of these successes without actually fully serializing them. A deferral is hence still a characteristic string over the rich, unserialized alphabet Σ_∞.

Definition 4.1 (Realizations and deferrals). Consider a characteristic string $w = ((s_1, a_1), \ldots, (s_n, a_n)) \in \Sigma_\infty^n$. A *realization* of w is a string $r = ((h_1, a_1), \ldots, (h_n, a_n)) \in (\mathbb{N} \times \mathbb{N})^n$ where for each $i \in [n]$ we have $s_i = \mathsf{round}_\mathsf{H}(h_i)$. Let

$$r = ((h_1, a_1), \ldots, (h_n, a_n))$$
$$r' = ((h'_1, a'_1), \ldots, (h'_n, a'_n), (h'_{n+1}, a'_{n+1}))$$

be two realizations, where each (h_i, a_i) and (h'_i, a'_i) are elements of $\mathbb{N}^2$. We say that r' is a *1-deferral* of r if

(1) for each $t \in \{0, \ldots, n\}$, $\sum_{i=1}^{t} a_i \leq \sum_{i=1}^{t+1} a'_i \leq \sum_{i=1}^{t+1} a_i$, and
(2) for each $t \in \{0, \ldots, n\}$, $\sum_{i=1}^{t} h_i \leq \sum_{i=1}^{t+1} h'_i \leq \sum_{i=1}^{t+1} h_i$,

where we adopt the convention that $a_{n+1} = h_{n+1} = 0$. Finally, consider two characteristic strings

$$w = ((s_1, a_1), \ldots, (s_n, a_n)) \in \Sigma_\infty^n \ ,$$
$$w' = ((s'_1, a'_1), \ldots, (s'_n, a'_n), (s'_{n+1}, a'_{n+1})) \in \Sigma_\infty^{n+1} \ .$$

We say that w' is a 1-deferral of w if there are realizations r (of w) and r' (of w') so that r' is a 1-deferral of r. Let $\mathcal{D}_1(w)$ denote the set of all 1-deferrals of w. As we only consider 1-deferrals in this work, we sometimes simply call them deferrals.

The following lemma is an analogue of Lemma 3.3, showing that any 1-tree of w can be seen as a 0-tree of some 1-serialization of w. We prove it in the full version [11].

Lemma 4.2. *Let $w \in \Sigma_\infty^n$ and $F \vdash_1 w$. Then there is a 1-deferral $w' \in \mathcal{D}_1(w)$ and an equivalent tree $F' \equiv F$ such that $F' \vdash_0 w'$.*

We can now establish the following lemma, which is again an analogue of Corollary 3.5, and is proven in the full version [11].

Lemma 4.3. *Let $w \in \Sigma_\infty^n$, then*

$$\beta_\ell^1(w) \leq \max_{w' \in \mathcal{D}_1(w)} \beta_{\ell+1}^0(w') + 2 \ .$$

4.2 The Recurrence $\mathbf{B}_\ell(\cdot)$

In this section we define an easily computable recurrent function $\mathbf{B}_\ell$ that we later use to upper-bound β_ℓ of a particular string w. The definition of $\mathbf{B}_\ell$ will be composed of several basic functions that we define first. After that, we give a recursive description of how $\mathbf{B}_\ell$ can be computed using these basic constituent operations.

The basic intuition underlying the computation of $\mathbf{B}_\ell(w)$ is to internally simulate the computation of $\beta_\ell^0(w')$ on all possible deferrals $w' \in \mathcal{D}_1(w)$, as $\beta_\ell^0(w')$ is precisely described in Theorem 3.6.

More concretely, $\mathbf{B}_\ell$ returns a tuple

$$\mathbf{B}_\ell(w) = ((\beta_0, a_0), (\beta_\mathsf{H}, a_\mathsf{H}), (\beta_\mathsf{h}, a_\mathsf{h})) \in (\mathbb{Z} \times \mathbb{N})^3$$

where each pair (β_s, a_s) for $s \in \{0, \mathsf{H}, \mathsf{h}\}$ keeps track of the best (in a well-defined sense detailed below) achievable margin β_s and the number of delayed adversarial successes a_s after processing a deferral of w that: (0) does not produce an honest carry-over from slot $|w|$ to $|w| + 1$; (h) produces a single such honest carry-over; or (H) produces a multi-honest such carry-over. The definition of $\mathbf{B}_\ell$ then describes how to update this tuple $\mathbf{B}_\ell(w)$ to arrive at $\mathbf{B}_\ell(wz)$ for any $z \in \Sigma_\infty$.

Basic operations. For any $(\beta, a, a') \in \mathbb{Z} \times \mathbb{N} \times \mathbb{N}$ we introduce the following functions:

$$\mathsf{NHE}(\beta, a, a') \triangleq (\beta + a, a') \ ,$$
$$\mathsf{HE}(\beta, a, a') \triangleq (\beta + a - 1, a') \ ,$$
$$\mathsf{NO}(\beta, a, a') \triangleq \begin{cases} (\max\{0, \beta + a\}, a' + \min\{0, \beta + a\}) & \\ \qquad\qquad \text{if } \beta \in \{-a - a', \ldots, 0\}, \\ \mathsf{HE}(\beta, a, a') \quad \text{otherwise.} \end{cases} \tag{3}$$

Their names stand for *(no) honest effect* and *neutralization opportunity*, respectively. Intuitively, these functions will be invoked in the update step computing $\mathbf{B}_\ell(wz)$ from $\mathbf{B}_\ell(w)$ with their inputs (β, a) being one of the pairs (β_s, a_s) in $\mathbf{B}_\ell(w)$ for some s, and a' being the number of adversarial successes in the currently processed symbol z. The functions then return a new, updated value pair (β^*, a^*) if (NHE) there was no honest effect on β_ℓ in this round (e.g., no delayed honest success from previous slot and no honest success in this slot either); or (HE) there was an effect of an honest success that decreased β_ℓ by 1; or (NO) there was a neutralization opportunity and whether an honest effect occurred depends on the current running value of β.

Note that which of these basic functions are invoked when computing $\mathbf{B}_\ell(wz)$ from $\mathbf{B}_\ell(w)$ depends on information external to these functions: the honest carry from previous slot (i.e., which s is used to index into the previous tuple $\mathbf{B}_\ell(w)$), the honest success(es) recorded in the current symbol z, and the desired honest carry to the next slot (i.e., which pair of the new value $\mathbf{B}_\ell(wz)$ is being computed). In all cases, these functions are chosen to match the behavior of β^0_ℓ on the respective deferral as described by Theorem 3.6. Looking ahead, this inductive property will be established in Lemma 4.5.

For notational convenience, we also introduce a function HE^t_ℓ that behaves as NO or HE depending on two parameters $\ell, t \in \mathbb{N}$; ℓ will be the usual parameter of β_ℓ and t will be the current slot—HE^t_ℓ will hence be used to distinguish the "pre-ℓ" and "post-ℓ" settings:

$$\mathsf{HE}^t_\ell(\beta, a, a') \triangleq \begin{cases} \mathsf{NO}(\beta, a, a') \text{ if } t < \ell, \\ \mathsf{HE}(\beta, a, a') \text{ if } t \geq \ell. \end{cases}$$

To reason about these basic functions, we introduce a binary relation $\preceq$ on the elements $(\beta, a) \in \mathbb{Z} \times \mathbb{N}$ as follows:

$$(\beta_1, a_1) \preceq (\beta_2, a_2) :\Leftrightarrow [(\beta_1 + a_1 < \beta_2 + a_2) \vee \\ \vee (\beta_1 + a_1 = \beta_2 + a_2 \wedge a_1 \leq a_2)] \ . \quad (4)$$

It is easy to verify that $\preceq$ is in fact a total order on $\mathbb{Z} \times \mathbb{N}$. We use the standard notation $x \prec y$ for $(x \preceq y \wedge x \neq y)$. For convenience, let us define an operator $\max_\prec$ that, given a tuple $\{(x_i, y_i)\}_{i=1}^n$ of pairs from $\mathbb{Z} \times \mathbb{N}$, returns the maximum pair with respect to the total order $\preceq$. Finally, let $\perp$ represent the pair $(-\infty, 0)$; to handle $\perp$ we sometimes abuse the notation and extend $\preceq$ to $(\mathbb{Z} \cup \{-\infty\}) \times \mathbb{N}$ in the natural way. We also sometimes treat $\perp$ as a ternary function (akin to NHE, HE, NO) that always returns $(-\infty, 0)$, which will always be clear from the context.

Formal description of $\mathbf{B}_\ell$. Let $\mathbf{B}_\ell(\varepsilon) \triangleq ((0,0), \perp, \perp)$. Furthermore, if $\mathbf{B}_\ell(w) = ((\beta_0, a_0), (\beta_\mathsf{H}, a_\mathsf{H}), (\beta_\mathsf{h}, a_\mathsf{h}))$ and $|w| + 1 = t$ then

$$\mathbf{B}_\ell(w(0, a')) = (\max_\prec \begin{Bmatrix} \mathsf{NHE}(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{HE}^t_\ell(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}, \perp, \perp) \ ,$$

$$\mathbf{B}_\ell(w(\mathsf{H}, a')) = (\max_\prec \begin{Bmatrix} \mathsf{NO}(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{NO}(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}, \max_\prec \begin{Bmatrix} \mathsf{NHE}(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{NO}(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}, \\ \max_\prec \begin{Bmatrix} \mathsf{NO}(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{NO}(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}) \ ,$$

$$\mathbf{B}_\ell(w(\mathsf{h}, a')) = (\max_\prec \begin{Bmatrix} \mathsf{HE}^t_\ell(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{NO}(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}, \perp, \max_\prec \begin{Bmatrix} \mathsf{NHE}(\beta_0, a_0, a') \\ \mathsf{NO}(\beta_\mathsf{H}, a_\mathsf{H}, a') \\ \mathsf{HE}^t_\ell(\beta_\mathsf{h}, a_\mathsf{h}, a') \end{Bmatrix}) \ .$$

We additionally introduce some notation that allows us to conveniently reason about $\mathbf{B}_\ell$. For some

$$\mathbf{B}_\ell(w) = ((\beta_0, a_0), (\beta_\mathsf{H}, a_\mathsf{H}), (\beta_\mathsf{h}, a_\mathsf{h}))$$

and $s \in \{0, \mathsf{H}, \mathsf{h}\}$ we use the notation $\mathbf{B}_\ell(w)[s]$ to refer to the pair (β_s, a_s) in $\mathbf{B}_\ell(w)$. Moreover, we let

$$B_\ell(w) \triangleq \max_{s \in \{0,\mathsf{H},\mathsf{h}\}} \beta_s + a_s \ .$$

Intuitively, given some $w \in \Sigma^*_\infty$ and $z \in \Sigma_\infty$, the final step of computation of $\mathbf{B}_\ell(wz)$ (processing the trailing symbol $z \in \Sigma_\infty$) can be seen as determined by a three-dimensional table of $3^3 = 27$ cells, each cell specifying a single operation $\mathsf{op} \in \{\mathsf{NHE}, \mathsf{HE}^t_\ell, \mathsf{NO}, \perp\}$ that needs to be applied to $\mathbf{B}_\ell(w)[s_\mathsf{prev}]$ if the "honest carry" from the previous step is s_prev, the honest part of the current symbol z is s_cur (i.e., $z = (s_\mathsf{cur}, a')$), and the desired honest carry to the next slot is s_next; with all $s_\mathsf{prev}, s_\mathsf{cur}, s_\mathsf{next} \in \{0, \mathsf{H}, \mathsf{h}\}$. We sometimes explicitly refer to this operation as $\mathsf{op}_{[s_\mathsf{prev}, s_\mathsf{cur}, s_\mathsf{next}]} \in \{\mathsf{NHE}, \mathsf{HE}^t_\ell, \mathsf{NO}, \perp\}$. For example $\mathsf{op}_{[0,0,0]} \equiv \mathsf{NHE}$, $\mathsf{op}_{[\mathsf{H},0,0]} \equiv \mathsf{NO}$, $\mathsf{op}_{[\mathsf{h},0,0]} \equiv \mathsf{HE}^t_\ell$, $\mathsf{op}_{[0,0,s]} \equiv \perp$ for any $s \in \{0, \mathsf{h}, \mathsf{H}\}$, and so on.

Monotonicity. We conclude this section by stating a simple monotonicity property of all the basic functions NHE, HE, NO and HE^t_ℓ underlying $\mathbf{B}_\ell$. Given partial orders $(S, \prec_S)$ and $(T, \prec_T)$, recall that a function $f\colon S \to T$ is called (weakly) monotone if

$$\forall x, y \in S\colon \ (x \preceq_S y \Rightarrow f(x) \preceq_T f(y)) \ .$$

We defer the proof of the following lemma to the full version [11].

Lemma 4.4. *For any fixed $a' \in \mathbb{N}$ and $t, \ell \geq 1$, the functions $\mathsf{NHE}(\cdot,\cdot,a')$, $\mathsf{HE}(\cdot,\cdot,a')$, $\mathsf{HE}^t_\ell(\cdot,\cdot,a')$ and $\mathsf{NO}(\cdot,\cdot,a')$ mapping $\mathbb{Z} \times \mathbb{N} \to \mathbb{Z} \times \mathbb{N}$ are monotone with respect to the total order $\preceq$ of* (4).

4.3 Upper-bounding Deferral Margin by $\mathbf{B}_\ell(\cdot)$

The following lemma is the key technical result that formalizes the intuition behind the definition of $\mathbf{B}_\ell$. We provide an outline of its proof below, deferring a detailed treatment to the full version [11].

Lemma 4.5. *Let $w \in \Sigma^n_\infty$ and let $w' \in \mathcal{D}_1(w)$. Writing $w' = x'(s'_{n+1}, a'_{n+1})$, so that $x' \in \Sigma^n_\infty$ consists of the first n symbols of w' and $(s'_{n+1}, a'_{n+1}) \in \{0, \mathsf{H}, \mathsf{h}\} \times \mathbb{N}$ is the last symbol. Then we have*

$$\left(\beta^0_\ell(x'), a'_{n+1}\right) \preceq \mathbf{B}_\ell(w)[s'_{n+1}] \ .$$

Proof outline. We proceed by induction on the length $n \in \mathbb{N}$ of $w \in \Sigma^*_\infty$. The base case is straightforward, hence we focus on the induction step.

Let $w \in \Sigma^n_\infty$ and $w' \in \mathcal{D}_1(w)$. Write $w = x(s_n, a_n)$, where x consists of the first $n-1$ symbols of w. We will first construct $\overline{x}'$ that is a deferral of x and shares the first $n-1$ symbols as x'. We observe that w' naturally gives rise to a deferral of x. To describe this, let $r = (h_1, a_1), \ldots, (h_n, a_n)$ and $r' = (h'_1, a'_1), \ldots, (h'_{n+1}, a'_{n+1})$ be realizations of w and w', respectively, for which r' is a deferral of r. Letting q denote the first $n-1$ symbols of the realization r, it's clear that q is a realization of x. Then we observe that an adaptation of the suffix of r' (and w') yields a deferral of x (the prefix of w). Specifically, defining $(\overline{h}'_n, \overline{a}'_n) = (h'_n, a'_n) + (h'_{n+1}, a'_{n+1}) - (h_n, a_n)$ (where arithmetic is coordinatewise) it is easy to confirm that $\overline{q}' \triangleq (h'_1, a'_1), \ldots, (h'_{n-1}, a'_{n-1}), (\overline{h}'_n, \overline{a}'_n)$ is a deferral of the realization q.

To reiterate and organize the notation, we arrange these in a table, where we use the notation $w \leftarrow r$ to indicate that r is a realization of the string w, and $w \rightsquigarrow w'$ to indicate that w' is a 1-deferral of w.

$$\begin{aligned} & w \leftarrow r = (h_1, a_1), \ldots, (h_n, a_n) \\ & \wr\!\wr \\ & w' \leftarrow r' = (h'_1, a'_1), \ldots, (h'_n, a'_n), (h'_{n+1}, a'_{n+1}) \\ \\ & x \leftarrow q = (h_1, a_1), \ldots, (h_{n-1}, a_{n-1}) \\ & \wr\!\wr \\ & \overline{x}' \leftarrow \overline{q}' = (h'_1, a'_1), \ldots, (h'_{n-1}, a'_{n-1}), (\overline{h}'_n, \overline{a}'_n) \end{aligned}$$

Let z' be the $(n-1)$-prefix of w' (or x') and let $\bar{s}'_n \in \{0, \mathsf{h}, \mathsf{H}\}$ be the "rounded" version of $\overline{h}'_n$, i.e., $\bar{s}'_n \triangleq \mathsf{round}_{\mathsf{H}}(\overline{h}'_n)$. By induction hypothesis we have $(\beta^0_\ell(z'), \bar{a}'_n) \preceq \mathbf{B}_\ell(x)[\bar{s}'_n]$.

The inductive step of the argument is now established in a sequence of manipulations that respect the ordering $\succeq$. Namely, we will prove that

$$\begin{aligned}\mathbf{B}_\ell(w)[s'_{n+1}] &\overset{(a)}{=} \max_{\prec}\left\{\begin{array}{l}\mathsf{op}_{[0,s_n,s'_{n+1}]}\,(\mathbf{B}_\ell(x)[0], a_n)\\ \mathsf{op}_{[\mathsf{H},s_n,s'_{n+1}]}\,(\mathbf{B}_\ell(x)[\mathsf{H}], a_n)\\ \mathsf{op}_{[\mathsf{h},s_n,s'_{n+1}]}\,(\mathbf{B}_\ell(x)[\mathsf{h}], a_n)\end{array}\right\}\\ &\overset{(b)}{\succeq} \mathsf{op}_{[\bar{s}_n,s_n,s'_{n+1}]}\left(\mathbf{B}_\ell(x)\left[\bar{s}'_n\right], a_n\right) \qquad (5)\\ &\overset{(c)}{\succeq} \mathsf{op}_{[\bar{s}_n,s_n,s'_{n+1}]}\left(\beta^0_\ell(z'), \bar{a}'_n, a_n\right)\\ &\overset{(d)}{=} \left(\beta^0_\ell(x^*), a^*\right) \overset{(e)}{\succeq} \left(\beta^0_\ell(x'), a'_{n+1}\right)\ ,\end{aligned}$$

where x^*, a^* are simple modifications of x', a'_{n+1} that we precisely define. Note that establishing (5) concludes the inductive step and hence also the whole proof of the lemma.

Equation (a) follows from the definition of $\mathbf{B}_\ell$, recall that $\mathsf{op}_{[s,s_n,s'_{n+1}]}$ is the operation that is used in the computation of $\mathbf{B}_\ell$ in the cell where the honest carry from previous slot is s, the honest part of the symbol in the current slot is s_n, and the desired honest carry to the next slot is s'_{n+1}. Step (b) then follows by definition of $\max_\prec$. Step (c) is a direct application of the induction hypothesis and the monotonicity of $\mathsf{op}_{[\bar{s}_n,s_n,s'_{n+1}]}$ with respect to its first two inputs, as established in Lemma 4.4.

The main effort in establishing the induction case lies in verifying the other part of step (d), namely, the first component of $\mathsf{op}_{[\bar{s}_n,s_n,s'_{n+1}]}\left(\beta^0_\ell(z'), \bar{a}'_n, a_n\right)$ being equal to $\beta^0_\ell(x^*)$. This amounts to verifying that, intuitively, the operation performed in the cell of the definition of $\mathbf{B}_\ell$ determined by $(\bar{s}_n, s_n, s'_{n+1})$ is identical to how $\beta^0_\ell(x^*) = \beta^0_\ell(z'x^*_n)$ evolves from $\beta^0_\ell(z')$ when processing the last symbol x^*_n of x^*. Luckily, this behavior of β^0_ℓ is exactly described by Theorem 3.6, and hence this claim can be verified by a straightforward case analysis considering each of the cells separately and comparing it to the behavior guaranteed by Theorem 3.6. Finally, establishing (e) turns out to be easy. □

Given Lemma 4.5, we can now establish our main result.

Theorem 4.6. *Let $w \in \Sigma^*_\infty$. Then $\beta^1_\ell(w) \leq B_{\ell+1}(w) + 2$.*

Proof. First, Lemma 4.3 gives us

$$\beta^1_\ell(w) \leq \max_{w' \in \mathcal{D}_1(w)} \beta^0_{\ell+1}(w') + 2\ .$$

Let $w^* \in \mathcal{D}_1(w)$ be the 1-deferral of w that maximizes $\beta^0_{\ell+1}(\cdot)$ above, and as before let $w^* = x^*(s^*_{n+1}, a^*_{n+1})$ with $x^* \in \Sigma^n_\infty$ and $(s^*_{n+1}, a^*_{n+1}) \in \{0, \mathsf{H}, \mathsf{h}\} \times \mathbb{N}$. Let

$$\mathbf{B}_{\ell+1}(w) = ((\beta_0, a_0), (\beta_\mathsf{H}, a_\mathsf{H}), (\beta_\mathsf{h}, a_\mathsf{h}))\ ,$$

then we have

$$\beta^0_{\ell+1}(w^*) \overset{(a)}{\leq} \beta^0_{\ell+1}(x^*) + a^*_{n+1} \overset{(b)}{\leq} \max_{s \in \{0,\mathsf{H},\mathsf{h}\}} \beta_s + a_s = B_{\ell+1}(w)$$

as desired, where inequality (a) follows from Theorem 3.6, and inequality (b) is a direct consequence of Lemma 4.5. □

Finally, we remark that for characteristic strings of the special form $w = w'(0,0)$, i.e. terminating with a success-free slot, we clearly have $\beta^1_\ell(w) = \beta^0_\ell(w)$, this leads to a stronger statement without the additional additive term +2 for this special case.

5 EXPLICIT BOUNDS

In this section, we study explicit bounds provided by our analysis. As described, we are interested in the setting where honest and adversarial block production are determined by Poisson processes with parameters r_h and r_a, while network delay of block delivery is upper bounded Δ_r time.

We collect results for both a Bitcoin-like system—with 600 second inter-block time corresponding to a 1/600 rate Poisson process—and an Ethereum-like system—with 13 second inter-block periods corresponding to a 1/13 rate process. The 90th percentile block propagation time for Bitcoin (resp. Ethereum) has been measured to be around 4 seconds [18] (resp. around 2 seconds [7], partly due to smaller block sizes); we will use these values as the values of Δ_r in the respective settings. To provide more data that are directly comparable with a previous work [17], we will also give results for a 10 seconds delay bound for Bitcoin and a 5 seconds delay bound for Ethereum.

5.1 Numerical Evaluation of the Upper Bounds

The distribution of the characteristic string w is as follows. Each symbol $w_i = (s_i, a_i) \in \{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N}$ is independent and: (i) a_i follows a Poisson distribution with parameter $r_a\Delta_\mathsf{r}$, and (ii) s_i is determined by a Poisson random variable X with parameter $r_h\Delta_\mathsf{r}$ so that $s_i = \mathsf{round}_\mathsf{H}(X)$ (refer to (1)). Let $D(r_a, r_h, \Delta_\mathsf{r}; n)$ denote the distribution on $(\{0, \mathsf{h}, \mathsf{H}\} \times \mathbb{N})^n$ given by this rule.

Temporal settlement rules. Examining the conclusions of the previous section and, in particular, the recursive description of the tuple $\mathbf{B}_\ell$, it is clear that one can efficiently determine the value $\mathbf{B}_\ell(w)$ for any particular characteristic string w. Furthermore, considering that the distribution $D(r_a, r_h, \Delta_\mathsf{r}; n)$ calls for independent symbols, it is straightforward to determine the exact distribution of $\mathbf{B}_\ell(wa)$, where a is an additional independent symbol, from that of $\mathbf{B}_\ell(w)$. Specifically, we consider a "six-dimensional" table T_n, with one cell for each possible value of $\mathbf{B}_\ell$ (thus a value has the form $(\beta_0, a_0, \beta_\mathsf{h}, a_\mathsf{h}, \beta_\mathsf{H}, a_\mathsf{H})$), whose cells are populated with the probabilities that this value emerges in $\mathbf{B}_\ell(w)$ (with w drawn from $D(r_a, r_h, \Delta_\mathsf{r}; n)$). Given the "kernel" distribution for the next symbol a, each cell of the corresponding table T_{n+1} can be determined as an appropriate convex combination of the entries in T_n with the kernel distribution. The symbol distribution has infinite support; however, the Poisson distribution decays very rapidly, allowing us to use finite approximations that suitably control errors.

Initially, we must settle on a distribution of $\mathbf{B}_\ell$ at time ℓ (corresponding to the moment in time when the transaction of interest was submitted to the blockchain). While this does depend on ℓ, the distribution converges quickly to an exponentially decaying distribution (in the sense that the entries are $\exp(-\lambda(\beta_0 + a_0))$). For this reason, rather than selecting some particular ℓ in our numerical evaluation, we choose a very large ℓ that corresponds to the steady state of the blockchain. Specifically, we select a large enough ℓ so that the difference in total variation observed by evolving for an additional step is bounded by 10^{-5}. (Intuitively, this initial distribution reflects the number of private blocks that the adversary may have, along with any deferred honest blocks from slot ℓ.)

For simplicity, we append a concluding $(0,0)$ onto the end of the generated characteristic string which, recalling the semantics of $\mathbf{B}_\ell$, permits us to focus on a single pair, (β_0, α_0); as the string does not terminate with any honest victories, we may neglect the $+2$ of Theorem 4.6 and the event of interest is simply $\beta_0+\alpha_0 \geq 0$, in which case the adversary can launch a successful "double spend" attack. Note that this postpended $(0,0)$ in fact corresponds to an observable event–it can be guaranteed by witnessing a "quiet" region of length $2\Delta_r$. Finally, we compute the probability that the margin should *ever* climb above zero after our threshold of interest, by continuing to evolve the probability forward in time, but effectively "freezing" any probability mass on positive values of margin. We then evolve the system forward until the (exponentially decaying) contributions from further evolution are negligible.

Block-based settlement rules. We also consider the settlement rule that is actually used in Bitcoin "Wait for the transaction to be buried by k blocks." This requires a small adaptation to the framework above because an individual symbol may generate multiple blocks: in this case, one maintains a graded data structure that reflects the probabilities conditioned on observing a particular total number of block-creation events. A further complication arises in the interpretation of margin for this stopping time. In particular, this stopping time is quite different from the simple stopping time "wait for k block creation events," which is not even an observable event. For example, note that if $\beta_\ell()$ is $2k$ at time ℓ, an adversary can immediately activate the settlement of "buried by k blocks" and can double spend. Observe that if $\beta_\ell(w) = s$ at time ℓ (so that $|w| = \ell$), then at least $2k - s$ block creation events must take place in order for the adversary to successfully create a double spend (which will expose $2k$ blocks to the observer). With this observation in place, we carry out the natural numerical evolution, conditioned on the value of β arising at $w = \ell$. (We specifically use $\beta_0 + \alpha_0$.)

5.2 Lower Bounds from Private Mining Attacks

We obtain lower bounds on the consistency failure probability by analyzing the well known private mining attack strategy. This attack strategy simply attempts to build a competing chain in private that tries to double spend a target transaction. If there ever comes a point in time after the target transaction has been settled, that the adversary's private chain becomes longer than the public honest chain, the adversary releases its private chain and the private mining attack succeeds. If such a time never occurs, the private mining attack fails. In more detail, the private mining attack consists of two stages. Before time ℓ (the time when the target transaction appears in the system), the attacker tries to build a longer private chain: if its private chain is longer than or equally long as the public chain, it tries to extend its private chain; however, if its private chain is overtaken by the public chain, it gives its private chain, and tries to mine a new private chain from the tip of the public chain. After time ℓ, the attacker goes all-in and keeps mining on the private chain that double spends the target transaction.

We calculate the success probability of the private mining attack in the lock-step model $\mathcal{D}[0, \Delta_r]$ and assuming that neutralization

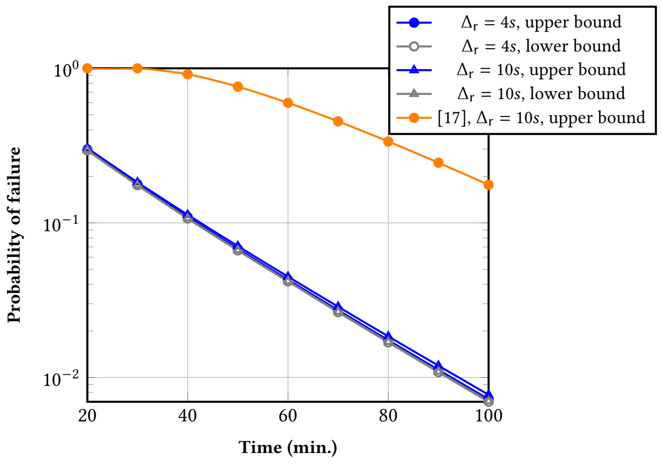

Figure 4: Bitcoin temporal settlement failure for a 10% adversary, results from [17] for comparison.

never occurs. These assumptions weaken the capabilities of the adversary. Note that the private mining attack is *not* optimal when the network delay is non-zero, as shown in [6]. Hence, computing the success probability of this strategy with a weakened adversary gives a lower bound on the probability of consistency failures.

We also provide simulation results for the success probability of the private mining attack in the actual continuous time model $C[\Delta_r]$. For each parameter setting, we run the private mining attack 10000 times in 10 experiments and then plot its success rate with one standard deviation. Since the simulation results account for neutralization, they give better (but noisy) lower bounds. Since our upper and lower bounds already match closely for Bitcoin parameters, we only carry out simulation for Ethereum parameters. We also remark that we can only provide simulation results when the settlement failure probability is relatively high (i.e., short confirmation time or few confirmation blocks); when the settlement failure probability is extremely small, we would have needed a very large number of simulation runs to make reasonable estimates.

5.3 Results

Figures 4 and 5 give our results for temporal settlement in Bitcoin and Ethereum, respectively. These figures depict both lower bounds and upper bounds on the settlement error as a function of time. More results for temporal settlement are given in Table 1. Results for the block-based settlement rule are summarized in Figures 6 and 7, and a more detailed record is given in Table 2.

Figure 4, in particular, clearly shows that our method obtains highly accurate settlement times for the temporal settlement rule for Bitcoin. To elaborate (and as mentioned earlier in the paper), our upper and lower bounds are merely minutes away. For example, for Bitcoin with $\Delta_r = 10s$ delays and a 10% adversary, settlement error probability at the one-hour mark is at most 4.489% (from the upper bound computed in 1-deferral setting), while 90 seconds before that, the settlement error probability is at least 4.494% (due to the lower bound given by private mining attack). (These results are not included in the provided tables, but are obtained using the methods we described in this section.)

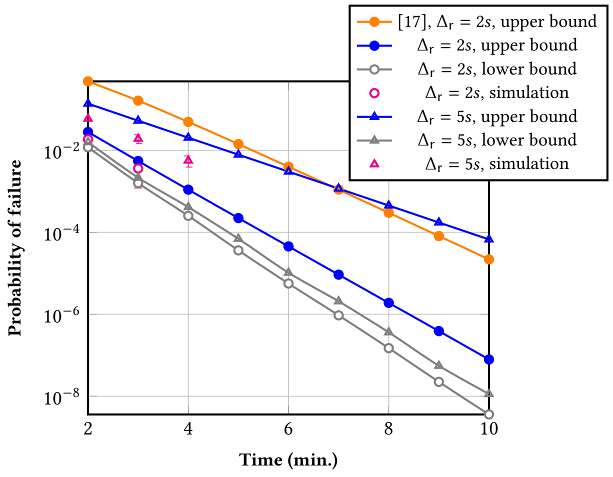

Figure 5: Ethereum temporal settlement failure for a 10% adversary, results from [17] for comparison.

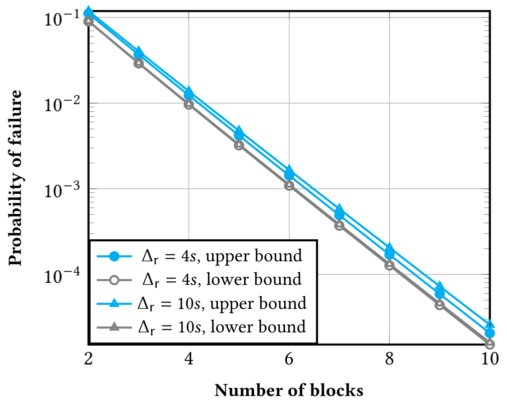

Figure 6: Bitcoin block-based settlement failure for a 10% adversary.

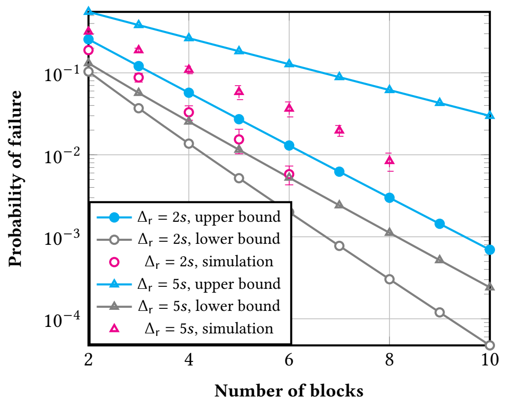

Figure 7: Ethereum block-based settlement failure for a 10% adversary.

Towards comparing with prior art [17], we plot the upper bound results from [17] in Figures 4 and 5. As an example, their method concludes that for a 10% adversary and $\Delta_r = 10s$, a Bitcoin block is settled with at most 0.1% error probability after 5 hours 20 minutes, while our new results bound it within 2 hours and 30 minutes.

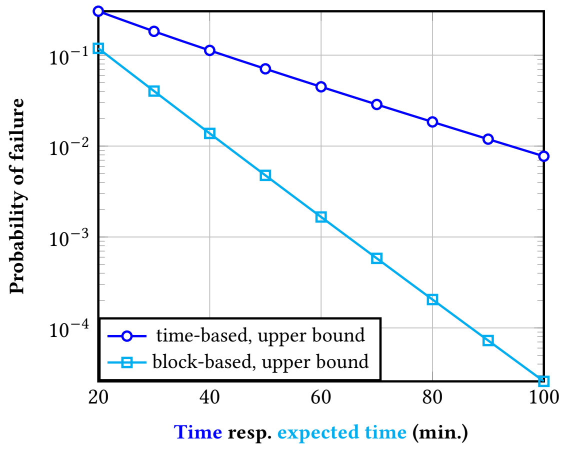

Figure 8: Bitcoin settlement method comparison: time- vs. block-based, for a 10% adversary and $\Delta_r = 10s$.

Furthermore, our new results are no more than 2 minutes and 30 seconds away from the optimum. The comparison is similarly favorable to our results for the Ethereum parametrization.

Comparing settlement modes. It is interesting to compare the settlement error probability for the temporal rule and block-based rule. At the first glance, one may intuitively feel that "waiting for 6 blocks" should provide similar consistency guarantees as "waiting for 60 minutes." We give this comparison for Bitcoin in Figure 8 where we plot the upper bounds on the temporal settlement error as a function of time (as indicated by Figure 4), alongside with the upper bound on the block-based settlement error (as indicated by Figure 6) as a function of the expected time it takes for the particular number of blocks to appear under honest operation. As the graph illustrates, in the above-mentioned case of 60 minutes vs. 6 blocks, the block-based settlement guarantees are an order of magnitude better. This illustrates that under normal operation of the protocol, users are able to arrive at their desired settlement guarantee significantly faster if they apply a block-based settlement rule. This is because the block-based settlement rule naturally adapts to adversarial behavior in the sense that withholding adversarial blocks, in general, will cause the users to wait for longer before observing the requisite number of blocks.

6 CONCLUSIONS

Practical relevance. This work aims to provide concrete settlement-delay advice to all deployed PoW blockchains. Indeed, while the concrete results we quote are parametrized for Bitcoin and Ethereum, the two currently dominant PoW deployments, our methods can be directly applied to compute these statistics for any other choice of block interval, block propagation delay Δ_r, and assumed adversarial share of mining power. The value Δ_r can be estimated based on measurements, as we've done in Section 5.1 using existing work. Estimating the fraction of adversarial mining power is more difficult and ultimately comes down to each user's belief. Nonetheless, our results allow each individual user to choose their settlement times or blocks based on their own beliefs about the system and their acceptable failure probability (perhaps depending on the transacted amount).

Time	upper bounds		lower bounds	
(min)	$\Delta_r = 10s$	$\Delta_r = 4s$	$\Delta_r = 10s$	$\Delta_r = 4s$
	Bitcoin; 10% adversary			
20	0.304519	0.298281	0.295002289	0.294517617
30	0.182942	0.177858	0.175313516	0.175442853
40	0.112861	0.109011	0.10716052	0.106774039
50	0.0707863	0.0679664	0.066656956	0.066352072
60	0.0448913	0.0428636	0.041949497	0.041845542
70	0.0286956	0.0272542	0.026621323	0.026448959
80	0.0184528	0.0174364	0.017000505	0.016874481
90	0.011922	0.0112094	0.010910293	0.010819242
100	0.00773178	0.00723447	0.007029835	0.006964646
	Bitcoin; 20% adversary			
20	0.505249	0.498112	0.494926979	0.492990797
30	0.383382	0.376117	0.373064366	0.371644562
40	0.295733	0.288859	0.286104409	0.284040603
50	0.230435	0.224161	0.221742068	0.219810527
60	0.180805	0.175197	0.173106136	0.17163273
70	0.142594	0.137653	0.135862886	0.134299714
80	0.11291	0.1086	0.107077575	0.105704297
90	0.0896948	0.0859628	0.084675044	0.083480911
100	0.0714434	0.0682312	0.067145976	0.066115746

Time	upper bounds		lower bounds	
(min)	$\Delta_r = 5s$	$\Delta_r = 2s$	$\Delta_r = 5s$	$\Delta_r = 2s$
	Ethereum; 10% adversary			
2	0.137626	0.0279521	0.015828578	0.011812983
3	0.0527935	0.00548293	0.002145191	0.001584263
4	0.0203159	0.0010971	0.000410932	0.000251815
5	0.00782799	0.000221883	6.9340E-05	3.615E-05
6	0.003018	4.51668e-05	1.0273E-05	5.61563E-06
7	0.00116389	9.23251e-06	2.0634E-06	9.38321E-07
8	0.00044892	1.89193e-06	3.6112E-07	1.48653E-07
9	0.000173164	3.87677e-07	5.5071E-08	2.23033E-08
10	6.67978e-05	7.87459e-08	1.1272E-08	3.57683E-09
	Ethereum; 20% adversary			
2	0.384056	0.156394	0.117232788	0.092808469
3	0.245871	0.0697603	0.043623877	0.033041265
4	0.158287	0.0317031	0.019425505	0.012949768
5	0.102233	0.0145709	0.008178393	0.004849916
6	0.0661652	0.00674751	0.003249166	0.001899153
7	0.0428818	0.00314153	0.001500296	0.000774062
8	0.0278188	0.00146854	0.000650086	0.00030802
9	0.0180596	0.000688627	0.000264363	0.000119565
10	0.0117302	0.000323706	0.000123975	4.80722E-05

Table 1: The failure probability of the temporal settlement rule for Bitcoin and Ethereum under different settlement time, adversary ratio and network delays.

Confs.	upper bounds		lower bounds	
	$\Delta_r = 10s$	$\Delta_r = 4s$	$\Delta_r = 10s$	$\Delta_r = 4s$
	Bitcoin; 10% adversary			
2	0.118882	0.111154	0.091072133	0.090289244
3	0.0402842	0.0368385	0.029544274	0.029154201
4	0.0137891	0.0123524	0.009793747	0.009616722
5	0.00476516	0.00418514	0.003294434	0.003217994
6	0.00165992	0.00143009	0.001120043	0.00108804
7	0.000582003	0.000492027	0.000383901	0.000370782
8	0.000205151	0.000170222	0.000132434	0.000127138
9	7.2633e-05	5.91573e-05	4.5926E-05	4.38129E-05
10	2.58108e-05	2.06366e-05	1.5996E-05	1.51607E-05
	Bitcoin; 20% adversary			
2	0.466437	0.45271	0.319646859	0.317452323
3	0.288865	0.277594	0.188866365	0.186940123
4	0.177784	0.169269	0.113180954	0.111647534
5	0.109524	0.103348	0.068498618	0.06733905
6	0.0676876	0.0633137	0.041762682	0.040912987
7	0.0419841	0.0389337	0.025608615	0.024999029
8	0.0261309	0.0240265	0.015775657	0.015344919
9	0.016314	0.0148737	0.009755295	0.009454384
10	0.0102126	0.00923299	0.006051757	0.005843401

Confs.	upper bounds		lower bounds	
	$\Delta_r = 5s$	$\Delta_r = 2s$	$\Delta_r = 5s$	$\Delta_r = 2s$
	Ethereum; 10% adversary			
2	0.554298	0.256406	0.13124146	0.103613076
3	0.38244	0.120911	0.056912431	0.037008885
4	0.264554	0.0571909	0.025438212	0.013695484
5	0.183481	0.0271947	0.011522827	0.005185141
6	0.12746	0.0129908	0.005263762	0.001993565
7	0.0886243	0.00622754	0.002419679	0.000774795
8	0.0616519	0.00299308	0.001117841	0.000303492
9	0.0428996	0.00144124	0.000518528	0.000119582
10	0.0298552	0.000694936	0.000241348	4.73342E-05
	Ethereum; 20% adversary			
2	1.03875	0.673397	0.410871826	0.351521687
3	0.889277	0.479654	0.282740403	0.219222634
4	0.749407	0.337735	0.198320599	0.139166224
5	0.628655	0.237452	0.140561678	0.089448203
6	0.526782	0.167208	0.100281703	0.058018531
7	0.441386	0.118012	0.071879557	0.037893876
8	0.369901	0.0834758	0.051707349	0.024883864
9	0.310061	0.0591607	0.037305049	0.016411126
10	0.259949	0.0419967	0.026980289	0.010861279

Table 2: The failure probability of the block-based settlement rule for Bitcoin and Ethereum under different number of confirmations, adversary ratio and network delays.

Future work. The main open question remaining unresolved after our work is to provide analogous practically relevant settlement bounds also for other Nakamoto-style (i.e., longest-chain) ledger consensus protocols, employing different Sybil-protection mechanisms such as proof of stake [5] and proof of space [4]. These are, alongside PoW, also deployed in existing blockchain projects currently carrying billions of dollars in value.

Acknowledgements. This research was supported in part by the National Science Foundation Grants 1801487 and 2143058.

REFERENCES

[1] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2020. Consensus Redux: Distributed Ledgers in the Face of Adversarial Supremacy. Cryptology ePrint Archive, Report 2020/1021. https://eprint.iacr.org/2020/1021.

[2] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 913–930. https://doi.org/10.1145/3243734.3243848

[3] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 585–602.

[4] Bram Cohen and Krzysztof Pietrzak. July, 2019. The Chia Network Blockchain. https://www.chia.net/assets/ChiaGreenPaper.pdf.

[5] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018, Part II (LNCS, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3

[6] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2020. Everything is a Race and Nakamoto Always Wins. In *ACM CCS 20*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 859–878. https://doi.org/10.1145/3372297.3417290

[7] Ethstats. 2021. https://ethstats.net/.

[8] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *FC 2014 (LNCS, Vol. 8437)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, Heidelberg, 436–454. https://doi.org/10.1007/978-3-662-45472-5_28

[9] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 281–310. https://doi.org/10.1007/978-3-662-46803-6_10

[10] Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2020. Tight Consistency Bounds for Bitcoin. In *ACM CCS 20*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 819–838. https://doi.org/10.1145/3372297.3423365

[11] Peter Gaži, Ling Ren, and Alexander Russell. 2021. Practical Settlement Bounds for Proof-of-Work Blockchains. Cryptology ePrint Archive, Paper 2021/805. https://eprint.iacr.org/2021/805 https://eprint.iacr.org/2021/805.

[12] Dongning Guo and Ling Ren. 2022. Bitcoin's Latency–Security Analysis Made Simple. *arXiv preprint arXiv:2203.06357* (2022).

[13] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO 2017, Part I (LNCS, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[14] Lucianna Kiffer, Rajmohan Rajaraman, and Shelat Abhi. 2018. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 729–744.

[15] Leslie Lamport. 2019. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*. 277–317.

[16] Jing Li and Dongning Guo. 2019. On Analysis of the Bitcoin and Prism Backbone Protocols in Synchronous Networks. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 17–24.

[17] Jing Li, Dongning Guo, and Ling Ren. 2020. Close Latency–Security Trade-off for the Nakamoto Consensus. *arXiv preprint arXiv:2011.14051* (2020).

[18] DSN Bitcoin Monitoring. 2021. https://dsn.tm.kit.edu/bitcoin/.

[19] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008).

[20] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 643–673.

[21] Ling Ren. 2019. Analysis of Nakamoto Consensus. Cryptology ePrint Archive, Report 2019/943. https://eprint.iacr.org/2019/943.

[22] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[23] Jun Zhao, Jing Tang, Zengxiang Li, Huaxiong Wang, Kwok-Yan Lam, and Kaiping Xue. 2020. An analysis of blockchain consistency in asynchronous networks: Deriving a neat bound. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 179–189.