# Longest Chain Consensus Under Bandwidth Constraint

Joachim Neu
jneu@stanford.edu

Srivatsan Sridhar
svatsan@stanford.edu

Lei Yang
leiy@csail.mit.edu

David Tse
dntse@stanford.edu

Mohammad Alizadeh
alizadeh@csail.mit.edu

## ABSTRACT

Spamming attacks are a serious concern for consensus protocols, as witnessed by recent outages of a major blockchain, Solana. They cause congestion and excessive message delays in a *real* network due to its bandwidth constraints. In contrast, longest chain (LC), an important family of consensus protocols, has previously only been proven secure assuming an *idealized* network model in which all messages are delivered within bounded delay. This model-reality mismatch is further aggravated for Proof-of-Stake (PoS) LC where the adversary can spam the network with equivocating blocks. Hence, we extend the network model to capture bandwidth constraints, under which nodes now need to choose carefully which blocks to spend their limited download budget on. To illustrate this point, we show that 'download along the longest header chain', a natural download rule for Proof-of-Work (PoW) LC, is insecure for PoS LC. We propose a simple rule 'download towards the freshest block', formalize two common heuristics 'not downloading equivocations' and 'blocklisting', and prove in a unified framework that PoS LC with any one of these download rules is secure in bandwidth-constrained networks. In experiments, we validate our claims and showcase the behavior of these download rules under attack. By composing multiple instances of a PoS LC protocol with a suitable download rule in parallel, we obtain a PoS consensus protocol that achieves a constant fraction of the network's throughput limit even under worst-case adversarial strategies.

## 1 INTRODUCTION

*Consensus.* In the state machine replication (SMR) formulation of the consensus problem, a group of *nodes* aim to order *transactions* received from the environment into a common *ledger*. For this purpose, nodes exchange messages and perform computations as prescribed by the consensus protocol. Consensus is made nontrivial by an adversary who has some control over message delays, controls a certain fraction of nodes, and can cause them to deviate from the protocol in an arbitrary (*Byzantine*) manner in a concerted effort to disturb consensus. *Secure* consensus is achieved if the resulting transaction ledgers across different honest nodes and points in time are *consistent* so that it is meaningful to speak of *the* single common ledger (which is *safe*), and if that ledger is *live* in the sense that every transaction gets assigned a position in the ledger soon after it is input to honest nodes for the first time.

*Nakamoto's Longest Chain Protocol.* In the seminal Bitcoin whitepaper [41], Satoshi Nakamoto describes the *longest chain* (LC) consensus protocol. In this protocol, honest nodes broadcast blocks to each other. A block contains a list of transactions, a nonce, and a reference to a parent block, resulting in chains of blocks up to a root genesis block that is common knowledge. A block is *valid* if a cryptographic hash of it is smaller than a certain fixed threshold, and if the transactions it contains have been legitimized by the owners of the affected assets and are consistent with respect to transactions preceding it as ordered in the same block and its ancestor blocks. Every node adds valid blocks it receives to its local copy of the block tree. Nodes also aim to produce new blocks. For this purpose they bundle recently received transactions together with a reference to the block at the tip of the longest chain in their local block tree and use brute force search to determine a nonce such that the resulting block is valid (*i.e.*, the hash inequality is satisfied). Newfound valid blocks are broadcast to other nodes, completing the process. Each node outputs as ledger the transactions as ordered in the prefix of the block that is $k$-deep in the longest chain of its local block tree.

Besides being remarkably simple, Nakamoto's LC consensus protocol has two outstanding properties. First, it enables consensus in a *permissionless* setting by using *proof-of-work* (PoW) puzzles as a Sybil resistance mechanism [21, 33]. The bottleneck to block production is finding nonces which lead to valid blocks which satisfy the hash inequality, and as long as the majority of hash power at every point in time is controlled by honest nodes, honest nodes output a secure ledger [25, 44]. Second, the LC can tolerate *dynamic participation* in the sense that the ledger remains secure even as the total hash power participating in the protocol as well as its distribution among participants varies over time.

*Proof-of-Stake Longest Chain.* A drawback of Nakamoto's PoW LC is the high electricity consumption and as a result a tendency for centralization of nodes at places of relatively low electricity cost. To overcome the drawbacks of PoW LC while retaining its advantages, protocols such as Ouroboros [4, 17, 35] and Sleepy Consensus [16, 45] preserve the operating principle of the LC but replace PoW with *proof-of-stake* (PoS) lotteries, where a party is assigned random block production opportunities in proportion to the amount of stake it holds in the system, effectively substituting 'one CPU, one vote' by 'one coin, one vote'. For this purpose, nodes use synchronized clocks to count time slots of a predetermined duration. For every time slot, nodes evaluate a block production lottery associated with their cryptographic identity. For instance in [4, 17], nodes get to produce a new valid block if the output of a *verifiable random function* (VRF) is below a threshold proportional to the node's stake.

*Proof-of-Stake Longest Chain Under Bandwidth Constraint.* While PoS LC behaves in some aspects similar to PoW LC, it differs drastically in others. For instance, in PoS, block production opportunities can be 'reused' in the sense that when a node is eligible to produce a block in a certain time slot, it can in fact create many equivocating but equally valid blocks for the same time slot, each potentially with a different set of transactions and/or attached to a different

JN, SS and LY contributed equally and are listed alphabetically.

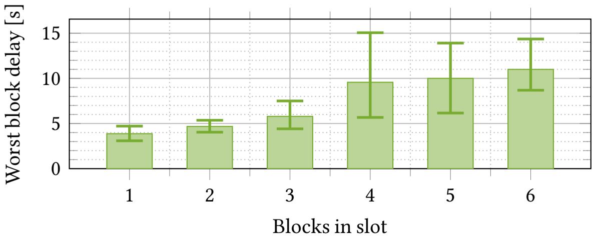

**Figure 1: Time taken (10-th percentile, mean, 90-th percentile) for all nodes to download all blocks mined in a slot, when different number of new blocks are produced and broadcast in a slot. The delay increases as the number of blocks is increased, showing that network delay is not independent of network load. We use Cardano's Ouroboros implementation. Details of the experimental setup are given in Appendix B.1.**

parent block. This problem arises because block production 'lottery tickets' in PoS can not depend on the proposed block's transactions. Otherwise an adversary could increase its chances to produce a block by trying various sets of transactions (*grinding*). Similarly, the PoS lotteries can not depend on the parent block, as the adversary could extend several chains at once to increase their chance of block production (*nothing-at-stake* attack [5]). In PoW however, each block production opportunity corresponds to a unique block (a combination of transaction set, parent block, and nonce), thus the rate of block production opportunities simultaneously bounds the rate at which new valid blocks can be created.

Previous analysis [17, 20, 45] shows that this difference is immaterial in the synchronous network model where the message propagation delay between honest nodes is controlled by the adversary, but below a known upper bound $\Delta$. Under such a network model, PoS LC and PoW LC behave the same in terms of security, transaction throughput and confirmation latency. This model, however, is over-idealized in that it assumes a fixed delay upper bound for every single message, even when many messages are transmitted simultaneously (which may be under normal execution or due to adversarial actions). The model does not capture notions of capacity and congestion which have a significant impact on the behavior of real networks. In fact, an increase in network delay with increasing network load (via increased block size) has been demonstrated previously for Bitcoin [19]. Similarly, increasing the network load (via increasing the number of blocks per slot) leads to increased network delay in our experiments (see Figure 1) with Cardano's Ouroboros implementation—a PoS protocol. Once we enrich the network model to capture such phenomena, the difference in the behavior of PoW LC and PoS LC with respect to reuse of block production opportunities strikes. The possibility of producing (infinitely) many equivocating valid blocks per block production opportunity opens up new adversarial strategies in which the adversary aims to exhaust limited network resources with useless spam in an attempt to disturb consensus. This protrudes in another experiment (see Figure 2) where nodes run PoS LC with our implementation of Cardano's block download logic as per [30]. Adversarial spamming (through block equivocations) causes significant network traffic at the victim nodes, leaving insufficient bandwidth for the victims to download honest blocks. As a result, block production on the honest chain stalls, and the victim node can be easily fooled by a longer chain from the adversary, potentially resulting in a safety violation.

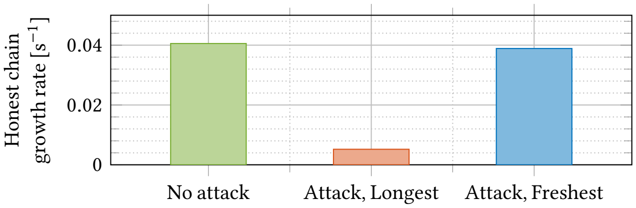

**Figure 2: The honest chain growth rate in three scenarios: without spamming attack; under attack while downloading the longest header chain first (priority rule in Cardano's block download logic); under attack while downloading the freshest block first (introduced in this work). Details of the experimental setup are given in Section 5.2. For a trace of the chain growth in the same experiment, see Figure 5.**

*Modelling Bandwidth Constraints.* We model a bandwidth constrained network as follows. Recall that blocks in Nakamoto consensus consist of a list of transactions as *block content*, and the information pertaining to the PoS/PoW lottery and the block tree structure (reference to parent block) as *block header*. Since a block's header is small compared to its content, we assume that block headers propagate with a known delay upper bound $\Delta_{\mathrm{h}}$ between honest nodes. At any point after obtaining a block header, a node can request the corresponding block content from the network. Since a block's content is large, every honest node can only download a limited number of blocks' contents per time slot. This model is inspired by the peer-to-peer network designs used for blockchain protocols. For instance, in the Cardano network [14, 15], each node advertises its block header chain to its peers, which in turn decide based on the block headers which block contents to fetch. Without a carefully designed *download rule* for the protocol to determine which blocks honest nodes should spend their scarce bandwidth on, we will see that consensus cannot be achieved with PoS LC.

*The 'Download Along The Longest Header Chain' Rule.* Given that in LC, honest nodes extend the longest chain, a natural download rule is 'download along the longest header chain', *i.e.*, based on the block tree structure obtained from block headers, a node identifies the longest (header) chain, and prioritizes downloading the blocks along that chain. Indeed, Bitcoin does exactly that [1]. Cardano's Ouroboros implementation also follows this paradigm in broad strokes [14, 15, 18] for chain selection [32] and block downloads [31]. As long as the block production rate is low relative to the download bandwidth, this (and other rules that ensure that nodes download a block at most once) work well for PoW LC, simply because the number of distinct blocks is limited by the number of block production opportunities.

Unfortunately, as illustrated in Figure 3, this download rule fails for PoS LC in that the resulting protocol is not secure, even if the block production rate is low and the adversary controls a small minority of the stake. The reason is that the adversary can use consecutive adversarial block production opportunities (at $t$ and

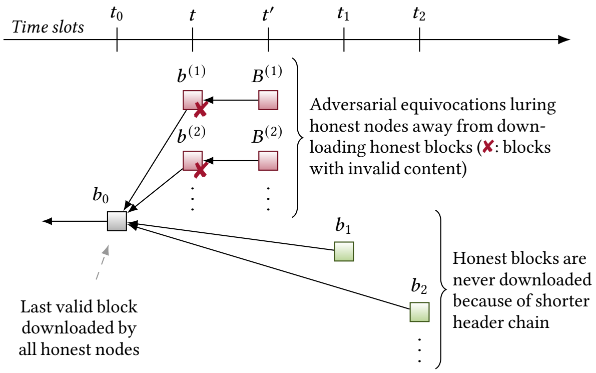
Time slots
$t_0$
$t$
$t'$
$t_1$
$t_2$
$b^{(1)}$
$B^{(1)}$
$b^{(2)}$
$B^{(2)}$
Adversarial equivocations luring honest nodes away from downloading honest blocks (✘: blocks with invalid content)
$b_0$
$b_1$
$b_2$
Honest blocks are never downloaded because of shorter header chain
Last valid block downloaded by all honest nodes

**Figure 3: In PoS LC with 'download along the longest header chain' rule, an adversary can stall consensus indefinitely if it has two consecutive block production opportunities $t < t'$ at which it creates infinitely many equivocating chains $b_0 \leftarrow b^{(i)} \leftarrow B^{(i)}$ where $b^{(i)}$ have invalid content. The blocks of later honest block production opportunities $\ldots > t_2 > t_1 > t' > t$ are never downloaded by other honest nodes, because they prioritize the longer adversarial header chains, wasting their bandwidth downloading each $b^{(i)}$ only to discard it immediately thereafter because of invalid content.**

$t'$ in Figure 3) to produce infinitely many equivocating chains ($b_0 \leftarrow b^{(i)} \leftarrow B^{(i)}$ in Figure 3). To avoid honest nodes building on these equivocating chains, the adversary fills $b^{(i)}$ with invalid content, which honest nodes can only detect after they have already wasted their scarce bandwidth to download it. As a result, honest nodes produce blocks off $b_0$ in their block production opportunities ($b_1, b_2, \ldots$ at $t_1 < t_2 < \ldots$ in Figure 3), but these are never downloaded by other honest nodes because the adversarial header chains are longer and thus of higher download priority. This is clearly an attack on liveness but it also implies an attack on safety because the adversary could now build a chain longer than the honest parties (who are stalled) even though the adversary owns very little stake. The impact of this attack is seen in our experiments with a PoS LC node implementing this download rule (Figure 2).

The above attack suggests that securing PoS LC under bandwidth constraints requires a carefully designed download rule. In practice, protocols follow various heuristics to attempt to mitigate spamming/equivocation attacks. However, a rigorous analysis is usually missing. Our goal in this work is to identify simple download rules that can be proven secure in the bandwidth constrained model.

In the attack in Figure 3, we observe that even though new honest blocks are being proposed, the download rule prioritizes older adversarial equivocating blocks. If honest nodes downloaded the 'fresher' blocks proposed in more recent time slots $t_1, t_2, \ldots$ instead, then this attack would not succeed. This intuition extends beyond the specific attack of Figure 3. We would like that whenever an honest node proposes a block, other honest nodes download that block and its prefix 'soon'. This way, honest nodes have a chance to produce blocks extending it, and to align their block production efforts toward a particular chain. This is arguably the key to LC security and central to prior security analysis [20, 45] on which we build. This insight naturally motivates the following simple download rule.

*The 'Download Towards The Freshest Block' Rule.* We propose a simple download rule for PoS LC, 'download towards the freshest block', *i.e.*, in every time slot an honest node identifies the block proposed in the most recent time slot based on the header information, and downloads any missing blocks in its prefix, including that freshest block. Thus, when an honest node proposes a block, within the same time slot, other honest nodes prioritize downloading that block and its prefix. The length of the prefix cannot be too long since valid chains cannot contain equivocations. By making the time slot long enough to allow downloading the whole prefix, this rule directly satisfies our desired property that honest nodes download honestly proposed blocks 'soon' (within the same time slot). This property is the key step in prior security analysis, thus allowing us to use prior techniques to prove the security of PoS LC with this download rule. In particular, this download rule avoids the attack of Figure 3 and the honest chain's growth rate remains unaffected by this spamming attack (Figure 2). Importantly, note that the freshest block rule is only a download priority rule. Honest nodes still propose blocks extending their longest valid downloaded chain.

*Other Download Rules.* More generally, we identify other download rules with the property that whenever an honest block is proposed, all honest nodes download that block and its prefix 'soon'. Thus, we develop a unified framework to prove security of PoS LC with any download rule with this property. We consider the following two commonly proposed heuristics against equivocations, formalize them and give a rigorous proof of security.

(1) 'Equivocation avoidance': We modify 'download along the longest header chain' such that an honest node refrains from downloading a chain whose tip is an equivocating block header (*i.e.*, it has seen another block header from the same slot and validator). A rule of this kind can be seen used in PoS Ethereum [2].
(2) 'Blocklisting': An honest node avoids downloading any chain whose tip is proposed by a validator that has equivocated before (in its view of block headers). Note that this notion of blocklisting only affects the download priority rule. It does not invalidate a block, as doing so independently at each node risks introducing split views, and doing so consistently would require consensus in the first place.

Due to the simplicity and efficiency of the 'download towards the freshest block' rule, and because it directly satisfies the key property that enables our security proofs, we use this rule as a running example to illustrate our model and the analysis. We then extend this analysis to the other two rules.

*Our Contributions.* By means of experiments and a concrete attack strategy, we show that the bounded delay model fails to capture network congestion and spamming attacks. We show that using suitable download rules, we can provably secure PoS LC in networks with bandwidth constraint in which the adversary can (*inter alia*) spam the network with equivocating blocks at an arbitrary rate, withhold blocks, and release blocks with invalid content that honest nodes discard after downloading. We identify a key property of a download rule that enables it to secure PoS LC. We use this to develop a unified framework to prove the security of PoS LC with any download rule that satisfies this property. We propose a simple rule ‘download towards the freshest block’ that satisfies this property. We also formalize heuristics in the form of the ‘equivocation avoidance’ and ‘blocklisting’ rules for which we provide a rigorous security proof using our framework. We show that parallel composition of multiple instances of PoS LC with a secure download rule (inspired by [24]) yields a consensus protocol that achieves a constant fraction of the network’s throughput limit even in the worst case.

*Related Work.* Network-level attacks on Bitcoin have been studied in [3, 11]. Eclipse attacks on peer-to-peer networks, where an adversary uses several IP addresses to occupy all connections maintained by a victim node and thus cut said node off from the network, have been studied in [28, 49–51] and in the context of Bitcoin in [12]. The authors of [13] show that if one can connect with consensus validators that are pseudo-randomly chosen every few slots based on their stake, then one can secure PoS LC against Sybil attacks and eclipse attacks on the network layer. These earlier works share their focus on network topology, an important aspect not captured by the bounded delay network model. Our work instead focuses on bandwidth constraints, an orthogonal feature of real networks not captured by the bounded delay model. However, our works share the philosophy of co-designing consensus and network layer protocols.

The impact of spamming was seen recently in the temporary shutdown of a PoS protocol Solana [53] on multiple occasions in 2021-2022 [39, 40, 46]. These shutdowns were reportedly due to an increase in the transaction load in the network, and the “lack of prioritization of network-critical messaging caused the network to start forking” [39]. These incidents indicate that messages that are critical for consensus among honest nodes (*e.g.*, blocks) must be appropriately prioritized during periods of congestion. Consensus-critical blocks are easily prioritized at the network level over less critical transaction requests, as the two are different kinds. Thus, this work focuses instead on the design of a download rule with which the consensus protocol assists the network in prioritizing consensus-critical blocks over similarly looking spam blocks.

In practice, implementations show awareness of and attempt to mitigate equivocation-based spamming attacks using various heuristics. However, their efficacy and side effects are often not fully understood. For instance, Cardano’s Ouroboros implementation disconnects from peers once they propagate invalid or equivocating blocks [14, 15, 18]. However, an adversary can boost the impact of its attack by creating more Sybil network peers (recall that there is no relation between consensus validators and peers in the underlying communication network), so that disconnected peers are likely replaced by new adversarial peers, ready to waste more of the honest node’s resources [49–51].

Slashing is routinely proposed as a solution to mitigate spamming with equivocations, as such attacks can be attributed to specific validators [7, 8, 43, 48]. Typical crypto-economic guarantees are of the form “if human intervention is needed to recover from a safety attack, then 33% of stake is slashable“ [9, 43, 48]. However, the attack in Figure 3 only requires two consecutive block production opportunities, which can be obtained by an adversary with a very small fraction of stake. Hence in this case, slashing would impose a very small penalty for an attack that violates security and potentially incurs large costs due to human intervention and other losses. Instead, we take the approach of preventing attacks in the first place by using download rules that are proven secure. Once security is proven, slashing can be employed as an additional measure to disincentivize equivocation-based spamming.

The need for careful modelling of bandwidth constraints in the context of high throughput protocols was identified in [6, 24]. Earlier works [6, 19, 52] note that the network delay increases with the message size (*i.e.*, block size in this case). In this model, it is assumed that as long as the network load is less than the bandwidth, every message is downloaded within a given delay bound which depends on the message size but is independent of total network load.

In the PoS context, [24] captures congestion due to increased network load by modelling the inbox of each node as a queue. Each message undergoes a propagation delay before being added to the recipient’s inbox queue. The recipient can retrieve messages from their queue at a rate limited by their bandwidth, resulting in a queuing delay. However, the security result [24, Theorem 1] still assumes a bounded (propagation+queuing) delay. This assumption is only shown to hold under honest executions when the adversary does not corrupt any nodes and does not send or delay any messages [24, Theorem 3], and therefore the security claim does not hold for all adversarial strategies. In particular, this excludes adversaries that can spam the network using equivocating blocks and cause attacks such as in Figure 3. The model we use is a variant of that in [24] with the difference that nodes can inspect a small segment (block header) at the beginning of every message in their queue and decide based on that which message (block content) to prioritize for download (subject to the bandwidth constraint). This modification allows us to prove security against a general adversary, even with unbounded equivocations.

Although our work is the first to prove PoS LC secure under bandwidth constraints, our analysis builds on tools from several years of security analysis for LC protocols [5, 17, 20, 25, 27, 44, 45, 47], particularly the concept of pivots [45] (cf. Nakamoto blocks [20]).

*Outline.* We state the PoS LC protocol augmented with a download rule and introduce our formal model for bandwidth constrained networks in Section 2. In Section 3, we provide a high-level description of our unified framework for proving security of PoS LC with different download rules under bandwidth constraints. In Section 4, we show the key steps towards this unified proof, and analyze the ‘download towards the freshest block’ rule. We present experimental evidence for the robustness and superior performance of the ‘freshest block’ rule in Section 5. We formalize and analyze the ‘equivocation avoidance’ and ‘blocklisting’ rules in Section 6. Finally, we sketch in Section 7 how to use PoS LC with a suitable download rule as a building block to obtain a consensus protocol with a constant fraction of the network’s throughput limit in the worst case.

## 2 PROTOCOL AND MODEL

*Model Main Features.* For ease of exposition, we assume a static set of $N$ active *nodes*, each with a cryptographic identity corresponding to one unit of stake. Our analysis can be extended to the setting of heterogeneous and dynamic stake using tools from [16, 17]. Nodes’ cryptographic identities are common knowledge. We are interested in the large system regime $N \to \infty$. A *static* adversary $\mathcal{A}$ chooses a set of nodes (up to a fraction $\beta$ of all nodes,

**Algorithm 1** Idealized PoS LC consensus protocol $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$ with a download rule (helper functions: Appendix A.1, $\mathcal{F}^{\rho}_{\text{headertree}}$: Algorithm 3, $\mathcal{Z}$: Appendix A.2)

```
1:  on init(genesisHeaderChain, genesisTxs)
2:      ▷ Initialize header tree $h\mathcal{T}$, longest downloaded chain $dC$, and mapping from block headers to contents (lists of transactions) blkTxs
3:      $h\mathcal{T}, dC \leftarrow$ {genesisHeaderChain}, genesisHeaderChain
4:      blkTxs[$dC$] $\leftarrow$ genesisTxs    ▷ Unset entries of blkTxs are unknown
5:  on receivedHeaderChain($C$)    ▷ Called by $\mathcal{Z}$ or $\mathcal{A}$
6:      assert $\mathcal{F}^{\rho}_{\text{headertree}}$.verify($C$)    ▷ Validate header chain (Algorithm 3)
7:      $h\mathcal{T} \leftarrow h\mathcal{T} \cup$ prefixChainsOf($C$)    ▷ Add $C$ and its prefixes to $h\mathcal{T}$
8:      $\mathcal{Z}$.broadcastHeaderChain($C$)
9:  on receivedContent($C$, txs)    ▷ Called by $\mathcal{Z}$ or $\mathcal{A}$
10:     ▷ Defer processing the content until we received the corresponding header chain $C$, and its prefixes' contents are downloaded and valid
11:     defer until $C \in h\mathcal{T}$
12:     defer until $\forall C' \prec C$: blkTxs[$C'$] $\notin$ {unknown, invalid}
13:     assert $C$.txsHash = Hash(txs)
14:     if txsAreSemanticallyValidWrtPrefixesOf($C$, txs)
15:         blkTxs[$C$] $\leftarrow$ txs
16:         $\mathcal{Z}$.uploadContent($C$, txs)
17:     else
18:         blkTxs[$C$] $\leftarrow$ invalid
19:     ▷ Update the longest downloaded chain among downloaded valid chains
20:     $\mathcal{T}' \leftarrow h\mathcal{T} \setminus \{C' \in h\mathcal{T} \mid$ blkTxs[$C'$] $\in$ {unknown, invalid}$\}$
21:     $dC \leftarrow \arg\max_{C \in \mathcal{T}'} |C|$
22: on scheduleContentDownload()
23:     ▷ Pick next block to download according to download rule (cf. Algs. 2, 4)
24:     if $C \neq \bot$ with $C \leftarrow$ downloadRule($h\mathcal{T}$, blkTxs)
25:         $\mathcal{Z}$.requestContent($C$)
26:     ▷ receivedContent will be triggered by $\mathcal{Z}$ on successful download
27: for time slots $t \leftarrow 1, \ldots, T_{\mathrm{h}}$ of duration $\tau$    ▷ PoS LC protocol main loop
28:     txs $\leftarrow \mathcal{Z}$.receivePendingTxsSemanticallyValidWrt($dC$)
29:     ▷ Produce and disseminate a new block if eligible, see Alg. 3
30:     if $C' \neq \bot$ with $C' \leftarrow \mathcal{F}^{\rho}_{\text{headertree}}$.extend($t$, $dC$, txs)
31:         $\mathcal{Z}$.uploadContent($C'$, txs)
32:         $\mathcal{Z}$.broadcastHeaderChain($C'$)
33:     ▷ Download block contents (starting after $\Delta_{\mathrm{h}}$ time into the slot)
34:     while end of current time slot $t$ not reached
35:         scheduleContentDownload()
36:     $\mathcal{Z}$.outputLedger($dC^{\lceil T_{\mathrm{conf}}}$)    ▷ Ledger of node $i$ at time $t$: $\mathrm{LOG}_i^t$
```

where $\beta$ is common knowledge) to corrupt before the randomness of the protocol is drawn and the execution commences. Uncorrupted *honest* nodes follow the protocol as specified at all times, corrupted *adversarial* nodes deviate from the protocol in an arbitrary *Byzantine* manner coordinated by the adversary in an attempt to inhibit consensus. For simplicity, we have assumed that all nodes are always *awake*. Our analysis builds on techniques from [45] and the refined machinery therein can be used to extend our analysis to the setting of asleep/awake honest nodes.

**Algorithm 2** 'Freshest block' download rule

```
function downloadFreshestBlock($h\mathcal{T}$, blkTxs)
    ▷ Ignore chains with invalid content in any block
    $\mathcal{T} \leftarrow \{C \in h\mathcal{T} \mid \forall C' \preceq C$: blkTxs[$C'$] $\neq$ invalid$\}$
    ▷ Find the chain ending in the freshest block (i.e., from most recent slot)
    $C \leftarrow \arg\max_{C' \in \mathcal{T}} C'$.time
    ▷ Find the first not downloaded block on that chain (if non-existent: $\bot$)
    $C \leftarrow \arg\min_{C' \preceq C:\ \text{blkTxs}[C']=\text{unknown}} |C'|$
    return $C$
```

**Algorithm 3** Idealized functionality $\mathcal{F}^{\rho}_{\text{headertree}}$: block production lottery and header block chain structure (cf. Appendix A.1)

```
on init(genesisHeaderChain, numNodes)
    $N \leftarrow$ numNodes
    $\mathcal{T} \leftarrow$ {genesisHeaderChain}    ▷ Global set of valid header chains
on isLeader($P$, $t$) from $\mathcal{A}$ (only for adversarial $P$) or $\mathcal{F}^{\rho}_{\text{headertree}}$
    ▷ Abstraction of proof-of-stake lottery: each node is chosen leader in each slot with probability $\rho/N$ independent of other nodes and slots
    if lottery[$P$, $t$] = $\bot$
        lottery[$P$, $t$] $\xleftarrow{\$}$ (true with probability $\rho/N$, else false)
    return lottery[$P$, $t$]
on extend($t'$, $C$, txs) from node $P$ (possibly adversarial) at time slot $t$
    ▷ New header chain is valid if parent chain $C$ is valid, $P$ is leader for slot $t'$, and $t'$ is later than the tip of $C$ and is not in the future
    if ($C \in \mathcal{T}$) $\wedge$ isLeader($P$, $t'$) $\wedge$ ($C$.time $< t' \leq t$)
        ▷ Produce a new block header extending $C$
        $C' \leftarrow C \,\|\,$ newBlock(time: $t'$, node: $P$, txsHash: Hash(txs))
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{C'\}$    ▷ Register new header chain in header tree
        return $C'$
    return $\bot$
on verify($C$)
    return $C \in \mathcal{T}$ ▷ Header chain is valid if previously added to header tree
```

*Protocol Main Features.* Pseudocode of an idealized PoS LC Nakamoto consensus protocol parameterized by a download rule is provided in Algorithm 1 (cf. [45, Figure 3]). The 'download towards the freshest block' rule is given in Algorithm 2. Implementation details of the block production lottery and the handling of the blockchain data structure are abstracted away in the idealized functionality $\mathcal{F}^{\rho}_{\text{headertree}}$ provided in Algorithm 3 (cf. [45, Figure 2]). An index of the helper functions used in the pseudocode is provided in Appendix A.1. With specific implementations of $\mathcal{F}^{\rho}_{\text{headertree}}$, a variety of PoS LC protocols can be modelled such as protocols from the Ouroboros family [4, 17, 35] and the Sleepy Consensus [16, 45] family. A more formal description of the environment $\mathcal{Z}$ (idealized functionality modeling the network) is given in Appendix A.1. In the main loop of the PoS LC protocol (Algorithm 1, lines 27ff.) the node attempts in every time slot (which is of duration $\tau$) to produce a new block containing transactions txs and to extend the longest downloaded chain (denoted $dC$) in the node's local view. If successful, the block content txs and the resulting new block header chain $C'$ are provided to the environment $\mathcal{Z}$ for dissemination to all nodes.

*Dissemination of Block Headers and Contents.* The network model and dissemination of headers and contents is illustrated in Figure 4. Block header chains broadcast via $\mathcal{Z}$.broadcastHeaderChain are delivered by the environment $\mathcal{Z}$ to every node with a delay determined by $\mathcal{A}$, up to a delay upper bound $\Delta_{\mathrm{h}}$ that is common knowledge. Once an honest node receives a new valid header chain (Algorithm 1, lines 5ff.), the node adds it to its local header tree $h\mathcal{T}$. Block content uploaded via $\mathcal{Z}$.uploadContent is kept by $\mathcal{Z}$ in an idealized repository. In every time slot, honest nodes use a download rule

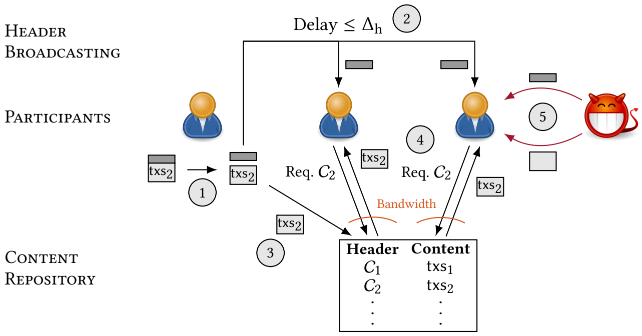

**Figure 4: In our model, block headers are propagated with a known delay upper bound $\Delta_\mathrm{h}$, while block content is subject to a bandwidth constraint. ① An honest node produces a new valid block, consisting of header and content. ② Block headers are broadcast ($\mathcal{Z}$.broadcastHeaderChain) and arrive at honest nodes ($\Pi^{\rho,\tau,T_\mathrm{conf}}$.receivedHeaderChain) within at most $\Delta_\mathrm{h}$ delay. ③ Block content is submitted to an idealized 'repository' ($\mathcal{Z}$.uploadContent). A hash of the corresponding block content is included in the block header. ④ Upon request ($\mathcal{Z}$.requestContent), the content of a certain block is obtained from the 'repository' ($\Pi^{\rho,\tau,T_\mathrm{conf}}$.receivedContent), subject to a constraint on the rate of downloaded block contents. ⑤ An adversary can push block headers and block content to honest nodes independent of delay and bandwidth constraints. See Appendix A.2 for details on $\mathcal{Z}$.**

to select block headers for which they wish to request the content (Algorithm 1, line 22). Honest nodes can request the content for a particular header via $\mathcal{Z}$.requestContent. If available, the content requested from the repository will be delivered by $\mathcal{Z}$ to the honest node by triggering the callback receivedContent (Algorithm 1, lines 9ff.). We set the slot duration as $\tau = \Delta_\mathrm{h} + \frac{K}{C}$ such that all honest nodes receive block headers proposed at the start of the current slot, and thereafter $\mathcal{Z}$ delivers at most $K$ block contents requested from the repository to each honest node per time slot, thereby constraining the bandwidth to $C$ blocks per second.[1] Upon verifying that the content matches the hash in the block header and that the txs are valid with respect to the ledger determined by the block's prefix, the node adds txs to its local view. Otherwise, the block is marked as invalid, to prevent downloading it or any of its descendants in the future. Finally, the node updates its longest downloaded chain.

*'Download Towards The Freshest Block' Rule.* Motivated by the earlier arguments in Section 1, we introduce the 'download towards the freshest block' download rule (Algorithm 2). In this download rule, first the header tree $\mathrm{h}\mathcal{T}$ is pruned by invalid blocks and their descendants. Then, the first unknown block in the prefix of the freshest block is requested. Ties are broken by the adversary.

*Adversarial Strategies And Powers.* Adversarial strategies and powers include but are not limited to: reusing block production opportunities to produce multiple blocks (*equivocations*, by calling $\mathcal{F}^{\rho}_{\text{headertree}}$.extend multiple times, each with a different txs or a different parent chain $C$); extending any chain using past block production opportunities as long as the purported block production time slots along any chain are strictly increasing; releasing block headers late or selectively to honest nodes; proactively pushing block headers or block content to honest nodes irrespective of delay or bandwidth constraints (by triggering the node's respective receivedHeaderChain or receivedContent callback); withholding the content of blocks; including invalid txs in blocks; breaking ties in chain selection and the download rule.

*Reality Check.* Note that in practice the prioritization of blocks according to some download rule does not have to take place only at the endpoints of the network or be limited to block content. Rather, it can also be applied to block headers and by intermediary nodes of the underlying communication or peer-to-peer gossip overlay network as they forward blocks. This effectively shifts the download rule from the edge into the network. Honest participants focus their resources on what the scheduling logic determines as 'high importance' traffic, and save it from being drowned out by adversarial spam. The result is that headers of the blocks which might be of interest to an honest node based on the prioritization stipulated by the download rule will be made available to that honest node by the network within reasonable delay despite adversarial interference. Because of this, we believe that our model leads to protocols that can fare well under bandwidth constraints and spamming in practice.

Various constructions are used to realize $\mathcal{F}^{\rho}_{\text{headertree}}$ in real-world protocols, depending on the desired properties. The block production lottery (Algorithm 3, line 7) is typically implemented by checking whether the output of a random function is below a certain threshold. Against static adversaries, a collision resistant hash function suffices [45]; against adaptive adversaries, a verifiable random function (VRF) is used [35]. Although the ideal functionality $\mathcal{F}^{\rho}_{\text{headertree}}$ relies on the knowledge of $N$ to tune the threshold $\rho/N$, in PoS realizations such as in [17] the factor $1/N$ is replaced by the fraction of the total stake owned by the node as per the confirmed segment of the blockchain.[2] The binding between a block and the production opportunity it stems from (Algorithm 3, line 9) is established using digital signatures.

The idealized repository maintained by $\mathcal{Z}$ is just a way to abstract block dissemination in a peer-to-peer gossip network. In reality, each node requests their peers for the block content (using information from the block header), and honest peers respond with the content. Correspondingly, the idealized repository indexes block content by the block header, and delivers it upon request, if available. Note that block content associated with a particular block header may be unavailable when requested by some honest node at one point in time (*e.g.*, if the adversary did not make it available), but available when requested by another honest node at a later time (*e.g.*, if the adversary made it available in the meantime). Thus, the block header does not ensure data availability or consistency among honest nodes' download attempts, unlike in stronger primitives such as verifiable information dispersal [10, 29, 42, 54]. By modelling the network as an idealized repository, we abstract out details such

[1] Unlike [24], we do not model the upload bandwidth because honest nodes only send very few messages in our protocol.

[2] In our simplified model, each node owns one unit of stake, which is the same as $1/N$ fraction of the total stake where $N$ is the number of nodes.

as the network topology and data availability that are orthogonal to the issue being considered here: the bandwidth constraint.

## 3 HIGH LEVEL SECURITY ARGUMENT

Our proofs follow the techniques of [45] and [20]. The key difference between these techniques and our proof is that the former assume that the block propagation delay is always bounded by a constant $\Delta$. In our case, we first prove that under 'suitable' download rules and protocol parameters, with overwhelming probability, a large fraction of honestly proposed blocks are downloaded by all honest nodes within bounded delay.

To this effect, we consider *uniquely successful* time slots, in which there is exactly one honest block proposal (any other slots with block proposals are called *adversarial*). For a given download rule and protocol execution, we define a property $\mathsf{MaxDL}_K$ (shorthand for Maximum Download) by which under all adversarial strategies and throughout the execution, the block proposed in a uniquely successful slot is downloaded by all honest nodes within the first $K$ blocks downloaded since the beginning of that slot (Definition 2). If the time slot is long enough to allow downloading $K$ blocks, then the block proposed in any uniquely successful slot will be downloaded by all honest nodes before the end of the same time slot. Then, each block proposed in a uniquely successful slot increases the minimum length of all honest nodes' longest downloaded chains. (Lemma 1). This is the key stepping stone of earlier security proofs of LC [20, 45]. We then employ techniques of [45] to prove that PoS LC with the right parameters and a download rule such that $\mathsf{MaxDL}_K$ holds with overwhelming probability is secure (Theorem 1). This gives a general framework where in order to prove security of PoS LC with a specific download rule, one only needs to show that the download rule satisfies $\mathsf{MaxDL}_K$ with overwhelming probability.

The property $\mathsf{MaxDL}_K$ suggests a natural download rule. In a uniquely successful slot, the block proposed in that slot can be identified as the unique freshest block. Thus, downloading towards the freshest block allows an honest node to download the block proposed in that slot most straightforwardly. If the prefix of the freshest block contains less than $K$ blocks that have not been downloaded yet, then $\mathsf{MaxDL}_K$ will be satisfied. Thus, for a suitably chosen $K$, the block proposed in a uniquely successful slot will be downloaded within the same time slot with overwhelming probability (Lemmas 2, 3). In Section 6, we apply a similar analysis to two other download rules, 'equivocation avoidance' and 'blocklisting', to show that they too satisfy $\mathsf{MaxDL}_K$ with overwhelming probability for suitable $K$ (Lemmas 4, 5).

In Corollary 1, we identify the parameter values under which the protocol $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$ with the freshest block download rule is secure for a given desired resilience $\beta$ (similarly for 'equivocation avoidance' and 'blocklisting' in Corollary 2). For the rate of uniquely successful slots to exceed the rate of adversarial slots, we require that the rate of block production per slot, $\rho$, be bounded as a function of $\beta$, so that most slots with honest block proposals are also uniquely successful slots. A similar constraint exists in the synchronous model [17, 20, 45] where the product of the block production rate and network delay $\Delta$ is bounded by a function of $\beta$. Next, we require that the per slot bandwidth $K = \Omega(\kappa)$ (where $\kappa$ is the security parameter) so that $\mathsf{MaxDL}_K$ is satisfied with overwhelming probability. This implies that the time slot $\tau = \Delta_{\mathrm{h}} + \frac{K}{C} = \Omega(\kappa)$. This is similar to [24] where under a bandwidth constrained model, the probabilistic delay bound increases with the security parameter. Finally, the confirmation time $T_{\mathrm{conf}} = \Omega(\kappa^2)$ slots, similar to that in the synchronous model [45].

## 4 SECURITY PROOF

### 4.1 Definitions

The PoS LC protocol $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$ has three parameters. The length of each time slot is $\tau$ seconds, the average number of nodes eligible to propose a block per time slot is $\rho$, and the confirmation latency is $T_{\mathrm{conf}}$ slots. The network has the following additional parameters. Each honest node has a download bandwidth of $C$ block contents per second (for convenience, we fix the size of the block content). Henceforth, we fix $\tau = \Delta_{\mathrm{h}} + \frac{K}{C}$ such that each honest node can download the content of $K$ blocks in one time slot after receiving the headers proposed in that slot. The adversary controls $\beta$ fraction of the stake. We denote by $\kappa$ the security parameter. An event $E_\kappa$ will be said to occur with *overwhelming* probability if $\Pr[E_\kappa] \geq 1 - \mathrm{negl}(\kappa)$. Here, a function $f(\kappa)$ is said to be negligible or $\mathrm{negl}(\kappa)$ if for all $n > 0$, there exists $\kappa_n^*$ such that for all $\kappa > \kappa_n^*$, $f(\kappa) < \frac{1}{\kappa^n}$.

Define the random variables $H_t$ and $A_t$ for $t = 1, 2, \ldots$ to be the number of honest and adversarial nodes eligible to propose a block in slot $t$, respectively. We consider the regime where the number of nodes $N \to \infty$ and each of them holds an equal fraction of the total stake. In this setting, by the Poisson approximation to a binomial random variable, we have $H_t \overset{\text{i.i.d.}}{\sim} \mathrm{Poisson}((1-\beta)\rho)$ and $A_t \overset{\text{i.i.d.}}{\sim} \mathrm{Poisson}(\beta\rho)$, all independent of each other. An execution $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$ of time horizon $T_{\mathrm{h}}$ is defined as the sequence $\{H_t, A_t\}_{t \leq T_{\mathrm{h}}}$.

Denote by $\mathrm{d}C_i(t)$ the longest fully downloaded chain of an honest node $i$ at the end of slot $t$. Let $|b|$ denote the height of a block $b$. We will also use the same notation $|C|$ to denote the length of a chain $C$. Define $L_i(t) = |\mathrm{d}C_i(t)|$ and $L_{\min}(t) = \min_i L_i(t)$. At the end of each slot, honest node $i$ outputs the ledger $\mathsf{LOG}_i^t = \mathrm{d}C_i(t)^{\lceil T_{\mathrm{conf}}}$, which consists of a list of transactions as ordered in all blocks in $\mathrm{d}C_i(t)$ with time slot up to $t - T_{\mathrm{conf}}$.

For a given execution of a consensus protocol, we define the following two properties:

- *Safety:* For all adversarial strategies, for all time slots $t, t'$ and honest nodes $i, j$, $\mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^{t'}$ or $\mathsf{LOG}_j^{t'} \preceq \mathsf{LOG}_i^t$.
- *Liveness with parameter* $T_{\mathrm{live}}$*:* For all adversarial strategies, if a transaction tx is received by all honest nodes before slot $t$, then for all honest nodes $i$ and slots $t' \geq t + T_{\mathrm{live}}$, $\mathrm{tx} \in \mathsf{LOG}_i^{t'}$.

A consensus protocol is *secure* over a time horizon $T_{\mathrm{h}}$ with parameter $T_{\mathrm{live}}$ if it satisfies safety and liveness with parameter $T_{\mathrm{live}}$ with overwhelming probability over executions of time horizon $T_{\mathrm{h}}$.

**Definition 1.** A slot $t$ is called *successful* if $A_t + H_t > 0$, *uniquely successful* if $A_t = 0$ and $H_t = 1$, and adversarial if it is successful but not uniquely successful. Define the predicates $\mathsf{Unique}(t)$ as true iff slot $t$ is uniquely successful and $\mathsf{Adv}(t)$ as true iff slot $t$ is adversarial.

For $s > r$, denote by $\mathcal{B}(r, s]$, $\mathcal{U}(r, s]$ and $\mathcal{A}(r, s]$, the number of successful, uniquely successful, and adversarial slots in the interval

$(r, s]$ respectively.

$$\mathcal{U}(r,s] \triangleq \sum_{t=r+1}^{s} \mathbb{1}\{\mathsf{Unique}(t)\}, \quad \mathcal{A}(r,s] \triangleq \sum_{t=r+1}^{s} \mathbb{1}\{\mathsf{Adv}(t)\} \quad (1)$$

and $\mathcal{B}(r,s] = \mathcal{U}(r,s] + \mathcal{A}(r,s]$. When $r = s$, then $(r,s] = \emptyset$ and thus $\mathcal{B}(r,s] = \mathcal{U}(r,s] = \mathcal{A}(r,s] = 0$. We define the following constants:

$$p \triangleq \Pr[A_t + H_t > 0] = 1 - e^{-\rho}, \quad (2)$$

$$p_{\mathrm{U}} \triangleq \Pr[\mathsf{Unique}(t)] \quad = (1-\beta)\rho e^{-\rho}, \quad (3)$$

$$p_{\mathrm{A}} \triangleq \Pr[\mathsf{Adv}(t)] \quad = p - p_{\mathrm{U}} \quad (4)$$

**Definition 2.** For a given download rule $\mathcal{D}$, execution $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$ and $r < s \leq T_{\mathrm{h}}$, $\mathsf{MaxDL}_{K,(r,s]}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})$ holds iff for all adversarial strategies, for all uniquely successful slots in $(r, s]$, the block proposed in that slot is downloaded by all honest nodes within the first $K$ blocks downloaded in that slot.

We abbreviate $\mathsf{MaxDL}_{K,(0,T_{\mathrm{h}}]}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})$ as $\mathsf{MaxDL}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})$. The inputs $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$ and $\mathcal{D}$ to predicates are omitted where obvious.

## 4.2 General Proof Overview

**Lemma 1.** *Let a download rule $\mathcal{D}$, an execution $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$ and $t_0 < s \leq T_{\mathrm{h}}$ be such that $\mathsf{MaxDL}_{K,(t_0,s]}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})$ holds. Let $t_1, ..., t_m$ be the uniquely successful slots in $(t_0, s]$. Then,*
*(1) For all $j \geq 1$, $|b_j| > |b_{j-1}|$, where $b_j$ is the block proposed in $t_j$.*
*(2) For all $0 \leq j \leq m$ and $t_j \leq t \leq s$,*

$$L_{\min}(t) - L_{\min}(t_j) \geq \mathcal{U}(t_j, t] \quad (5)$$

Proof. Part (1) is easily seen by the fact that honest nodes propose on their longest valid downloaded chain, $b_{j-1}$ is downloaded before $b_j$ is proposed, and is valid because it was proposed by an honest node. Now, fix a $j$ such that $0 \leq j \leq m$. If $j = m$, then $\mathcal{U}(t_m, t] = 0$ and $L_{\min}(t) \geq L_{\min}(t_m)$ for all $t_m \leq t \leq s$ because $L_{\min}$ is non-decreasing. For $j < m$, since honest nodes propose on their longest downloaded chain, $|b_{j+1}| \geq L_{\min}(t_{j+1} - 1) + 1 \geq L_{\min}(t_j) + 1$. From part (1) and that the blocks from uniquely successful slots in $(t_j, t]$ are downloaded before the end of their respective slots, we conclude that $L_{\min}(t) \geq |b_{j+1}| + \mathcal{U}(t_j, t] - 1 \geq L_{\min}(t_j) + \mathcal{U}(t_j, t]$. □

**Theorem 1.** *For all $K \in \mathbb{N}$ and download rules $\mathcal{D}$ such that*

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}} : \neg\mathsf{MaxDL}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})\right] \leq \mathrm{negl}(\kappa), \quad (6)$$

*if $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$, $\tau = \Delta_{\mathrm{h}} + \frac{K}{C}$ and $T_{\mathrm{conf}} = \Omega\left((\kappa + \ln T_{\mathrm{h}})^2\right)$, then the protocol $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$ with download rule $\mathcal{D}$ is secure with $T_{\mathrm{live}} = \Omega\left((\kappa + \ln T_{\mathrm{h}})^2\right)$.*

Theorem 1 is proved in Appendix D.1 using techniques similar to [45].

## 4.3 'Download Towards the Freshest Block' Rule

**Definition 3.** For an execution $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$, $\mathsf{ShortPrefixes}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}})$ holds iff

$$\forall t \leq T_{\mathrm{h}} : \max_{r<t:\ \mathsf{Unique}(r) \wedge (\mathcal{A}(r,t] \geq \mathcal{U}(r,t])} \mathcal{A}(r,t] < K. \quad (7)$$

**Lemma 2.** *For an execution $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$ and the freshest block download rule $\mathcal{D}_{\mathsf{fresh}}$ (Algorithm 2),*

$$\mathsf{ShortPrefixes}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}) \implies \mathsf{MaxDL}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D}_{\mathsf{fresh}}) \quad (8)$$

Proof. Let $t_1, ..., t_m$ be the uniquely successful slots in $(0, T_{\mathrm{h}}]$. Let $b_j$ be the block from $t_j$ for some $1 \leq j \leq m$. The header of $b_j$ is received by all honest nodes within $\Delta_{\mathrm{h}}$ time after the beginning of slot $t_j$. Due to the downloading rule, during slot $t_j$ all honest nodes download the chain containing $b_j$. Furthermore, since $b_j$ is an honest block and honest nodes only propose on their downloaded chain, the prefix of $b_j$ can be downloaded (*i.e.*, does not contain invalid or missing blocks). Thus, we only need to show that the prefix of $b_j$ contains at most $K$ blocks whose contents have not been downloaded.

For induction, assume that $\mathsf{MaxDL}_{K,(0,t_j-1]}$ holds. Using this, we will show that $\mathsf{MaxDL}_{K,(0,t_{j+1}-1]}$ holds. For the base case, this is true for $j = 1$ since $t_1$ is the first uniquely successful slot by definition. Note that the block $b_j$, being honest, is proposed on the tip of $\mathrm{d}\mathcal{C}_i(t_j - 1)$ for some $i$. Let $r_j$ be the last unique time slot such that the block $b'_j$ from that time slot is in $\mathrm{d}\mathcal{C}_i(t_j - 1)$. Clearly, $r_j \leq t_j - 1$. Then,

$$|\mathrm{d}\mathcal{C}_i(t_j - 1)| \leq |b'_j| + \mathcal{A}(r_j, t_j - 1] \quad (9)$$

since blocks after $b'_j$ are from adversarial slots by definition of $r_j$. From $\mathsf{MaxDL}_{K,(0,t_j-1]}$ and part (1) of Lemma 1,

$$|b_{j-1}| \geq |b'_j| + \mathcal{U}(r_j, t_j - 1]. \quad (10)$$

Since $b_{j-1}$ is downloaded by the end of slot $t_{j-1}$ and $t_j - 1 \geq t_{j-1}$, $|\mathrm{d}\mathcal{C}_i(t_j - 1)| \geq |b_{j-1}|$, and this would imply from (9) and (10) that $\mathcal{A}(r_j, t_j - 1] \geq \mathcal{U}(r_j, t_j - 1]$. Note that time slots of blocks in a valid chain must be strictly increasing. Since $b'_j$ is already downloaded, the number of blocks in $\mathrm{d}\mathcal{C}_i(t_j - 1)$ whose content is not downloaded is at most $\mathcal{A}(r_j, t_j - 1]$. Since $b_j$ extends $\mathrm{d}\mathcal{C}_i(t_j - 1)$, the number of block contents to be downloaded including the prefix of $b_j$ is at most $\mathcal{A}(r_j, t_j - 1] + 1$. As per $\mathsf{ShortPrefixes}_K$, this is at most $K$ (note that $r_j \leq t_j - 1$). Therefore, $b_j$ is downloaded within one slot. Since there are no more uniquely successful slots in $(t_j, t_{j+1})$, this completes the induction step by showing that $\mathsf{MaxDL}_{K,(0,t_{j+1}-1]}$. For $j = m$, we would conclude with $\mathsf{MaxDL}_K$ as required. □

**Lemma 3.** *If $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$ and $K = p_{\mathrm{A}} T(1+\epsilon_2)$ for some $\epsilon_2 > 0$ where $T = \frac{\Omega(\kappa + \ln T_{\mathrm{h}})}{\alpha_2 p}$, then*

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}} : \neg\mathsf{ShortPrefixes}_{K}(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}})\right] \leq \mathrm{negl}(\kappa), \quad (11)$$

*where $\alpha_2 = \min\left\{\frac{\epsilon_1^2}{36}, \frac{\epsilon_2^2}{\epsilon_2+2} \frac{p_{\mathrm{A}}}{p}\right\}$.*

**Corollary 1.** *The protocol $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$ with the freshest block download rule and parameters $\rho$ such that $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$, $\tau = \Omega(\kappa + \ln T_{\mathrm{h}})$, $T_{\mathrm{conf}} = \Omega\left((\kappa + \ln T_{\mathrm{h}})^2\right)$ is secure with $T_{\mathrm{live}} = \Omega\left((\kappa + \ln T_{\mathrm{h}})^2\right)$.*

Lemma 3 is proved in Appendix D.4 and Corollary 1 is obtained by setting $\tau = \Delta_{\mathrm{h}} + \frac{K}{C}$ and $K$ as per Lemma 3.

# 5 EXPERIMENTS

## 5.1 Implementation Details

We implemented our PoS LC node in 800 lines of Golang.[3] For all of our experiments, the slot duration $\tau$ is set to 1 second, and the

[3] The source code is available at: https://github.com/yangl1996/synclc-sim

total block production rate is 0.06 blocks/s. There is no transaction processing. Instead, nodes fill blocks with random bytes up to a size limit (100 KB in our experiments).

Our implementation has a fully-featured network stack modelled after Cardano's node software [14, 15]. Similar to Cardano, block propagation involves two subsystems: *chain sync*, and *block fetch*. The chain sync subsystem allows a node to advertise the header chain of the longest chain it has downloaded and validated, and to track the header chains advertised by peers. Because the header only takes a tiny fraction of space in a block, the bandwidth consumed by the chain sync subsystem is negligible. In all of our experiments, chain sync only consumed up to 1.2% of the available bandwidth.

The block fetch subsystem periodically examines the header chains learned from peers through chain sync, and sends requests to download block bodies according to a download rule. We implement the two download rules discussed in Section 1: 'download along the longest header chain', and 'download towards the freshest block'. Similar to Cardano, our block fetch logic limits the maximum number of peers to concurrently download from, an important parameter which we call the *in-flight cap*. This ensures the limited network bandwidth is never spread too thin across too many concurrent downloads. Finally, chain sync and block fetch share the same TCP connection for each pair of peers. To avoid head-of-line blocking, we multiplex the two subsystems so that chain sync is never impaired by block fetch traffic.

To simulate bandwidth constraints, we build our testbed using Mininet [37]. Each blockchain node runs in a Mininet virtual host with its own network interface, and is connected to a central switch through a link with limited bandwidth and artificial propagation delay. Specifically, we limit the bandwidth of honest nodes to 20 Mbps, and adversarial nodes to 1 Gbps. We set the round-trip time between any pair of nodes to 100 ms. The testbed runs on a workstation with two Intel Xeon E5-2623 v3 CPUs and 32 GB of RAM.

## 5.2 Demonstration of the Spamming Attack

In this experiment, we show that the widely-adopted 'download along the longest chain' rule is vulnerable to adversarial spamming, and the 'download towards the freshest block' rule mitigates this attack. There are 20 honest nodes connected in a full mesh topology. Honest nodes equally split 67% of the total stake, so each honest node has a block production rate of 0.002 block/s. The adversary controls 33% of the stake (0.02 block/s), and sets up 5 attacking nodes. Each attacking node connects to all honest nodes. The adversary uses the attacking nodes to monitor the longest chains announced by honest nodes, and tries to mine equivocating spam chains (cf. Figure 3). When successful, the adversary announces them and hopes honest nodes download these spam chains.

Figure 5 shows the time series of honest chain growth over an hour when the in-flight cap is set to 2. Note that honest chain growth stalls after 400 seconds when nodes download the longest chain. Since there are 5 attacking nodes, once the adversary gets a longer chain by luck, each honest user will use all of its 2 in-flight slots to download spam chains (from 2 of the 5 attacking nodes), leaving no room for honest blocks. Before any honest node finishes downloading a spam block, the adversary will have advertised another equivocating chain, keeping the honest nodes busy. Although honest nodes can still mine blocks, they cannot download blocks from each other, so each honest node effectively mines on its own fork. The resulting heavy forking causes the honest chains to grow slower than the adversary mining rate, and the adversary maintains the lead in chain length and sustains this attack (red-shaded areas in Figure 5) until the experiment ends.

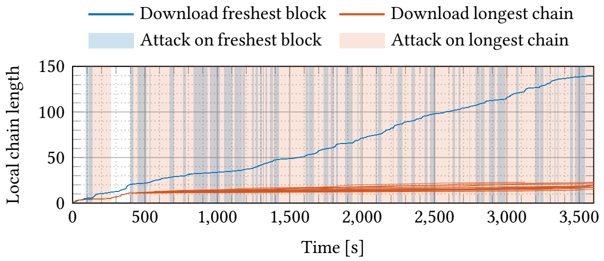

**Figure 5: Traces of honest chain growth under spamming attack (cf. Figure 3) when using different download rules and an in-flight cap of 2. Each curve represents one honest node. Shaded areas represent time periods when nodes are suffering from the attack and are downloading invalid blocks. PoS LC downloading longest chain stalls. PoS LC downloading freshest blocks is robust.**

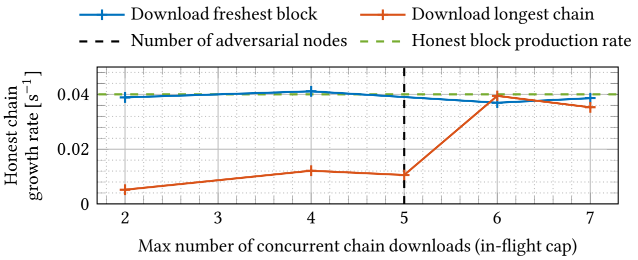

**Figure 6: Honest chain growth rate under spamming attack (cf. Figure 3) while allowing concurrent block downloads from different number of peers. With in-flight cap below the number of adversarial peers, PoS LC downloading longest chain shows performance degradation; PoS LC downloading freshest block is robust.**

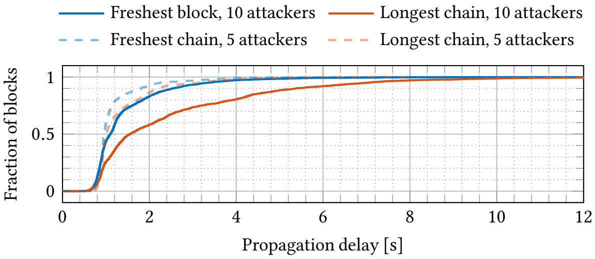

**Figure 7: Empirical cumulative density function of block propagation delay under different download rules, facing different number of attackers, and an infinite in-flight cap.**

In comparison, honest chain growth is unaffected when nodes download towards the freshest block. Note that although the adversary is still able to trick honest nodes into downloading spam blocks (blue-shaded areas in Figure 5), the adversary cannot *sustain* the attack: when a new honest block is produced, the chain containing that fresh block will be prioritized. Before the adversary manages to produce a fresher block, all honest nodes will have caught up on the correct chain. Further experiments in Appendix B.2 show that honest chain growth is unaffected with even larger block sizes.

## 5.3 Impact of the In-Flight Cap

We now extend the previous experiment by varying the in-flight cap between 2 and 7, and demonstrate the relationship between the in-flight cap and the number of attacking nodes. Figure 6 shows the results. When the in-flight cap is equal to or smaller than the number of attacking nodes, downloading the longest chain is not secure. This may remind readers of the eclipse attack [12, 28, 49–51]: the adversary is in fact eclipsing the honest nodes in the block fetch subsystem by occupying all its in-flight slots. Meanwhile, downloading the freshest chain is always secure *regardless* of the in-flight cap, because a fresh honest block can *break* such eclipse.

Figure 6 seems to suggest that downloading the longest chain is secure when the in-flight cap is larger than the number of attackers. Is it true? Should we then increase the in-flight cap to infinity? We point out that the in-flight cap ensures each in-flight download gets a sufficiently large share of the available bandwidth to complete in a reasonable amount of time. This is critical in ensuring low propagation delay for honest blocks. As an extreme example, assume that there are a large number of attacking nodes and an infinite in-flight cap. Although a node will always start downloading an honest block as soon as it receives the announcement, the bandwidth allocated to download this block will be extremely small due to the competing downloads of adversarial blocks, effectively halting the download. As a result, a finite in-flight cap is necessary, and the attacker can always attack the ‘download along the longest chain’ rule by outnumbering the in-flight cap.

To demonstrate this effect, we remove the in-flight cap, increase the number of attacking nodes to 10, and measure the block propagation time. The results in Figure 7 show that the propagation time under both rules increases. This is because when the attack *is* active, there are more competing flows downloading spam blocks, leaving less bandwidth for honest blocks. Still, the chain growth rate is unharmed when downloading the freshest chain, at 0.041 block/s. This is because nodes can break away from the spam chain as soon as a new honest block is produced, regardless how bad the propagation time is under active spam. In comparison, the propagation delay when downloading the longest chain becomes much worse. In fact, the higher delay causes the chain growth rate to drop to 0.035 block/s. In conclusion, removing or increasing the in-flight cap does not save the ‘download along the longest chain’ rule, but impacts the block propagation delay of the ‘download towards the freshest block’ rule only slightly so that security is unaffected.

## 5.4 Bandwidth Consumption

Besides block bodies, a blockchain node needs to receive other types of traffic in real time, such as unconfirmed transactions, requests from clients, and remote control data. A practical download rule must not consume all the available bandwidth at a node at all time. As explained in Section 5.2, under the ‘download towards the freshest block’ rule, an honest node breaks free from the spam chains when an honest block is mined. That is, spamming stops when there is a time slot with only one honest block proposed. Intuition suggests that as long as the overall mining rate is not too high, such event should happen frequently. Indeed, the ingress traffic traces in Figure 8 show that periods of high network utilization only last for tens of seconds when downloading the freshest block, quickly succeeded by long windows of low utilization. In comparison, when downloading the longest chain, the period of high utilization lasts until the end of the experiment, leaving no room for honest blocks or other traffic.

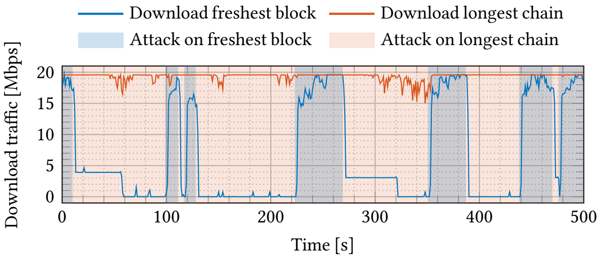

**Figure 8: Traces of download traffic over a 500-second period at one of the victim nodes when using different download rules and in-flight cap of 4. Shaded areas represent time periods when the node is suffering from the attack and downloading invalid blocks.**

# 6 OTHER DOWNLOAD RULES

## 6.1 Equivocation Avoidance

We formalize a common heuristic to deal with equivocations, namely downloading only one out of many equivocations in Algorithm 4. In every time slot when a node wishes to download blocks, it filters the tree consisting of block headers it has received by retaining only one leaf in the tree for each block production opportunity (determined by the proposing node and the time slot of each block). We strengthen the adversary by allowing it to decide, per honest node and time slot, which among multiple equivocating headers would be retained. After removing equivocations, invalid chains, and chains that are already downloaded from the header tree, the node selects the longest header chain, and downloads the content for the first missing block in this chain.

Here, we have illustrated equivocation avoidance as a modification to the longest header chain download rule. By doing so, we show that while the longest header chain download rule was insecure by itself, it can be made secure by ‘avoiding equivocations’. However, equivocation avoidance could also be added to the freshest block rule to make it more efficient. Our analysis below suggests that the freshest block rule by itself is already more efficient than the longest header chain rule with equivocation avoidance because the latter requires downloading a much larger number of blocks within one time slot (see Corollaries 1, 2), leading to longer time slots and poorer bandwidth utilization.

**Algorithm 4** 'Equivocation avoidance' download rule; replaces downloadRule in Algorithm 1 (cf. Appendix A.1)

```
on $\Delta_\mathrm{h}$ time into each time slot $t$
    ▷ Before beginning block content downloads for time slot $t$, filter current header tree to keep at most one leaf per block production opportunity, i.e., per (node, time) pair (equivocation avoidance; ties broken adversarially)
    $\mathrm{h}\mathcal{T}^* \leftarrow \mathrm{oneLeafPerProductionOpportunity}(\Pi^{\rho,\tau,T_\mathrm{conf}}.\mathrm{h}\mathcal{T})$
function avoidEquivocations($\mathrm{h}\mathcal{T}$, blkTxs)
    ▷ Ignore chains with invalid content in any block
    $\mathcal{T}' \leftarrow \{C \in \mathrm{h}\mathcal{T}^* \mid \forall C' \preceq C\colon \mathrm{blkTxs}[C'] \neq \mathtt{invalid}\}$
    ▷ Ignore downloaded chains
    $\mathcal{T}' \leftarrow \{C \in \mathcal{T}' \mid \mathrm{blkTxs}[C] = \mathsf{unknown}\}$
    ▷ Select the longest chain
    $C \leftarrow \arg\max_{C' \in \mathcal{T}'} |C'|$
    ▷ Find the first not downloaded block on that chain (if non-existent: $\perp$)
    $C' \leftarrow \arg\min_{C'' \preceq C\colon \mathrm{blkTxs}[C''] = \mathsf{unknown}} |C''|$
    return $C'$
```

## 6.2 Analysis

We use the general framework developed in Section 4 to prove security of PoS LC under the equivocation avoidance download rule. Recall that we only need to prove that $\mathsf{MaxDL}_K$ (Definition 2) holds with overwhelming probability.

In a uniquely successful slot, honest nodes may not immediately download towards the block from that slot. This is because there could be other chains in a node's header tree that are longer (recall that Algorithm 4 prioritizes the longest header chain after removing equivocations and invalid prefixes). However, we can bound the number of blocks that will be downloaded before downloading the block from the uniquely successful slot. With equivocation avoidance, honest nodes retain only one leaf in their header tree per block production opportunity. So, honest nodes download at most one chain per block production opportunity. Since block production opportunities are bounded, we will show in Lemma 4 that there can not be too many longer chains in the honest node's header tree.

Define for slots $s \leq t$,

$$W_{s,t} \triangleq \max_{r<s\colon \mathsf{Unique}(r) \wedge (\mathcal{A}(r,s] \geq \mathcal{U}(r,t])} \mathcal{A}(r,s]\,. \tag{12}$$

**Definition 4.** For an execution $\mathcal{E}^{\rho,\beta,T_\mathrm{h}}$, $\mathsf{FewLongChains}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}})$ holds iff

$$\forall t \leq T_\mathrm{h}\colon W_{t-1,t-1} + \sum_{s \leq t} A_s W_{s,t} < K. \tag{13}$$

**Lemma 4.** *For an execution $\mathcal{E}^{\rho,\beta,T_\mathrm{h}}$ and the longest header chain download rule with equivocation avoidance $\mathcal{D}_{\mathsf{lhc-ea}}$ (Algorithm 4),*

$$\mathsf{FewLongChains}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}}) \implies \mathsf{MaxDL}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}}, \mathcal{D}_{\mathsf{lhc-ea}}) \tag{14}$$

**Lemma 5.** *If $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$ and $K = p_\mathrm{A} T_\mathrm{b}(1+\beta\rho T_\mathrm{b}(1+\epsilon_3))(1+\epsilon_2)$ for some $\epsilon_2, \epsilon_3 > 0$ where $T_\mathrm{b} = \frac{\Omega(\kappa + \ln T_\mathrm{h})}{\alpha_3 p}$, then*

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_\mathrm{h}}\colon \neg\mathsf{FewLongChains}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}})\right] \leq \mathrm{negl}(\kappa) \tag{15}$$

*where* $\alpha_3 = \max\left\{\frac{\epsilon_1^2}{36}, \frac{\epsilon_2^2}{\epsilon_2+2}\frac{p_\mathrm{A}}{p}, \frac{p_\mathrm{U}(1-\epsilon_3)}{p}\ln\left(\frac{p_\mathrm{U}}{1-p_\mathrm{U}}\right), \frac{\epsilon_3^2 p_\mathrm{U}}{2p}, \frac{\epsilon_3^2\beta\rho}{(\epsilon_3+2)p}\right\}$.

Lemma 4 is proved in Appendix D.5 and Lemma 5 in Appendix D.6. Then, we obtain the following corollary of Theorem 1.

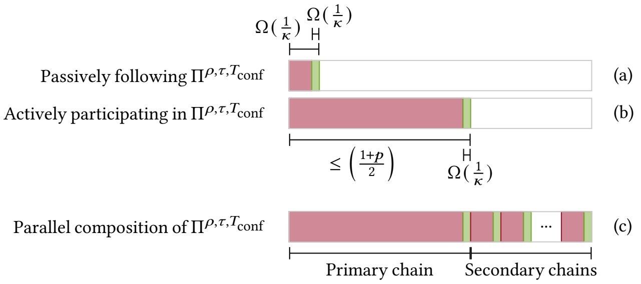
$\Omega(\frac{1}{\kappa})$
$\Omega(\frac{1}{\kappa})$
Passively following $\Pi^{\rho,\tau,T_\mathrm{conf}}$
(a)
Actively participating in $\Pi^{\rho,\tau,T_\mathrm{conf}}$
(b)
$\leq \left(\frac{1+p}{2}\right)$
$\Omega(\frac{1}{\kappa})$
Parallel composition of $\Pi^{\rho,\tau,T_\mathrm{conf}}$
...
(c)
Primary chain
Secondary chains

**Figure 9: Worst-case throughput and bandwidth consumption, as a fraction of the total bandwidth. Green portions represent bandwidth consumption that contributes to throughput, while red portions represent bandwidth consumption that is caused by the adversary and may not contribute to throughput (*e.g.*, empty/invalid blocks, spamming).**

**Corollary 2.** *The protocol $\Pi^{\rho,\tau,T_\mathrm{conf}}$ with the equivocation avoidance download rule and parameters $\rho$ such that $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$, $\tau = \Omega\left((\kappa + \ln T_\mathrm{h})^2\right)$ and $T_\mathrm{conf} = \Omega\left((\kappa + \ln T_\mathrm{h})^2\right)$, is secure with $T_\mathrm{live} = \Omega\left((\kappa + \ln T_\mathrm{h})^2\right)$.*

## 6.3 Blocklisting

Another common heuristic to deal with equivocations is 'blocklisting' the proposer of equivocating blocks. Blocklisting can be implemented at the level of the download rule as follows: an honest node never downloads a chain whose tip is proposed by a party for which the node has seen two block headers with the same time slot (an equivocation). Blocklisting is a decision that is taken unilaterally by each honest node and may be taken at different points of time by different nodes.

Note that this is only a stricter version of the equivocation avoidance rule described in Section 6.1 because in any given time slot, a block that is rejected in the equivocation avoidance rule will also be rejected in the blocklisting rule. Moreover, any chain whose tip is proposed by an honest node will not be discarded under this rule. Therefore, for any execution $\mathcal{E}^{\rho,\beta,T_\mathrm{h}}$ and the blocklisting rule $\mathcal{D}_\mathsf{blist}$

$$\mathsf{MaxDL}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}}, \mathcal{D}_{\mathsf{lhc-ea}}) \implies \mathsf{MaxDL}_K(\mathcal{E}^{\rho,\beta,T_\mathrm{h}}, \mathcal{D}_\mathsf{blist}). \tag{16}$$

Therefore, security of PoS LC with the 'blocklisting' rule is implied by security of PoS LC with 'equivocation avoidance' rule and the same parameters.

## 7 HIGH THROUGHPUT UNDER BANDWIDTH CONSTRAINT

In what follows, we use the freshest block download rule as our running example, but the results carry over analogously to other download rules analyzed using our unified framework, such as those in Section 6. From Corollary 1, we parameterize $\Pi^{\rho,\tau,T_\mathrm{conf}}$ with the freshest block download rule with $\tau = \Omega(\kappa)$ for security, so the protocol gets slower as the security parameter increases. A similar slowdown is also observed in the analysis in [24]. Thus, the throughput of $\Pi^{\rho,\tau,T_\mathrm{conf}}$ decreases with increasing security parameter. Indeed, we show in Section 7.2 that the worst-case throughput of $\Pi^{\rho,\tau,T_\mathrm{conf}}$ is lower bounded by $\frac{2p_\mathrm{U}-p}{\tau} = \frac{1}{\Omega(\kappa)}$.

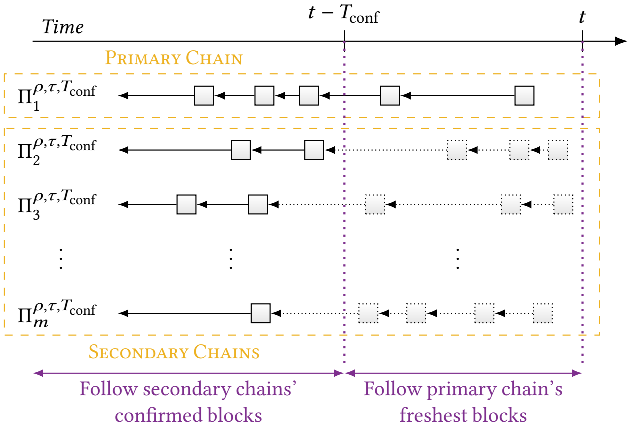
Time
$t - T_{\text{conf}}$
$t$
PRIMARY CHAIN
$\Pi_1^{\rho,\tau,T_{\text{conf}}}$
$\Pi_2^{\rho,\tau,T_{\text{conf}}}$
$\Pi_3^{\rho,\tau,T_{\text{conf}}}$
$\Pi_m^{\rho,\tau,T_{\text{conf}}}$
SECONDARY CHAINS
Follow secondary chains' confirmed blocks
Follow primary chain's freshest blocks

**Figure 10: In the parallel chains construction using $\Pi^{\rho,\tau,T_{\text{conf}}}$, each node is assigned one primary chain; the other $(m-1)$ chains are secondary. Nodes participate actively in their primary chain using, for example, the freshest block download rule, and follow their secondary chains passively by downloading confirmed blocks only.**

The slow block production rate also means that *passively following* the confirmed blocks of a chain only requires downloading up to $\frac{p}{\tau} = \frac{1}{\Omega(\kappa)}$ blocks per second because the secure protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ has already achieved consensus on these blocks (see Figure 9(a)). In fact, the ratio between throughput and the bandwidth required to download the confirmed blocks is the chain quality (fraction of honest blocks in the chain). This fraction, $\frac{2p_{\text{U}}-p}{p} > 0$ is independent of the security parameter $\kappa$. This suggests to invoke the idea of Parallel Chains [23, 24]: fill the available bandwidth using multiple instances of the slow LC protocol in parallel and combine the transactions of all instances into a single ledger. By increasing the number of chains, one can compensate for the decreasing throughput of the individual chains as $\kappa$ is increased.

However, following the confirmed chains alone is not enough to achieve consensus on all these chains. Note that the bandwidth consumption of a node *actively participating* in $\Pi^{\rho,\tau,T_{\text{conf}}}$ may be higher than what is required to download only the confirmed chain, due to spamming attacks. By spending this additional bandwidth, the nodes actively participating in $\Pi^{\rho,\tau,T_{\text{conf}}}$ make the protocol secure, which is what allows other nodes to passively follow and download the confirmed chain with little bandwidth consumption. However, even under spamming attacks, we show in Section 7.2 that the worst-case bandwidth consumption is only a little more than half the available bandwidth $C$ (shown in Figure 9(b)). This leaves nearly half the bandwidth available for a node participating in $\Pi^{\rho,\tau,T_{\text{conf}}}$ to download the confirmed portions of other parallel chains. This still allows us to increase the number of parallel chains to occupy the remaining bandwidth (shown in Figure 9(c)). So, we modify the parallel chains construction from [23, 24] as described in the following section.

## 7.1 Parallel Chains Construction

The protocol consists of $m$ parallel instances of $\Pi^{\rho,\tau,T_{\text{conf}}}$ (see Figure 10). For simplicity, assume that at genesis (and after the adversary has chosen which nodes to corrupt), stakeholders are randomly partitioned into $m$ equally sized sets, and the nodes of each set get assigned a particular instance of $\Pi^{\rho,\tau,T_{\text{conf}}}$ as their *primary chain*. Nodes are responsible for maintaining consensus on their primary chain. For this purpose, they download blocks as per a secure download rule and propose blocks on their primary chain as described in $\Pi^{\rho,\tau,T_{\text{conf}}}$. The remaining $(m-1)$ instances of $\Pi^{\rho,\tau,T_{\text{conf}}}$ that are not a node's primary chain are considered its *secondary chains*. Nodes do not participate actively in consensus building on their secondary chains, but only download the confirmed blocks from those chains, as determined by the $T_{\text{conf}}$-deep LC confirmation rule based on the block headers. Transactions from the confirmed portion of all the chains are first ordered by their time slots and then by the index of the protocol instance they appear in, to then be merged into a single output ledger. Moreover, every transaction can be included only in a single $\Pi^{\rho,\tau,T_{\text{conf}}}$ instance determined, *e.g.*, based on the transaction's hash or sender address, so as to avoid duplicating transactions across different $\Pi^{\rho,\tau,T_{\text{conf}}}$ instances. See Appendix C for pseudocode for the above parallel chains construction.

Each instance of $\Pi^{\rho,\tau,T_{\text{conf}}}$ is secure if at most $\beta$ fraction of nodes for whom this instance is the primary chain are corrupt, and the parameters $\rho, \tau, T_{\text{conf}}$ satisfy the constraints in Theorem 1. Note that if the number of stakeholders assigned to each primary chain is large, then the adversarial power in each instance of $\Pi^{\rho,\tau,T_{\text{conf}}}$ is very likely close to the overall adversarial power, rendering the construction secure against non-adaptive adversaries that corrupt at most $\beta$ fraction of the nodes. See Appendix E for a more detailed security analysis.

## 7.2 Throughput and Bandwidth Consumption

To quantify the throughput of $\Pi^{\rho,\tau,T_{\text{conf}}}$, we first note that the longest chain in any honest node's view grows at least at the rate of uniquely successful slots, $p_{\text{U}}$ blocks per slot (Lemma 1). Moreover, we can lower bound the chain quality, *i.e.*, the fraction of blocks in the blockchain in any honest node's view, which are proposed by honest nodes. All blocks proposed by honest nodes will contain distinct and valid transactions. Therefore, the chain quality along with the chain growth rate give a lower bound on the throughput.

**Lemma 6.** *(Throughput) There exists a constant $T_1$ such that for any honest node $i$ and time slots $t_1, t_2 \geq t_1 + T$ with $T \geq T_1$, $\mathrm{d}C_i(t_2) \backslash \mathrm{d}C_i(t_1)$ contains at least $\theta T(1-\epsilon_4)$ blocks proposed by honest nodes, with probability at least $1 - \exp(-\alpha_4 T)$, where $\theta = 2p_{\text{U}} - p$.*

From Lemma 6, the throughput of each chain is at least $\text{TP}_1 = \frac{\theta}{\tau}$ blocks per second.[4] Note that this lower bound holds under the worst-case adversary strategy.

Next, we calculate the bandwidth consumption of passively following the confirmed blocks of a secondary chain. Due to the security of $\Pi^{\rho,\tau,T_{\text{conf}}}$ run by the nodes for whom the corresponding

[4] For simplicity, we consider a constant number of transactions in each block. Hence, this directly translates to throughput in transactions per second.

chain is primary, the confirmed chain contains only valid available blocks and can be downloaded by spending little bandwidth without any interference from spamming blocks.

**Lemma 7.** *(Passive Bandwidth Consumption) There exists a constant $T_2$ such that for any honest nodes $i, i'$ and time slots $t_1, t_2 \geq t_1 + T$ such that $T \geq T_2$, $\mathrm{LOG}_{i'}^{t_2} \setminus \mathrm{LOG}_{i}^{t_1}$ contains at most $\phi_{\mathrm{p}} T(1+\epsilon_5)$ blocks, with probability at least $1 - \exp(-\alpha_5 T)$, where $\phi_{\mathrm{p}} = p$.*

Finally, we analyze the worst-case bandwidth consumption of active nodes in $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$. As per the freshest block download rule (see Algorithm 2, lines 6ff.), once all blocks proposed in the most recent non-empty time slot have been downloaded, the downloading node stays idle (because then $C = \perp$ in Algorithm 1, line 6). Since in every uniquely successful slot, each node downloads the freshest block within one slot (Lemma 2), the node thereafter remains idle until the next block proposal. This gives a simple lower bound on the worst-case fraction of time a node's bandwidth consumption is idle. (See Figure 8 for a matching observation in our experiments.)

**Lemma 8.** *(Active Bandwidth Consumption) There exists a constant $T_3$ such that for any honest node $i$ and time slots $t_1, t_2 \geq t_1 + T$ with $T \geq T_3$, node $i$ does not download any blocks for at least $\phi_{\mathrm{idle}} T\tau(1-\epsilon_6)$ time during the interval of time slots $(t_1, t_2]$, with probability at least $1 - \exp(-\alpha_6 T)$, where $\phi_{\mathrm{idle}} = \frac{p_{\mathrm{U}}(1-p)}{p} \geq \frac{1-p}{2}$.*

Lemmas 6, 7 and 8 are proved in Appendix D.7. Lemma 8 implies that a bandwidth of at least $\phi_{\mathrm{idle}} \cdot C$ remains unutilized by each node's primary chain. From Lemma 7, each node needs to download on average $\phi_{\mathrm{p}}$ blocks per slot, or $\frac{\phi_{\mathrm{p}}}{\tau}$ blocks per second, to follow the confirmed blocks of one of the secondary chains. This allows each node to follow $m - 1 = \frac{\phi_{\mathrm{idle}}}{\phi_{\mathrm{p}}} C\tau$ number of secondary chains. Therefore the $m$ parallel chains have an aggregate throughput of

$$\begin{aligned}\mathrm{TP}_m = m\,\mathrm{TP}_1 = \left(1 + \frac{\phi_{\mathrm{idle}}}{\phi_{\mathrm{p}}} C\tau\right)\frac{\theta}{\tau} &\geq \frac{(1-p)(2p_{\mathrm{U}}-p)}{2p} C \\ &= \frac{(1-p)\epsilon_1}{2} C \text{ blocks per second} \qquad (17)\end{aligned}$$

using $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$ from Theorem 1. Therefore, the aggregate throughput of the parallel chains remains within a constant fraction of the optimal throughput which is the bandwidth of $C$ blocks per second. This is true even if the number of secondary chains is parameterized so that the protocol produces an average load of only a certain fraction of the bandwidth left over by the primary chain, so as to bound queuing delays due to fluctuations in bandwidth utilization.

Notice that the throughput and passive bandwidth consumption of the protocol do not change with the download rule. With 'equivocation avoidance' and 'blocklisting', by doubling $K$ such that $\mathsf{MaxDL}_{K/2}$ holds (thereby roughly doubling the time slot duration), and ensuring that honest nodes do not download more than $K/2$ blocks in any slot, the active bandwidth utilization is explicitly bounded by half the available bandwidth. Thus, the parallel chains construction with these download rules also behaves similarly.

The worst-case throughput of a single chain and that of the parallel construction are limited by the chain quality factor $\epsilon_1 = \frac{2p_{\mathrm{U}}-p}{p}$ due to the possibility of selfish mining attacks [22]. Using the Conflux inclusion rule from [38] (which is also employed in [24]), this factor can be improved to $\frac{p_{\mathrm{U}}}{p}$ (which does not vanish as we push the resilience $\beta$ closer to $1/2$). In this rule, each block includes pointers to blocks that are not in its prefix, in order to include them in the ledger. To adapt this rule to bandwidth constrained networks, we modified it to ensure that only one block from each block production opportunity is pointed to and the number of pointers in each block is limited yet enough to include honest blocks. The details of this construction are in Appendix F.

Finally, in a comparison with [24], both works show a parallel chains construction that achieves throughput up to a constant fraction of the network capacity. However, our work proves this under worst-case adversarial strategies (including, *inter alia*, equivocation-based spamming), while [24] proves security only for adversaries that do not aggravate network congestion so much that a delay upper bound is violated. On the other hand, the security of our construction requires static corruption and honest majority among nodes in each chain (as each nodes performs consensus on one chain), whereas [24] works under a global honest majority assumption (as each node participates in all chains).

## ACKNOWLEDGMENT

We thank Dan Boneh and Ertem Nusret Tas for fruitful discussions. JN is supported by the Protocol Labs PhD Fellowship, a gift from Ethereum Foundation, and the Reed-Hodgson Stanford Graduate Fellowship.

## REFERENCES

[1] 2020. *Bitcoin Developer Guide – P2P Network – Initial Block Download – Headers-First.* https://developer.bitcoin.org/devguide/p2p_network.html#headers-first

[2] 2021. *Ethereum 2.0 networking specification.* https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md

[3] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP).* 375–392. https://doi.org/10.1109/SP.2017.29

[4] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *Conference on Computer and Communications Security (CCS '18).* ACM, 913–930.

[5] Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2019. Proof-of-Stake Longest Chain Protocols: Security vs Predictability. *arXiv preprint arXiv:1910.02218* (2019).

[6] 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 (CCS '19).* ACM, 585–602.

[7] Vitalik Buterin. 2014. Proof of Stake: How I Learned to Love Weak Subjectivity. Retrieved 2022-05-05 from https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/

[8] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. *CoRR* abs/1710.09437 (2017). arXiv:1710.09437 http://arxiv.org/abs/1710.09437

[9] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. 2020. Combining GHOST and Casper. *CoRR* abs/2003.03052 (2020). arXiv:2003.03052 https://arxiv.org/abs/2003.03052

[10] C. Cachin and S. Tessaro. 2005. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05).* 191–201. https://doi.org/10.1109/RELDIS.2005.9

[11] Tong Cao, Jiangshan Yu, Jérémie Decouchant, and Paulo Esteves-Verissimo. 2018. Revisiting Network-Level Attacks on Blockchain Network. (2018). https://orbilu.uni.lu/bitstream/10993/38142/1/bcrb18-cao.pdf

[12] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. 2003. Secure Routing for Structured Peer-to-Peer Overlay Networks. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2003), 299–314. https://doi.org/10.1145/844128.844156

[13] Sandro Coretti, Aggelos Kiayias, Cristopher Moore, and Alexander Russell. 2022. The Generals' Scuttlebutt: Byzantine-Resilient Gossip Protocols. Cryptology ePrint Archive, Report 2022/541. https://ia.cr/2022/541.

[14] Duncan Coutts, Neil David, Marcin Szamotulski, and Peter Thompson. 2020. *Introduction to the design of the Data Diffusion and Networking for Cardano Shelley.*

Technical Report. IOHK. Version 1.9.

[15] Duncan Coutts, Alex Vieth, Neil Davies, Marcin Szamotulski, Karl Knutsson, Marc Fontaine, and Armando Santos. 2021. *The Shelley Networking Protocol.* Technical Report. IOHK. Version 1.2.0, Revision 49.

[16] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security (FC '19)*. Springer, 23–41.

[17] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*. Springer, 66–98.

[18] Edsko de Vries, Thomas Winant, and Duncan Coutts. 2020. *The Cardano Consensus and Storage Layer.* https://github.com/input-output-hk/ouroboros-network/tree/master/ouroboros-consensus/docs/report

[19] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *P2P*. IEEE, 1–10.

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

[21] Cynthia Dwork and Moni Naor. 1992. Pricing via Processing or Combatting Junk Mail. In *CRYPTO (Lecture Notes in Computer Science, Vol. 740)*. Springer, 139–147.

[22] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.

[23] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition. Cryptology ePrint Archive, Report 1119.

[24] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2020. Proof-of-Stake Blockchain Protocols with Near-Optimal Throughput. Cryptology ePrint Archive, Report 2020/037. https://eprint.iacr.org/2020/037.

[25] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*. Springer, 281–310.

[26] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Stake Sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, 139–156.

[27] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2020. Tight Consistency Bounds for Bitcoin. (2020). https://eprint.iacr.org/2020/661.

[28] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 129–144. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[29] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. 2007. Verifying distributed erasure-coded data. In *PODC*. ACM, 139–146.

[30] IOHK. 2020. *input-output-hk/ouroboros-network*. https://github.com/input-output-hk/ouroboros-network/blob/master/ouroboros-network/src/Ouroboros/Network

[31] IOHK. 2020. *input-output-hk/ouroboros-network*. https://github.com/input-output-hk/ouroboros-network/blob/master/ouroboros-network/src/Ouroboros/Network/BlockFetch/Decision.hs#L162

[32] IOHK. 2021. *input-output-hk/ouroboros-network*. https://github.com/input-output-hk/ouroboros-network/blob/master/ouroboros-consensus-shelley/src/Ouroboros/Consensus/Shelley/Protocol.hs#L281

[33] Markus Jakobsson and Ari Juels. 1999. Proofs of Work and Bread Pudding Protocols. In *Communications and Multimedia Security (IFIP Conference Proceedings, Vol. 152)*. Kluwer, 258–272.

[34] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020. Proof-of-Burn. In *Financial Cryptography (Lecture Notes in Computer Science, Vol. 12059)*. Springer, 523–540.

[35] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO 2017*. Springer, 357–388.

[36] Aggelos Kiayias and Dionysis Zindros. 2019. Proof-of-Work Sidechains. In *Financial Cryptography Workshops (Lecture Notes in Computer Science, Vol. 11599)*. Springer, 21–34.

[37] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets*. ACM, 19.

[38] Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. 2018. Scaling Nakamoto Consensus to Thousands of Transactions per Second. *arXiv preprint arXiv:1805.03870* (2018).

[39] Michael McSweeney. 2021. *Solana experiences transaction stoppage as developers report 'intermittent instability'.* Retrieved 2021-11-21 from https://www.theblockcrypto.com/linked/117624/solana-experiences-transaction-stoppage-as-developers-report-intermittent-instability

[40] Mike Millard. 2022. *Solana restarted after seven-hour outage caused by surge of transactions.* Retrieved 2022-05-04 from https://www.theblockcrypto.com/linked/144639/solana-restarted-after-seven-hour-outage-caused-by-surge-of-transactions

[41] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf.

[42] Kamilla Nazirkhanova, Joachim Neu, and David Tse. 2021. Information Dispersal with Provable Retrievability for Rollups. *CoRR* abs/2111.12323 (2021).

[43] Joachim Neu, Ertem Nusret Tas, and David Tse. 2021. The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets. *CoRR* abs/2105.06075 (2021). arXiv:2105.06075 https://arxiv.org/abs/2105.06075

[44] R Pass, L Seeman, and A Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.

[45] Rafael Pass and Elaine Shi. 2017. The Sleepy Model of Consensus. In *ASIACRYPT 2017*. Springer, 380–409.

[46] Brian Quarmby. 2022. *Solana hit with another network incident causing degraded performance.* Retrieved 2021-05-04 from https://cointelegraph.com/news/solana-hit-with-another-network-incident-causing-degraded-performance

[47] Ling Ren. 2019. Analysis of Nakamoto Consensus. *IACR Cryptol. ePrint Arch.* (2019), 943.

[48] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. 2021. BFT Protocol Forensics. In *CCS*. ACM, 1722–1743.

[49] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. 2004. Defending against Eclipse Attacks on Overlay Networks. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop* (Leuven, Belgium) *(EW 11)*. Association for Computing Machinery, New York, NY, USA, 21–es. https://doi.org/10.1145/1133572.1133613

[50] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. 2006. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM*. IEEE.

[51] Emil Sit and Robert Morris. 2002. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Peer-to-Peer Systems*, Peter Druschel, Frans Kaashoek, and Antony Rowstron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 261–269.

[52] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 507–527.

[53] Anatoly Yakovenko. 2018. Solana: A new architecture for a high performance blockchain v0.8.13. Retrieved 2021-11-21 from https://solana.com/solana-whitepaper.pdf

[54] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 493–512. https://www.usenix.org/conference/nsdi22/presentation/yang

# A REFERENCE ALGORITHMS

## A.1 Helper Functions for Algorithms 1, 3, 4

- $\mathrm{Hash}(\mathrm{txs})$:
  Cryptographic hash function to produce a binding commitment to txs (modelled as a random oracle)
- $C' \preceq C$:
  Relation describing that $C'$ is a prefix of $C$
- $C \| C'$:
  Concatenation of $C$ and $C'$
- $\mathrm{prefixChainsOf}(C)$:
  Set of prefixes of $C$
- $\mathrm{longestChain}(\mathcal{T})$:
  Determine longest chain among set $\mathcal{T}$ of chains. Ties are broken by the adversary.
- $C^{\lceil T_{\mathrm{conf}}}$:
  Prefix of chain $C$ consisting of all blocks with time slots up to $T_{\mathrm{conf}}$ less than the current time slot
- $\mathrm{txsAreSemanticallyValidWrtPrefixesOf}(C, \mathrm{txs})$:
  Verifies for each transaction in txs that the transaction is semantically valid with respect to and properly authorized by the owner of the underlying assets as determined by the transaction's prefix in the ledger resulting from appending txs to the transactions as ordered in $C$ (assumes that content of all blocks in $C$ is known to the node)
- $\mathrm{newBlock}(\mathsf{time}\colon t, \mathsf{node}\colon P, \mathsf{txsHash}\colon \mathrm{Hash}(\mathrm{txs}))$:
  Produces a new block header with the given parameters
- $\mathrm{oneLeafPerProductionOpportunity}(\mathrm{h}\mathcal{T})$:

Filter header tree h$\mathcal{T}$ to keep at most one leaf per block production opportunity, *i.e.*, per (node, time) pair (equivocation avoidance; ties broken adversarially)

## A.2 Environment $\mathcal{Z}$

The environment $\mathcal{Z}$ initializes $N$ nodes and lets $\mathcal{A}$ corrupt up to $\beta N$ nodes at the beginning of the execution. Corrupted nodes are controlled by the adversary. Honest nodes run $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$. The environment maintains a mapping blkTxs from block headers to the block content (transactions). This mapping is referred to as the 'idealized repository' in Section 2. $\mathcal{Z}$ also maintains for each node a queue of pending block headers to be delivered after a delay determined by the adversary (at most $\Delta_{\mathrm{h}}$). Honest nodes and the adversary interact with $\mathcal{Z}$ via the following functions:

- $\mathcal{Z}$.broadcastHeaderChain($C$):
  If called by an honest node, $\mathcal{Z}$ sends header chain $C$ to $\mathcal{A}$. Then, for each honest node $P$, on receiving deliver($C, P$) from $\mathcal{A}$, or when $\Delta_{\mathrm{h}}$ time has passed since $C$ was handed to $\mathcal{Z}$ for broadcasting, $\mathcal{Z}$ triggers $P$.receivedHeaderChain($C$).
- $\mathcal{Z}$.uploadContent($C$, txs):
  $\mathcal{Z}$ stores a mapping from the header chain $C$ to the content txs of its last block by setting blkTxs[$C$] = txs. $\mathcal{Z}$ only stores the content txs if Hash(txs) = $C$.txsHash.
- $\mathcal{Z}$.requestContent($C$):
  If blkTxs[$C$] is set, then let txs = blkTxs[$C$] (if not, $\mathcal{Z}$ ignores the request). On receiving this call from an honest node $P$ in a time slot $t$, if $\mathcal{Z}$ has triggered $P$.receivedContent(.) less than $K$ times in slot $t$, then $\mathcal{Z}$ triggers $P$.receivedContent($C$, txs). On receiving this call from $\mathcal{A}$, $\mathcal{Z}$ sends ($C$, txs) to $\mathcal{A}$.
- $\mathcal{Z}$.receivePendingTxsSemanticallyValidWrt($C$):
  $\mathcal{Z}$ generates a set of pending transactions that are not included in the block contents of but semantically valid (see Appendix A.1) with respect to $C$, and returns them.
- $\mathcal{Z}$.outputLedger($C$):
  On receiving this call from a node $P$, $\mathcal{Z}$ records $C$ as $P$'s ledger to be externalized. This constitutes $\mathrm{LOG}_i^t$, for which consistency and liveness are required for a secure consensus protocol.

# B SUPPLEMENTAL EXPERIMENTAL MATERIAL

## B.1 Experimental Setup Details for Figure 1

For this experiment, we start 17 Cardano nodes in 17 AWS data centers across the globe and connect them into a fully-connected graph. We point out that the Cardano block fetch logic includes an optimization to only download blocks that have larger heights than the locally-adopted longest chain. As a result, a node may not eventually download every block whose header it sees. To demonstrate network congestion in the absence of a suitable download rule, we modify the code to disable this optimization and ensure that every node eventually downloads all blocks. To show congestion, we configure a variable number ($N$) of nodes to all mine blocks at the beginning of the same slot, and report the time for all 17 nodes to download all $N$ blocks.

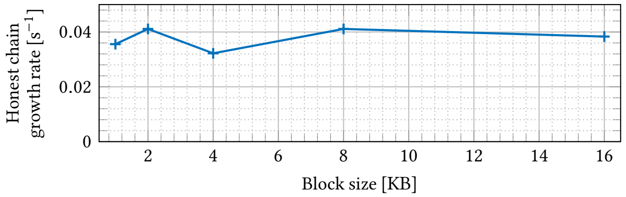

**Figure 11: Honest chain growth rate under spamming attack when using different block sizes and the download freshest block rule. Despite the increasing network load (through the increasing block size), there is no performance deterioration when downloading the freshest block.**

## B.2 Chain Growth with Larger Blocks

In this experiment, we look at the robustness of the 'download towards the freshest block' rule when we increase the block size. The topology is the same as previous experiments, but the in-flight cap is fixed to 1. Figure 11 shows that this rule maintains the chain growth rate, despite the increasing network load.

# C PARALLEL CHAINS PSEUDOCODE

Algorithm 5 gives pseudocode for the parallel chains construction using our PoS LC protocol parameterized with a download rule. Note the following main differences with respect to Algorithm 1. Upon initialization, each node is assigned a primary protocol instance index by the functionality $\mathcal{F}^{\rho,m}_{\mathrm{parallel}}$. Each node maintains a separate header tree and downloaded chain for each index. While scheduling content downloads, primary instance blocks get the highest priority, with the same download rule that parameterizes $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$. If there are no blocks left to be downloaded in the primary instance, the node picks among the confirmed longest chains of all secondary instances, the block with the oldest time slot with unknown content. Downloading the block with the oldest time slot allows the node to construct the ledger quickly, although this priority rule does not play a critical role in the consensus security. In line 25, the ledger is constructed by ordering the confirmed blocks of all the instances first by their time slots and then by the index of the protocol instance they appear in. The functionality $\mathcal{F}^{\rho,m}_{\mathrm{parallel}}$ (Algorithm 6) assigns the primary chain index for each node by uniformly and randomly partitioning the set of nodes across the $m$ chains. This can be approximated in instantiations by each node selecting as its primary chain index a hash of its public key modulo $m$.

Rather than by the transaction hash, another way to shard transactions is by distributing all accounts uniformly among the protocol instances, and requiring transactions in a particular instance to have both the source and destination accounts in the same instance. Transactions with the source and destination accounts in different instances would be split into two transactions, one which burns the funds in the source account and subsequently another one which recreates funds in the destination account (while showing a receipt of burn in the source chain), each transaction in its respective protocol instance (see [26, 34, 36] and references therein for background on this technique). Such a solution allows validation of each transaction with respect to its prefix within the same instance at the time

**Algorithm 5** Parallel Chains PoS LC consensus protocol $\Pi_{\text{pc}}^{\rho,\tau,T_{\text{conf}},m}$ (helper functions: Appendix C.1, $\mathcal{F}_{\text{parallel}}^{\rho,m}$: Algorithm 6, $\Pi^{\rho,\tau,T_{\text{conf}}}$: Algorithm 1)

```
1:  on INIT(genesisHeaderChain, genesisTxs)
2:      pri ← $\mathcal{F}_{\text{parallel}}^{\rho,m}$.PRIMARYCHAININDEX()
3:      for idx = 1, ..., $m$
4:          $\Pi_{\text{idx}}$ ← new $\Pi^{\rho,\tau,T_{\text{conf}}}$        ▷ Initialize $m$ instances of $\Pi^{\rho,\tau,T_{\text{conf}}}$
5:          $\Pi_{\text{idx}}$.INIT(genesisHeaderChain, genesisTxs)
6:  on RECEIVEDHEADERCHAIN(idx, $C$)
7:      $\Pi_{\text{idx}}$.RECEIVEDHEADERCHAIN($C$)
8:  on RECEIVEDCONTENT(idx, $C$, txs)
9:      $\Pi_{\text{idx}}$.RECEIVEDCONTENT($C$, txs)
10: on SCHEDULECONTENTDOWNLOAD()        ▷ Called when download idle
11:     $\Pi_{\text{pri}}$.SCHEDULECONTENTDOWNLOAD()  ▷ First priority for primary
12:     if no content requested by $\Pi_{\text{pri}}$
13:         ▷ Download first missing block along the confirmed portion of the
    longest header chains in the secondary instances.
14:         $\mathcal{S} \leftarrow \{\text{longestChain}(\Pi_{\text{idx}}.\text{h}\mathcal{T})^{\lceil T_{\text{conf}}} \mid \text{idx} \in \{1, \dots, m\} \setminus \{\text{pri}\}\}$
15:         $C \leftarrow \arg\min_{C'' \preceq C' \in \mathcal{S}\colon \text{blkTxs}[C'']=\text{unknown}} C''.\text{time}$
16:         $\mathcal{Z}$.REQUESTCONTENT($C$)
17: for time slots $t \leftarrow 1, \dots, T_{\text{h}}$ of duration $\tau$
18:     ▷ Only include valid txs whose accounts belong to the primary chain
19:     txs ← $\mathcal{Z}$.RECEIVEPENDINGTXSSEMANTICALLYVALIDWRT($\Pi_{\text{pri}}$.d$C$)
20:     ▷ Check eligibility to produce a new block, and if so do so, see Algorithm 6
21:     if $C' \neq \perp$ with $C' \leftarrow \mathcal{F}_{\text{parallel}}^{\rho,m}$.EXTEND(pri, $t$, $\Pi_{\text{pri}}$.d$C$, txs)
22:         $\mathcal{Z}$.UPLOADCONTENT(pri, $C'$, txs)
23:         $\mathcal{Z}$.BROADCASTHEADERCHAIN(pri, $C'$)
24:     while end of current time slot $t$ not reached
25:         SCHEDULECONTENTDOWNLOAD()
26:     ▷ Find the maximum time slot of all downloaded and confirmed chains
27:     tmax ← $\max\{t \mid \Pi_{\text{idx}}.\text{d}C^{\lceil T_{\text{conf}}}.\text{time} \geq t, \text{idx} \in \{1, \dots, m\}\}$
28:     ▷ Arrange confirmed and downloaded chains in increasing order of
    time slots, then chain index
29:     LOG ← sortBySlotThenIndex($\{C \mid C \preceq \Pi_{\text{idx}}.\text{d}C^{\lceil T_{\text{conf}}},$
    $C.\text{time} \leq \text{tmax}, \text{idx} \in \{1, \dots, m\}\}$)
30:     $\mathcal{Z}$.OUTPUTLEDGER(LOG)
```

**Algorithm 6** Idealized functionality $\mathcal{F}_{\text{parallel}}^{\rho,m}$ for parallel chains (see also $\mathcal{F}_{\text{headertree}}^{\rho}$: Algorithm 3)

```
on INIT(genesisHeaderChain, numParties)
    $\mathcal{P}_1, \dots, \mathcal{P}_m$ ← random equi-partition of {1, ..., numParties}
    for idx = 1, ..., $m$
        for $P \in \mathcal{P}_{\text{idx}}$
            pri[$P$] ← idx
        $\mathcal{F}_{\text{idx}}$ ← new $\mathcal{F}_{\text{headertree}}^{\rho}$        ▷ Initialize $m$ instances of $\mathcal{F}_{\text{headertree}}^{\rho}$
        $\mathcal{F}_{\text{idx}}$.INIT(genesisHeaderChain, numParties/$m$)
on PRIMARYCHAININDEX() from party $P$
    return pri[$P$]
on EXTEND(idx, $t'$, $C$, txs) from party $P$ at time slot $t$
    if pri[$P$] ≠ idx
        return ⊥
    return $\mathcal{F}_{\text{idx}}$.EXTEND($t'$, $C$, txs)
```

of block production (Algorithm 5 line 19), a property sometimes referred to as *predictable validity*. An important consequence of this is that there is no "ledger sanitization" procedure required while constructing the ledger out of the confirmed blocks. In other words, transactions once added to the chain cannot be invalidated in the ledger because they were validated with respect to their past state while proposing and forwarding the block. Thus, every transaction contributes to throughput.

## C.1 Additional Helper Functions for Algorithm 5 (see also Appendix A.1)

- sortBySlotThenIndex($\mathcal{S}$):
  Arranges the chains in the set $\mathcal{S}$ in increasing order of time slots of their tip. Chains with the same time slot from different protocol instances are arranged in increasing order of the index of their protocol instance.
- longestChain($\mathcal{T}$):
  Computes the longest chain in the tree $\mathcal{T}$, i.e. computes $\arg\max_{C\in\mathcal{T}} |C|$
- $\mathcal{Z}$.RECEIVEPENDINGTXSSEMANTICALLYVALIDWRT($C$):
  Same as in the case of a single chain, but only includes transactions for which the source account is defined in the same chain $C$.

# D PROOF DETAILS

## D.1 Proof of Theorem 1

**Definition 5.** A *pivot* is a slot $t$ such that

$$\forall (r,s] \ni t\colon \ (\mathcal{U}(r,s] > \mathcal{A}(r,s]) \vee (\mathcal{A}(r,s] = 0)\,. \tag{18}$$

The predicate $\text{Pivot}(t)$ is true iff $t$ is a pivot. A slot $t$ is a unique pivot slot iff $\text{Pivot}(t) \wedge \text{Unique}(t)$.

**Definition 6.** For an execution $\mathcal{E}^{\rho,\beta,T_{\text{h}}}$, $\text{FreqPivots}_{\gamma}(\mathcal{E}^{\rho,\beta,T_{\text{h}}})$ holds iff

$$\forall t \leq T_{\text{h}} - \gamma\colon \exists t' \in (t, t+\gamma]\colon \ \text{Pivot}(t') \wedge \text{Unique}(t'). \tag{19}$$

**Lemma 9.** *For all $K, \gamma \in \mathbb{N}$, $\rho \in \mathbb{R}^{+}$, executions $\mathcal{E}^{\rho,\beta,T_{\text{h}}}$ and download rules $\mathcal{D}$ such that*

$$\text{FreqPivots}_{\gamma}(\mathcal{E}^{\rho,\beta,T_{\text{h}}}) \wedge \text{MaxDL}_{K}(\mathcal{E}^{\rho,\beta,T_{\text{h}}}, \mathcal{D}) \tag{20}$$

*holds, if $\tau = \Delta_{\text{h}} + \frac{K}{C}$ and $T_{\text{conf}} = \gamma$, then the protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ with download rule $\mathcal{D}$ satisfies safety and liveness with $T_{\text{live}} = 2\gamma$ in $\mathcal{E}^{\rho,\beta,T_{\text{h}}}$.*

**Lemma 10.** *If $p_{\text{U}} = \frac{1}{2}p(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$ and $\gamma = \frac{\Omega((\kappa+\ln T_{\text{h}})^2)}{\alpha_1 p}$, then*

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_{\text{h}}}\colon \neg\text{FreqPivots}_{\gamma}(\mathcal{E}^{\rho,\beta,T_{\text{h}}})\right] \leq \text{negl}(\kappa) \tag{21}$$

*where $\alpha_1$ is a constant that depends on $\epsilon_1$ and $\rho$.*

Lemma 9 is proved in Appendix D.2 and Lemma 10 in Appendix D.3.

Proof of Theorem 1. Using Lemma 9, safety and liveness hold except with probability

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_{\text{h}}}\colon \neg\text{FreqPivots}_{\gamma}(\mathcal{E}^{\rho,\beta,T_{\text{h}}}) \vee \neg\text{MaxDL}_{K}(\mathcal{E}^{\rho,\beta,T_{\text{h}}}, \mathcal{D})\right]. \tag{22}$$

This probability is negligible as per a union bound with Lemma 10 and the assumption about the download rule. □

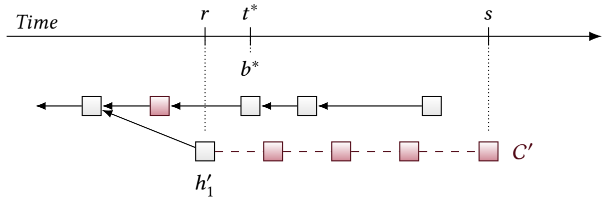

**Figure 12: An illustration of one example of the blocks and time slots defined in the proof of Lemma 11. The block $b^*$ is proposed in the unique pivot slot $t^*$. At the end of slot $s \geq t^*$, the chain $C' \not\ni b^*$ is the longest chain in some node's view. The last block from a uniquely successful slot in $C'$ is $h'_1$ proposed in the slot $r < t^*$. Red (🟥) and gray (□) blocks are proposed by adversarial and honest nodes, respectively. A red dashed link (- -) indicates that the block is withheld and released later. Note that in this example, $\mathcal{A}(r,s] = 4 > 3 = \mathcal{U}(r,s]$, which is in contradiction to** Pivot$(t^*)$**.**

## D.2 Proof of Lemma 9

**Lemma 11.** *Suppose that for a download rule $\mathcal{D}$ and execution $\mathcal{E}^{\rho,\beta,T_\text{h}}$,* $\text{MaxDL}_K(\mathcal{E}^{\rho,\beta,T_\text{h}},\mathcal{D})$ *holds. Let $t^*$ be a time slot such that* $\text{Pivot}(t^*) \wedge \text{Unique}(t^*)$. *Let $b^*$ be the block proposed in slot $t^*$. Then $b^* \in \text{d}C_i(t)$ for all $i$ and all $t \geq t^*$.*

Proof. For contradiction, suppose that $s \geq t^*$ is the first slot such that $b^* \notin \text{d}C_i(s)$ for some $i$. Let $C' = \text{d}C_i(s)$ such that $b^* \notin C'$. Let $h'$ be the last block corresponding to a uniquely successful slot on $C'$. Let $h'$ be proposed in the slot $r$. Clearly, $r \leq s$.

The block $h'$ extends $\text{d}C_{i'}(r-1)$ for some $i'$ since honest nodes propose blocks on their longest downloaded chain. Since $h' \in C'$ and $b^* \notin C'$, this means that $b^* \notin \text{d}C_{i'}(r-1)$. If $r > t^*$, this is a contradiction because we assumed that $s$ is the first slot such that $s \geq t^*$ and $b^* \notin \text{d}C_i(s)$ for some $i$. Since $\text{Unique}(t^*)$, $r \neq t^*$. So, we conclude that $r < t^*$. All blocks in $C'$ extending $h'$ are from successful slots that are not uniquely successful, *i.e.*, they are adversarial slots. So,

$$|C'| \leq |h'| + \mathcal{A}(r,s] \tag{23}$$

From Lemma 1,

$$L_{\min}(s) \geq L_{\min}(r) + \mathcal{U}(r,s]\,. \tag{24}$$

Note that $L_{\min}(s) \leq L_i(s)\ \forall i$ and $|C'| = L_i(s)$ for some $i$. Also note that $h'$ is from a uniquely successful slot $r$ and $\text{MaxDL}_K$ holds, so $L_{\min}(r) \geq |h'|$. Using the above observations with (23) and (24), we get

$$\mathcal{U}(r,s] \leq \mathcal{A}(r,s] \tag{25}$$

where $r < t^*$ and $s \geq t^*$. Since $\text{Pivot}(t^*)$, this is a contradiction. □

Lemma 11 shows that the block from every unique pivot slot stays in all honest nodes' downloaded longest chains thereafter. Therefore, under $\text{FreqPivots}_\gamma$, every interval of $\gamma$ slots brings at least one such block. To conclude with the proof of Lemma 9, one needs to show that the occurrence of such blocks leads to safety and liveness. This is done in Lemma 9.

Proof of Lemma 9. Let $T_\text{conf} = \gamma$. First, we prove safety by contradiction. Suppose that for some honest nodes $i, j$ and $t' \geq t$ that $\text{d}C_i(t)^{\lceil T_\text{conf}} \not\preceq \text{d}C_j(t')^{\lceil T_\text{conf}}$. We can assume that $t \geq \gamma$ because otherwise $\text{d}C_i(t)^{\lceil T_\text{conf}} = \emptyset$ and therefore $\text{d}C_i(t)^{\lceil T_\text{conf}} \preceq \text{d}C_j(t')^{\lceil T_\text{conf}}$ for all $t'$.

Consider all the uniquely successful slots $t_1, ..., t_m \in (t-\gamma, t]$ with block $b_j$ proposed in slot $t_j$. Suppose that $b_j \in \text{d}C_i(t)$ and $b_j \in \text{d}C_j(t')$. Then $\text{d}C_i(t)$ and $\text{d}C_j(t')$ match up to $b_j$. Since $t_j > t - \gamma$, $\text{d}C_i(t)^{\lceil T_\text{conf}} \preceq \text{d}C_j(t')$. Also, $t' \geq t$, therefore $\text{d}C_i(t)^{\lceil T_\text{conf}} \preceq \text{d}C_j(t')^{\lceil T_\text{conf}}$ which is a contradiction to our assumption. Therefore, for each $j = 1, ..., m$, either $b_j \notin \text{d}C_i(t)$ or $b_j \notin \text{d}C_j(t')$. This means that for all $j = 1, ..., m$, $b_j$ is not a great block. Due to $\text{ShortPrefixes}_K$ and Lemma 11, this also means that there are no unique pivot slots in the interval $(t-\gamma, t]$, which is a contradiction to $\text{FreqPivots}_\gamma$.

We next prove liveness. Assume a transaction tx is received by all honest nodes before time $t$. We know that there exists a unique pivot slot $t^*$ in the interval $(t, t+\gamma]$. The honest block $b^*$ from $t^*$ or its prefix must contain tx since tx is seen by all honest nodes at time $t < t^*$. Moreover, $b^*$ is also a great block, *i.e.*, $b^* \in \text{d}C_i(t')$ for all honest nodes $i$ and $t' \geq t^*$. Therefore, $\text{tx} \in \text{LOG}_i^{t'}$ for all $t' \geq t^* + T_\text{conf}$, which is at most $t + 2\gamma$. □

## D.3 Proof of Lemma 10

### D.3.1 Preliminaries.

**Definition 7** (Pivot condition)**.** The predicate $\text{PivotCondition}_{(r,s]}$ holds iff $\mathcal{U}(r,s] > \mathcal{A}(r,s]$.

Note that $\text{Pivot}(t)$ holds iff $\forall (r,s] \ni t$, $\text{PivotCondition}_{(r,s]} \vee (\mathcal{A}(r,s] = 0)$ holds.

**Definition 8** (Weak Pivot)**.** Time slot $t$ satisfies $\text{WeakPivot}_w(t)$ iff

$$\forall (r,s] \ni t, s-r < w\colon\ \text{PivotCondition}_{(r,s]} \vee (\mathcal{A}(r,s] = 0)\,. \tag{26}$$

**Proposition 1.** *If $p_\text{U} = \frac{1}{2}p(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$,*

$$\forall (r,s]\colon\ \Pr\left[\neg\text{PivotCondition}_{(r,s]}\right] \leq 2\exp\left(-\alpha'_1 p(s-r)\right), \tag{27}$$

*with $\alpha'_1 = \eta\epsilon_1^2$ and $\eta = 1/36$.*

Proof. By a simple Chernoff bound for $\epsilon > 0$,

$$\Pr\left[\mathcal{B}(r,s] \geq p(s-r)(1+\epsilon)\right] \leq \exp\left(-\frac{\epsilon^2 p(s-r)}{2+\epsilon}\right). \tag{28}$$

Also, by a Chernoff bound for $\epsilon \in (0,1)$,

$$\Pr\left[\mathcal{U}(r,s] \leq p_\text{U}(s-r)(1-\epsilon)\right] \leq \exp\left(-\frac{\epsilon^2 p_\text{U}(s-r)}{2}\right) \tag{29}$$

By choosing $\epsilon$ such that $\frac{1+\epsilon}{1-\epsilon} = 1 + \epsilon_1$, we obtain that

$$\begin{aligned}\mathcal{U}(r,s] &> p_\text{U}(s-r)(1-\epsilon)\\ &= \frac{1}{2}p(1+\epsilon_1)(s-r)(1-\epsilon)\\ &= \frac{1}{2}p(s-r)(1+\epsilon) > \frac{1}{2}\mathcal{B}(r,s]\\ \implies \mathcal{U}(r,s] &> \mathcal{A}(r,s]\,,\end{aligned}$$

except with probability

$$\exp\left(-\frac{\epsilon^2 p(s-r)}{2+\epsilon}\right) + \exp\left(-\frac{\epsilon^2 p_\text{U}(s-r)}{2}\right) \tag{30}$$

From $\frac{1+\epsilon}{1-\epsilon} = 1+\epsilon_1$, we get $\epsilon = \frac{\epsilon_1}{\epsilon_1+2} \geq \frac{\epsilon_1}{3}$. Further using $p_{\mathrm{U}} > \frac{p}{2}$, this probability is bounded by

$$2\exp\left(\frac{\epsilon_1^2 p(s-r)}{36}\right) \tag{31}$$

□

**Proposition 2.** *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for an execution horizon $T_{\mathrm{h}}$ and $w > \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1' p}$,*

$$\begin{aligned} &\Pr\left[\exists (r,s], s-r \geq w \colon \neg\mathsf{PivotCondition}_{(r,s]}\right] \\ &\qquad \leq 2T_{\mathrm{h}}^2 \exp\left(-\alpha_1' p w\right). \end{aligned} \tag{32}$$

Proof. Using a union bound and Proposition 1,

$$\begin{aligned} &\Pr\left[\exists (r,s], s-r \geq w \colon \neg\mathsf{PivotCondition}_{(r,s]}\right] \\ \leq &\sum_{(r,s], s-r\geq w} \Pr\left[\neg\mathsf{PivotCondition}_{(r,s]}\right] \\ \leq &\, 2T_{\mathrm{h}}^2 \exp(-\alpha_1' p w). \end{aligned}$$

□

**Proposition 3.** *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for a time horizon $T_{\mathrm{h}}$ and $w > \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1' p}$,*

$$\begin{aligned} &\Pr\left[\exists t \colon \mathsf{WeakPivot}_w(t) \land \neg\mathsf{Pivot}(t)\right] \\ &\qquad \leq 2T_{\mathrm{h}}^2 \exp(-\alpha_1' p w). \end{aligned} \tag{33}$$

Proof. If some $t$ is a weak pivot (with $w \geq \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1' p}$) and $t$ is not a pivot, then $\exists (r,s] \ni t$ with $s-r \geq w$ such that $\neg\mathsf{PivotCondition}_{(r,s]}$. But the probability for this is bounded accordingly by Proposition 2. □

**Proposition 4.** *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for time horizon $T_{\mathrm{h}}$,*

$$\forall t\colon \quad \Pr\left[\mathsf{WeakPivot}_w(t) \mid \mathsf{Unique}(t)\right] \geq p_1 \tag{34}$$

*where $p_1 = \frac{1}{2}(1-p_{\mathrm{A}})^{2v-1} > 0$ and $\frac{w}{2} > v = \frac{1}{\alpha_1' p}\ln\left(\frac{4(1+e^{-\alpha_1' p})}{(1-e^{-\alpha_1' p})^2}\right)$.*

Proof. For $v < w/2$ to be determined later, consider the events

$$E_1 \triangleq \{\mathcal{A}(t-v, t+v] = 0\}, \tag{35}$$

$$\begin{aligned} E_2 \triangleq \{&\forall (r,s] \ni t, s-r < w, (r,s] \notin (t-v, t+v] \colon \\ &\mathsf{PivotCondition}_{(r,s]}\}. \end{aligned} \tag{36}$$

Note that, $E_1 \cap E_2 \subseteq \{\mathsf{WeakPivot}_w(t)\}$ and $\Pr\left[E_1 \mid \mathsf{Unique}(t)\right] = (1-p_{\mathrm{A}})^{2v-1}$.

For bounding $\Pr\left[\neg E_2\right]$, we will use a union bound by carefully counting the number of intervals $(r,s] \ni t$ such that $s-r < w$ and $(r,s] \notin (t-v, t+v]$. Let $u = s-r$. For $u \leq v$, note that $(r,s] \ni t$ implies that $(r,s] \in (t-v, t+v]$. One can check that for $v+1 \leq u \leq 2v$, there are $2(u-v)-1$ intervals $(r,s] \ni t$ such that $(r,s] \notin (t-v, t+v]$. For $2v+1 \leq u < w$, all intervals $(r,s] \ni t$ are such that $(r,s] \notin (t-v, t+v]$, and there are $u$ such intervals. Therefore, from Proposition 1 and a union bound,

$$\begin{aligned} \Pr\left[\neg E_2\right] &\leq \sum_{u=v+1}^{w-1} \sum_{\substack{(r,s]\ni t\colon\\ s-r=u\land\\ (r,s]\notin(t-v,t+v]}} \Pr\left[\neg\mathsf{PivotCondition}(r,s]\right] \\ &\leq \sum_{u=v+1}^{2v} (2(u-v)-1)2e^{-\alpha_1' p u} + \sum_{u=2v+1}^{w-1} u2e^{-\alpha_1' p u} \\ &\leq \sum_{k=1}^{v} 2(2j-1)e^{-\alpha_1' p(v+j)} + \sum_{u=2v+1}^{w-1} 2ue^{-\alpha_1' p u} \\ &\leq \sum_{k=1}^{v} 2(2j-1)e^{-\alpha_1' p(v+j)} + \sum_{u=2v+1}^{\infty} 2ue^{-\alpha_1' p u} \\ &= \frac{2e^{-\alpha_1' p(v+1)}(1-(2v+1)e^{-\alpha_1' p v})}{1-e^{-\alpha_1' p}} \\ &\quad + \frac{4e^{-\alpha_1' p(v+2)}(1-e^{-\alpha_1' p v})}{(1-e^{-\alpha_1' p})^2} \\ &\quad + \frac{2(2v+1)e^{-\alpha_1' p(2v+1)}}{1-e^{-\alpha_1' p}} + \frac{2e^{-\alpha_1' p(2v+2)}}{(1-e^{-\alpha_1' p})^2} \\ &= \frac{2e^{-\alpha_1' p(v+1)}}{1-e^{-\alpha_1' p}} + \frac{4e^{-\alpha_1' p(v+2)} - 2e^{-\alpha_1' p(2v+2)}}{(1-e^{-\alpha_1' p})^2} \\ &\leq \frac{2e^{-\alpha_1' p(v+1)}}{1-e^{-\alpha_1' p}}\left(1+\frac{2e^{-\alpha_1' p}}{1-e^{-\alpha_1' p}}\right) \\ &\leq \frac{2e^{-\alpha_1' p v}(1+e^{-\alpha_1' p})}{(1-e^{-\alpha_1' p})^2} \end{aligned} \tag{37}$$

We may choose $v = \frac{1}{\alpha_1' p}\ln\left(\frac{4(1+e^{-\alpha_1' p})}{(1-e^{-\alpha_1' p})^2}\right)$, so that $\Pr\left[\neg E_2\right] \leq \frac{1}{2}$.

It is easy to see that $\Pr\left[E_2 \mid E_1 \cap \{\mathsf{Unique}(t)\}\right] \geq \Pr\left[E_2 \mid E_1\right] \geq \Pr\left[E_2\right]$.

$$\begin{aligned} \Pr\left[\mathsf{WeakPivot}_w(t) \mid \mathsf{Unique}(t)\right] &\geq \Pr\left[E_1 \cap E_2 \mid \mathsf{Unique}(t)\right] \\ &\geq \Pr\left[E_1 \mid \mathsf{Unique}(t)\right]\Pr\left[E_2\right] \\ &\geq \frac{1}{2}(1-p_{\mathrm{A}})^{2v-1}. \end{aligned}$$

for the given choice of $v$. □

**Proposition 5.** *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for horizon $T_{\mathrm{h}}$ and $w > \frac{2}{\alpha_1' p}\ln\left(\frac{4(1+e^{-\alpha_1' p})}{(1-e^{-\alpha_1' p})^2}\right)$,*

$$\begin{aligned} \forall t\colon &\Pr\left[\exists t' \in (t, t+\gamma] \colon \mathsf{WeakPivot}_w(t') \land \mathsf{Unique}(t')\right] \\ &\geq 1-\exp(-\alpha_1'' \gamma / w), \end{aligned} \tag{38}$$

*with $\alpha_1'' = \frac{p_1 p_{\mathrm{U}}}{2}$.*

Proof. Let $k$ be the largest integer such that $\gamma \geq 2wk$. For $i = 0, \ldots, (k-1)$, define $t_i = t + (2i+1)w$ and

$$E_i \triangleq \{\mathsf{WeakPivot}_w(t_i) \land \mathsf{Unique}(t_i)\} \tag{39}$$

$$E \triangleq \{\exists t' \in (t, t+\gamma] \colon \mathsf{WeakPivot}_w(t') \land \mathsf{Unique}(t')\}. \tag{40}$$

Thus, we have $\bigcup_{i=0}^{k-1} E_i \subseteq E$, and by construction $E_i$ are independent. Hence,

$$\begin{aligned} \Pr\left[E\right] \geq \Pr\left[\bigcup_{i=0}^{k-1} E_i\right] &= 1 - \Pr\left[\bigcap_{i=0}^{k-1} \neg E_i\right] \\ &\geq 1-(1-p_1 p_{\mathrm{U}})^k \\ &\geq 1-\exp(-p_1 p_{\mathrm{U}} k) \\ &= 1-\exp(-p_1 p_{\mathrm{U}} \gamma / 2w), \end{aligned} \tag{41}$$

where we have used Proposition 4. □

**Proposition 6.** *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for horizon $T_{\mathrm{h}}$, $w > \frac{2}{\alpha_1' p} \ln\left(\frac{4(1+e^{-\alpha_1' p})}{(1-e^{-\alpha_1' p})^2}\right)$ and $\gamma > \frac{w \ln(T_{\mathrm{h}})}{\alpha_1''}$,*

$$\Pr\left[\forall t\colon \exists t' \in (t, t+\gamma]\colon \mathsf{WeakPivot}_w(t') \wedge \mathsf{Unique}(t')\right] \geq 1 - T_{\mathrm{h}} \exp(-\alpha_1'' \gamma / w). \tag{42}$$

Proof. By a union bound over all $T_{\mathrm{h}}$ possible time slots, and using Proposition 5. □

### *D.3.2 Proof of Lemma 10.*

Proof. Finally, to prove Lemma 10, let

$$E_1 \triangleq \{\forall t\colon \exists t' \in (t, t+\gamma]\colon \mathsf{WeakPivot}_w(t') \wedge \mathsf{Unique}(t')\}$$
$$E_2 \triangleq \{\forall t\colon \mathsf{WeakPivot}_w(t) \Rightarrow \mathsf{Pivot}(t)\}$$
$$E \triangleq \{\forall t\colon \exists t' \in (t, t+\gamma]\colon \mathsf{Pivot}(t') \wedge \mathsf{Unique}(t')\}.$$

Note that $E_1 \cap E_2 \subseteq E$. Then we apply a union bound on the probabilities from Propositions 6 and 3.

$$\Pr[\neg E] \leq \Pr[\neg E_1] + \Pr[\neg E_2] \leq 2T_{\mathrm{h}}^2 e^{-\alpha_1' p w} + T_{\mathrm{h}} e^{-\alpha_1'' \gamma / w}. \tag{43}$$

Let $\kappa' = \kappa + \ln T_{\mathrm{h}}$. Pick $w$ such that $w = \frac{2\ln(\sqrt{2}T_{\mathrm{h}}) + \Omega(\kappa)}{\alpha_1' p}$. This ensures that the probability $2T_{\mathrm{h}}^2 e^{-\alpha_1' p w}$ corresponding to having more adversarial than honest slots in some interval of size at least $w$, is negl$(\kappa)$.

Finally, we pick $\gamma$ so that the probability $T_{\mathrm{h}} e^{-\alpha_1'' \gamma / w}$ corresponding to not finding a pivot slot in some interval of $\gamma$ slots, is negl$(\kappa)$. Therefore we get $\gamma \geq \frac{\ln(T_{\mathrm{h}}) + \Omega(\kappa)}{\alpha_1''} w$. Combining these, we have $\gamma \geq \frac{\Omega((\ln(T_{\mathrm{h}}) + \kappa)^2)}{\alpha_1' \alpha_1'' p}$. Choose $\alpha_1 = \alpha_1' \alpha_1''$ □

## D.4 Proof of Lemma 3

Proof. Define the event $F_t$ as

$$\max_{r<t\colon \mathsf{Unique}(r) \wedge (\mathcal{A}(r,t] \geq \mathcal{U}(r,t])} \mathcal{A}(r,t] \geq K. \tag{44}$$

This event can be equivalently expressed as

$$\exists r < t\colon \mathsf{Unique}(r) \wedge (\mathcal{A}(r,t] \geq \mathcal{U}(r,t]) \wedge (\mathcal{A}(r,t] \geq K). \tag{45}$$

The event $\{\neg\mathsf{ShortPrefixes}_K\}$ can be expressed as $\bigcup_{t \leq T_{\mathrm{h}}} F_t$. Then for some fixed $T$,

$$\begin{aligned}
\Pr[F_t] &\leq \Pr\left[\bigcup_{r=0}^{t-1} \{\mathcal{A}(r,t] \geq \mathcal{U}(r,t] \wedge \mathcal{A}(r,t] \geq K\}\right] \\
&\leq \sum_{r=0}^{t-T} \Pr[\mathcal{A}(r,t] \geq \mathcal{U}(r,t]] + \sum_{r=t-T}^{t-1} \Pr[\mathcal{A}(r,t] \geq K] \\
&\leq \sum_{k=T}^{\infty} 2\exp\left(-\alpha_1' p k\right) + T\exp\left(-\frac{\epsilon_2^2}{2+\epsilon_2} p_A T\right) \\
&= \frac{2\exp(-\alpha_1' p T)}{1-\exp(-\alpha_1' p)} + T\exp\left(-\frac{\epsilon_2^2}{2+\epsilon_2} p_A T\right) \\
&\leq 2T\exp\left(-\alpha_2 p T\right),
\end{aligned} \tag{46}$$

for $T \geq \frac{2}{1-\exp(-\alpha_1' p)}$ and $\alpha_2 = \min\left\{\alpha_1', \frac{\epsilon_2^2}{\epsilon_2+2} \frac{p_{\mathrm{A}}}{p}\right\}$. By using a union bound over the execution horizon $T_{\mathrm{h}}$, we get

$$\Pr[\neg\mathsf{ShortPrefixes}_K] \leq 2T_{\mathrm{h}} T \exp(-\alpha_2 p T) \leq 2T_{\mathrm{h}}^2 \exp(-\alpha_2 p T) \tag{47}$$

We then set $T = \frac{2\ln(\sqrt{2}T_{\mathrm{h}}) + \Omega(\kappa)}{\alpha_2 p}$ to make this probability negl$(\kappa)$. □

## D.5 Proof of Lemma 4

Proof. Let $t_1, \ldots, t_m$ be the uniquely successful slots in $(0, T_{\mathrm{h}}]$. Let $b_j$ be the block from slot $t_j$ for some $1 \leq j \leq m$.

For induction, assume that $\mathsf{MaxDL}_{K,(0,t_j-1]}$ holds. Using this, we will show that $\mathsf{MaxDL}_{K,(0,t_{j+1}-1]}$ holds. For the base case, this is true for $j = 1$ since $t_1$ is the first uniquely successful slot by definition. Suppose that there is a chain $C'$ in the header tree of an honest node in slot $t_j$ such that $|C'| \geq |b_j|$. Note that the tip of $C'$ can not be a unique block because unique blocks have increasing heights as per Lemma 1. Therefore the tip of $C'$ is from an adversarial slot. Consider such a chain $C'$ ending in a block from an adversarial slot $s_j \leq t_j$. Let $r_j$ be the last uniquely successful slot such that the block $b_j'$ from that slot is in $C'$. Then,

$$|C'| \leq |b_j'| + \mathcal{A}\left(r_j, s_j\right]. \tag{48}$$

From the assumption of $\mathsf{MaxDL}_{K,(0,t_j-1]}$ and part (1) of Lemma 1,

$$|b_j| \geq |b_j'| + \mathcal{U}\left(r_j, t_j\right]. \tag{49}$$

Since $|C'| \geq |b_j|$, this would mean that $\mathcal{A}\left(r_j, s_j\right] \geq \mathcal{U}\left(r_j, s_j\right]$. As a block from a uniquely successful slot, $b_j'$ was downloaded by all honest nodes within slot $r_j$. Therefore, there are at most $\mathcal{A}\left(r_j, s_j\right]$ blocks on the chain $C'$ that are yet to be downloaded. Therefore the number of blocks to be downloaded by each honest node on $C'$ is at most

$$\max_{r_j<s_j\colon \mathsf{Unique}(r_j) \wedge \left(\mathcal{A}\left(r_j,s_j\right] \geq \mathcal{U}\left(r_j,t_j\right]\right)} \mathcal{A}\left(r_j, s_j\right] = W_{s_j,t_j}. \tag{50}$$

Next, we count the number of such chains $C'$ with distinct block production opportunities at the tip. Due to the equivocation avoidance policy, the adversary can make honest nodes download at most one chain per adversarial block production opportunity in slots $s_j \leq t_j$. The total number of blocks to be downloaded in all these chains combined is $\sum_{s_j<t_j} A_{s_j} W_{s_j,t_j}$.

Finally, from the proof of Lemma 2, we note that the prefix of $b_j$ has at most $W_{t_j-1,t_j-1}$ blocks that need to be downloaded by any honest node. Therefore, the total number of blocks that any honest node needs to download before downloading $b_j$ is at most

$$W_{t_j-1,t_j-1} + \sum_{s_j \leq t_j} A_{r_j} W_{s_j,t_j}. \tag{51}$$

From the definition of $\mathsf{FewLongChains}_K$, this is less than $K$. Therefore, every honest node can download $b_j$ within the time slot $t_j$. This completes the induction step by showing that $\mathsf{MaxDL}_{K,(0,t_{j+1}-1]}$. For $j = m$, we conclude with $\mathsf{MaxDL}_K$ as required. □

## D.6 Proof of Lemma 5

Proof. From Lemma 3, we already know that for $N = p_{\mathrm{A}} T(1+\epsilon_2)$ and $T > \max\left\{\frac{2}{1-\exp(-\alpha_1' p)}, \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_2 p}\right\}$, we have

$$\Pr\left[\neg(\forall t \leq T_{\mathrm{h}} : W_{t,t} < N)\right] \leq 2T_{\mathrm{h}}^2 e^{-\alpha_2 p T}. \tag{52}$$

It is easy to see that for any given sample path (i.e. realization of $\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}$) and any $s \leq t$, $W_{s,t} \leq W_{t,t}$. Next, we can show that there exists some $T_{\mathrm{b}}$ such that $W_{s,t} = 0$ for all $s < t - T_{\mathrm{b}}$ and for all $t$, so that we have the following with overwhelming probability:

$$W_{t-1,t-1} + \sum_{s \le t} A_s W_{s,t} \le N + N T_{\mathrm{b}} \beta\rho(1+\epsilon). \tag{53}$$

This is because in any $T_{\mathrm{b}}$ slots, there are at most $T_{\mathrm{b}}\beta\rho(1+\epsilon)$ adversarial block production opportunities with probability at least $1 - T_{\mathrm{h}} \exp\left(-\frac{\epsilon^2 \beta\rho T_{\mathrm{b}}}{\epsilon+2}\right)$ (through a Chernoff bound and union bound).

To show that $W_{s,t} = 0$ for all $s < t - T_{\mathrm{b}}$ for a fixed $t$,

$$\begin{aligned}
&\Pr\left[\exists s < t - T_{\mathrm{b}} : W_{s,t} > 0\right] && (54)\\
\le&\Pr\left[\exists s < t - T_{\mathrm{b}}, \exists r < s : \mathcal{A}(r,s] \ge \mathcal{U}(r,t]\right] && (55)\\
\le&\Pr\left[\exists r < t - T_{\mathrm{b}} : \mathcal{A}(r, t-T_{\mathrm{b}}] \ge \mathcal{U}(r,t]\right] && (56)\\
\le&\Pr\left[\exists r < t - T_{\mathrm{b}} : \mathcal{A}(r, t-T_{\mathrm{b}}] \ge \mathcal{U}(r,t-T_{\mathrm{b}}] + \mathcal{U}(t-T_{\mathrm{b}},t]\right] && (57)\\
\le&\Pr\left[\exists r < t - T_{\mathrm{b}} : \mathcal{A}(r, t-T_{\mathrm{b}}] \ge \mathcal{U}(r,t-T_{\mathrm{b}}] + L\right] && (58)\\
&+\Pr\left[\mathcal{U}(t-T_{\mathrm{b}},t] < L\right] && (59)
\end{aligned}$$

where we choose $L = p_{\mathrm{U}} T_{\mathrm{b}}(1-\epsilon)$. The second term is bounded by a Chernoff bound

$$\Pr\left[\mathcal{U}(t-T_{\mathrm{b}},t] < L\right] \le \exp\left(-\frac{\epsilon^2 p_{\mathrm{U}} T_{\mathrm{b}}}{2}\right). \tag{60}$$

For calculating the first term, let

$$X_n = L + \mathcal{U}(t - T_{\mathrm{b}} - n, t - T_{\mathrm{b}}] - \mathcal{A}(t - T_{\mathrm{b}} - n, t - T_{\mathrm{b}}]$$

for $n \ge 0$ be a random walk. Let $p_l = \Pr\left[\exists n : X_n \le 0 \mid X_0 = l\right]$, i.e. the probability that the random walk ever hits 0 after starting from $l$. We can observe that $p_1 = 1 - p_{\mathrm{U}} + p_{\mathrm{U}} p_2$. We can also note that due to the translation invariance of the random walk,

$$\begin{aligned}
p_2 &= \Pr\left[\exists n : X_n \le 1 \mid X_0 = 2\right] \Pr\left[\exists n > n_1 : X_{n_1} \le 0 \mid X_{n_1} = 1\right]\\
&= \Pr\left[\exists n : X_n \le 0 \mid X_0 = 1\right]^2 = p_1^2.
\end{aligned}$$

Therefore, we obtain $p_1 = \frac{1-p_{\mathrm{U}}}{p_{\mathrm{U}}}$ by solving $p_1 = 1 - p_{\mathrm{U}} + p_{\mathrm{U}} p_1^2$. Finally, we note using the same logic as above that $p_L = p_1^L = \left(\frac{1-p_{\mathrm{U}}}{p_{\mathrm{U}}}\right)^L$ which is the required probability in the first term in (58).

Therefore, we have

$$\Pr\left[\exists s \le t - T_{\mathrm{b}} : W_{s,t} > 0\right] \le \left(\frac{1-p_{\mathrm{U}}}{p_{\mathrm{U}}}\right)^{p_{\mathrm{U}} T_{\mathrm{b}}(1-\epsilon)} + \exp\left(-\frac{\epsilon^2 p_{\mathrm{U}} T_{\mathrm{b}}}{2}\right) \tag{61}$$

Finally, by a union bound over the required probabilities, we have for $K = p_{\mathrm{A}} T(1 + \beta\rho T_{\mathrm{b}}(1+\epsilon))(1+\epsilon_2)$,

$$\begin{aligned}
&\Pr\left[\neg\mathsf{FewLongChains}_K\right] \le T_{\mathrm{h}} \left(\frac{1-p_{\mathrm{U}}}{p_{\mathrm{U}}}\right)^{p_{\mathrm{U}} T_{\mathrm{b}}(1-\epsilon)} && (62)\\
&+T_{\mathrm{h}} \exp\left(-\frac{\epsilon^2 p_{\mathrm{U}} T_{\mathrm{b}}}{2}\right) + T_{\mathrm{h}} \exp\left(-\frac{\epsilon^2 \beta\rho T_{\mathrm{b}}}{\epsilon+2}\right) + 2T_{\mathrm{h}}^2 \exp(-\alpha_2 p T) && (63)\\
&\le 5T_{\mathrm{h}}^2 \exp(-\alpha_3 p T_{\mathrm{b}}). && (64)
\end{aligned}$$

Here, we choose $T_{\mathrm{b}} = T$ and

$$\alpha_3 = \max\left\{\alpha_2, \frac{p_{\mathrm{U}}(1-\epsilon)}{p} \ln\left(\frac{p_{\mathrm{U}}}{1-p_{\mathrm{U}}}\right), \frac{\epsilon^2 p_{\mathrm{U}}}{2p}, \frac{\epsilon^2 \beta\rho}{(\epsilon+2)p}\right\}. \tag{65}$$

Finally, we set $T_{\mathrm{b}} = \frac{2\ln(\sqrt{5}T_{\mathrm{h}}) + \Omega(\kappa)}{\alpha_3 p}$ so that the required probability is negligible.

□

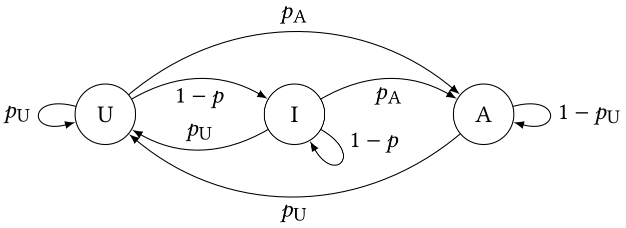

**Figure 13: An upper bound on the bandwidth utilization of our protocol can be calculated from the stationary distribution of this Markov chain**

## D.7 Proofs for Throughput and Bandwidth Consumption

### D.7.1 *Proof of Lemma 6.*

Proof. Due to Lemma 1, in any interval of slots $(t_1, t_2]$, the downloaded longest chain of every honest node grows by at least $\mathcal{U}(t_1, t_2]$ (even though all blocks on the chain may not be honest). Therefore, corresponding to the interval $(t_1, t_2]$ with $t_2 \ge t_1 + T$, at least $p_{\mathrm{U}} T(1-\epsilon)$ blocks are added to every node's downloaded longest chain with probability

$$\begin{aligned}
&\Pr\left[\mathcal{U}(t_1, t_2] \ge p_{\mathrm{U}} T(1-\epsilon)\right]\\
&\ge \Pr\left[\mathcal{U}(t_1, t_2] \ge p_{\mathrm{U}}(t_2 - t_1)(1-\epsilon)\right] \ge 1 - \exp\left(\frac{\epsilon^2}{2} p_{\mathrm{U}} T\right). && (66)
\end{aligned}$$

Now let $N = p_{\mathrm{U}} T(1-\epsilon)$. Consider any $N$ consecutive blocks in a valid blockchain. Let $t_1'$ and $t_2'$ be the time slots corresponding to the first and last blocks respectively in this set, and let $T' = t_2' - t_1'$. From the above probability bound, we have $T' \le T = \frac{N}{p_{\mathrm{U}}(1-\epsilon)}$. Also, with probability at least $1 - \exp\left(-\frac{\epsilon'^2}{2+\epsilon'} p_{\mathrm{A}} T'\right)$, there are at most $p_{\mathrm{A}} T'(1+\epsilon')$ adversarial slots in $(t_1', t_2']$, hence there are at most $p_{\mathrm{A}} T'(1+\epsilon')$ adversarial blocks in the $N$ consecutive blocks.

Therefore, corresponding to every interval $(t_1, t_2]$, there are at least $p_{\mathrm{U}} T(1-\epsilon) - p_{\mathrm{A}} T(1+\epsilon') = (p_{\mathrm{U}} - p_{\mathrm{A}}) T(1-\epsilon_4)$ honest blocks in any node's downloaded longest chain with probability at least $1 - \exp(-\alpha_4 T)$ for some constant $\alpha_4$. Finally, we note that $\theta = p_{\mathrm{U}} - p_{\mathrm{A}} = 2p_{\mathrm{U}} - p$. □

### D.7.2 *Proof of Lemma 7.*

Proof. Consider time slots $t_1$ and $t_2 \ge t_1 + T$. Due to the safety of $\Pi^{\rho,\tau,T_{\mathrm{conf}}}$, we know that $\mathsf{LOG}_i^{t_1} \preceq \mathsf{LOG}_{i'}^{t_2}$ for any honest nodes $i, i'$. The last block in $\mathsf{LOG}_i^{t_1}$ must have a time slot $t_1' \ge t_1 - 2T_{\mathrm{conf}}$ because between $t_1 - 2T_{\mathrm{conf}}$ and $t_1 - T_{\mathrm{conf}}$, there is at least one unique pivot slot which contributes a block to $\mathsf{LOG}_i^{t_1}$. Therefore $\mathsf{LOG}_{i'}^{t_2} \setminus \mathsf{LOG}_i^{t_1}$ contains only blocks with time slots in the interval $(t_1', t_2']$ where $t_2' = t_2 - T_{\mathrm{conf}}$. Note that blocks in the confirmed chain must have increasing time slots, so their number is limited by the number of slots with block proposal, *i.e.* $\mathcal{B}(t_1', t_2']$. The average number of slots with block proposal in the interval $(t_1', t_2']$ is $p(t_2' - t_1') \le p(t_2 - t_1 + T_{\mathrm{conf}}) = p(T + T_{\mathrm{conf}})$. Then by a Chernoff bound,

$$\Pr\left[\mathcal{B}(t_1', t_2'] > pT(1+\epsilon_5)\right] \le \exp\left(-\alpha_5 T\right) \tag{67}$$

for sufficiently large $T > T_{\mathrm{conf}}$ and some constant $\alpha_5$. □

### D.7.3 Proof of Lemma 8.

Proof. Consider the Markov chain shown in Figure 13 with three states—U corresponding to a uniquely successful slot, I corresponding to a slot without a block proposal such that the most recent block proposal was a uniquely successful slot, and A corresponding to adversarial slots or slots without block proposals such that the most recent block proposal was an adversarial slot.

The stationary distribution of this Markov chain is

$$\pi_{\mathrm{U}} = p_{\mathrm{U}}, \quad \pi_{\mathrm{I}} = \frac{p_{\mathrm{U}}(1-p)}{p}, \quad \pi_{\mathrm{A}} = \frac{p_{\mathrm{A}}}{p}. \tag{68}$$

Note that in time slots corresponding to the I (idle) state, there are no fresh blocks to be downloaded because the most recent block proposal was a unique honest block which was downloaded within 1 slot. Therefore, on average, in $\phi_{\mathrm{idle}}$ fraction of time slots, every honest node's bandwidth remains idle, where

$$\begin{aligned}\phi_{\mathrm{idle}} \geq \pi_{\mathrm{I}} &= \frac{p_{\mathrm{U}}(1-p)}{p} \\ &= \frac{1}{2}(1-p)(1+\epsilon_1) \\ &\geq \left(\frac{1-p}{2}\right).\end{aligned} \tag{69}$$

(For $\epsilon_1$, see the proof of Theorem 1.) Finally, by a Chernoff bound, the probability that for a given $t_1, t_2$, there are at least $\phi_{\mathrm{idle}} T(1-\epsilon_6)$ slots in the I state in the interval $(t_1, t_2]$ is at least $1-\exp\left(-\frac{\epsilon_6^2}{2}\phi_{\mathrm{idle}} T\right)$. □

## E SECURITY OF PARALLEL CHAINS

The below security theorem holds for any download rule which satisfies the requirement in Theorem 1, and in addition leaves a fraction $\phi_{\mathrm{idle}} \in (0,1)$ of the total bandwidth unutilized (cf. Lemma 8). The latter requirement can be easily achieved for any download rule for any desired $\phi_{\mathrm{idle}} \in (0,1)$ by increasing the time slot duration by a factor of $\frac{1}{\phi_{\mathrm{idle}}}$ and only downloading blocks in the first $\phi_{\mathrm{idle}}$ fraction of the time slot.

Also note that the below theorem holds under a static corruption adversary (*i.e.*, the adversary decides which nodes to corrupt before the randomness of the protocol is drawn).

**Theorem 2.** *For all $K \in \mathbb{N}$ and download rules $\mathcal{D}$ such that*

$$\Pr\left[\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}} : \neg\mathsf{MaxDL}_K(\mathcal{E}^{\rho,\beta,T_{\mathrm{h}}}, \mathcal{D})\right] \leq \mathrm{negl}(\kappa), \tag{70}$$

*if $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$, $\tau = \Omega(\kappa + \ln T_{\mathrm{h}})$, $T_{\mathrm{conf}} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$, Lemma 8 holds for some $\phi_{\mathrm{idle}} \in (0,1)$, and $m = 1+\frac{\phi_{\mathrm{idle}}}{\phi_{\mathrm{p}}} C\tau(1-\epsilon_7)$, then the protocol $\Pi_{\mathrm{pc}}^{\rho,\tau,T_{\mathrm{conf}},m}$ with the download rule $\mathcal{D}$ is secure with parameter $T_{\mathrm{live}} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$.*

Proof. Consider a particular protocol instance $\Pi_{\mathrm{idx}}$. Define $\mathrm{d}C_{i,\mathrm{idx}}$ to be the longest downloaded chain of node $i$ for protocol instance $\Pi_{\mathrm{idx}}$. From Theorem 1, for the given $\rho$, $\tau$ and $T_{\mathrm{conf}} = \gamma$, each protocol instance $\Pi_{\mathrm{idx}}$ satisfies safety and liveness with respect to the ledger defined by $\mathrm{d}C_{i,\mathrm{idx}}(t)^{\lceil T_{\mathrm{conf}}}$ and for nodes $i$ for which $\Pi_{\mathrm{idx}}$ is the primary chain, expect with probability $\mathrm{negl}(\kappa)$. By a union bound, safety and liveness for each protocol instance holds over $m = \mathrm{poly}(\kappa)$ protocol instances as well.

Due to safety of $\Pi_{\mathrm{idx}}$, $\mathrm{d}C_{i,\mathrm{idx}}(t)^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{j,\mathrm{idx}}(t')^{\lceil T_{\mathrm{conf}}}$ or $\mathrm{d}C_{j,\mathrm{idx}}(t')^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{i,\mathrm{idx}}(t)^{\lceil T_{\mathrm{conf}}}$ for all time slots $t, t'$ and all honest nodes $i, j$ for which $\Pi_{\mathrm{idx}}$ is the primary chain. However, this holds even if $\Pi_{\mathrm{idx}}$ is not the primary chain for node $i$ or $j$ because such nodes receive all block headers, determine the longest header chain based on them, and then download its confirmed prefix. More concretely, an adversary that pushes an inconsistent longest header chain to a node $j$ for which $\Pi_{\mathrm{idx}}$ is a secondary chain, can also do so with headers and contents for a node $j'$ for which $\Pi_{\mathrm{idx}}$ is the primary chain, thus causing a safety violation, which contradicts the earlier observation. Since all nodes have consistent confirmed chains (*i.e.* $\mathrm{d}C_{i,\mathrm{idx}}(t)^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{j,\mathrm{idx}}(t')^{\lceil T_{\mathrm{conf}}}$ or $\mathrm{d}C_{j,\mathrm{idx}}(t')^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{i,\mathrm{idx}}(t)^{\lceil T_{\mathrm{conf}}}$) for each protocol instance and the combined ledger is derived by ordering the blocks in all confirmed chains deterministically by their time slot, this implies safety of $\Pi_{\mathrm{pc}}^{\rho,\tau,T_{\mathrm{conf}},m}$ (*i.e.*, $\forall$ honest $i, j : \forall t, t' : \mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^{t'} \vee \mathsf{LOG}_j^{t'} \preceq \mathsf{LOG}_i^t$).

To show liveness, we first show that confirmed secondary chain blocks are downloaded with bounded delay. From Lemma 8, in any interval of $\tilde{T}$ slots, the bandwidth of each node is not requested for downloads related to the primary chain but available to download secondary chain blocks in at least $\phi_{\mathrm{idle}}\tilde{T}(1-\epsilon_5)$ slots. Further, from Lemma 7, in any interval of $\tilde{T}$ slots, the confirmed secondary chains grow by at most $\phi_{\mathrm{p}}\tilde{T}(1+\epsilon_6)$ blocks. These events happen with probability at least $1-\mathrm{negl}(\kappa)$ over a time horizon $T_{\mathrm{h}}$ with $\tilde{T} = \Omega(\kappa + \ln T_{\mathrm{h}})$. By a union bound over $m = \mathrm{poly}(\kappa)$ number of chains, these hold with at least $1-\mathrm{negl}(\kappa)$ probability over all chains. Therefore, in $\tilde{T}$ slots, all confirmed blocks in $m-1$ secondary chains can be downloaded, where $m - 1 = \frac{\phi_{\mathrm{idle}}\tilde{T}(1-\epsilon_5)}{\phi_{\mathrm{p}}\tilde{T}(1+\epsilon_6)} C\tau = \frac{\phi_{\mathrm{idle}}}{\phi_{\mathrm{p}}} C\tau(1-\epsilon_7)$ for some $\epsilon_7$.

Finally, note that liveness of each protocol instance guarantees liveness of the parallel chains construction. As per the transaction distribution rule described in Appendix C, each transaction belongs to a particular protocol instance. By the liveness of each protocol instance, any transaction input to all honest nodes in time slot $t$, is included in $\mathrm{d}C_i(t)^{\lceil T_{\mathrm{conf}}}$ for $t' \geq t+\gamma+T_{\mathrm{conf}}$ (see Proof of Lemma 9 in Appendix D.2) and all nodes $i$ for which the corresponding protocol instance is primary. Moreover, all honest nodes download confirmed secondary chains within $\tilde{T}$ delay. Therefore, $\Pi_{\mathrm{pc}}^{\rho,\tau,T_{\mathrm{conf}},m}$ satisfies liveness with total latency $\gamma + T_{\mathrm{conf}} + \tilde{T} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$. □

## F CONFLUX INCLUSION RULE

In order to prevent the throughput from vanishing as the resilience $\beta$ approaches $1/2$, we incorporate a modified version of the block inclusion rule from Conflux [38] (also used in [23]). In addition to the hash of the parent block, the header of a block $b$ also contains references to (hashes of) at most $R$ blocks which have time slots earlier than $b$ and are neither in the prefix nor are referenced by any blocks in the prefix of $b$. Moreover, in each chain, at most one block from each time slot may be referred. An honest block producer chooses to include the $R$ newest (by time slot) fully downloaded blocks in their view that satisfy the above criteria. The parameter $R$ is to be determined below. Blocks containing references that do not follow the above criteria will be considered invalid. The consensus protocol still uses the longest chain rule.

Note that downloading and validating a block now requires (in addition to downloading the block itself) downloading the content of all blocks in its prefix and all blocks referenced by blocks in the prefix. Unlike [23], we do not consider the reference links to be transitive as this would blow up the number of referred blocks to be downloaded. The output ledger of a node $i$ in slot $t$ (i.e. $\mathsf{LOG}_i^t$) will be formed by considering its truncated longest chain (i.e. $\mathrm{d}\mathcal{C}_i(t)^{\lceil T_{\mathrm{conf}}}$) and inserting blocks referred by a block $b$ between the parent of $b$ and $b$, in increasing order of their time slot. This may result in some transactions becoming invalid due to conflicting transactions appearing before them in the ledger. Such transactions would be removed (*sanitized*) while obtaining the ledger.

## F.1 Security

For security of the inclusive protocol, it is enough to set the time slot size to be $\tau = \Delta_{\mathrm{h}} + \frac{KR}{C}$ (previously $\Delta_{\mathrm{h}} + \frac{K}{C}$) where $K$ is set according to Theorem 1 (we do the analysis below for the freshest block download rule, but it can be done for the equivocation avoidance download rule as well). Since each block contains at most $R$ references, the number of blocks to be downloaded in the prefix of any honest freshest block increases at most by a factor of $R$. By setting the slot size as above, we ensure that the honest block proposed in every uniquely successful slot is downloaded (along with its prefix and references therein) within the same slot.

## F.2 Single Chain Throughput

**Lemma 12.** *If $R = \gamma p(1 - \epsilon_7)$, there exists a constant $T_4$ such that for any honest node i and time slots $t_1, t_2 \geq t_1 + T, t \geq t_2 + T_{\mathrm{conf}}$ with $T \geq T_4$, $\mathsf{LOG}_i^t$ contains at least $\theta_{\mathrm{inc}} T(1 - \epsilon_8)$ blocks proposed by honest nodes in slots $(t_1, t_2]$, with probability at least $1 - \exp(-\alpha_4 T)$, where $\theta_{\mathrm{inc}} = p_{\mathrm{U}}$.*

Lemma 12 indicates that the average throughput of a single instance of the inclusive protocol is at least $\frac{\theta_{\mathrm{inc}}}{\tau}$ blocks per second.

To prove Lemma 12, we only need to show that every honest block from a uniquely successful slot is included in the longest chain of every node either directly on the chain or through a reference. This will be achieved by setting $R$ to be large enough so that in any interval of slots with $R$ block production opportunities, at least one honest block is included in the longest chain. Then such an honest block would include references to the $R$ most recent blocks which would collectively include (at least) all honest blocks from uniquely successful slots.

Proof. From the security analysis (Lemma 10), we have that $\Pr\left[\mathsf{FreqPivots}_\gamma\right] \geq 1 - \mathrm{negl}(\kappa)$ where $\mathsf{FreqPivots}_\gamma$ is the event

$$\forall t\colon \exists t' \in (t, t+\gamma]\colon\ \mathsf{Pivot}(t') \wedge \mathsf{Unique}(t'). \tag{71}$$

Moreover, we have shown in Lemma 9 that the honest block proposed in a unique pivot slot remains in the longest downloaded chain of every honest node. This satisfies our requirement. Thus, we need to set $R = \gamma p(1 + \epsilon_7)$ so that there are at most $R$ uniquely successful slots between two pivot slots, i.e.

$$\forall t\colon\ \Pr\left[\mathcal{U}(t, t+\gamma] > R\right] \geq 1 - \exp\left(\frac{\epsilon_7^2}{\epsilon_7 + 2}\gamma p\right). \tag{72}$$

Therefore, the lemma holds under $T_4 = \gamma$ and the conditions from Lemma 10. □

## F.3 Parallel Chains Throughput

We still have honest nodes idle (not downloading any blocks) in at least $\phi_{\mathrm{idle}} \geq \frac{1-p}{2}$ fraction of slots. The average bandwidth required to download a confirmed chain still remains at $p$ blocks per slot. Therefore, we can increase the total throughput by constructing $m = 1 + \frac{\phi_{\mathrm{idle}} C}{\phi_{\mathrm{p}}/\tau}$ parallel chains resulting in aggregate throughput

$$\begin{aligned}
\mathrm{TP}_m &= \left(1 + \frac{\phi_{\mathrm{idle}}}{\phi_{\mathrm{p}}} C\tau\right) \frac{\theta_{\mathrm{inc}}}{\tau} \\
&\geq \frac{(1-p)p_{\mathrm{U}}}{2p} C \qquad \text{(for the ‘freshest block’ rule)} \\
&\geq \frac{(1-p)}{4} C \text{ blocks per second.}
\end{aligned} \tag{73}$$

This is a constant fraction of the capacity $C$ which does not vanish as $\beta \longrightarrow 1/2$.