Securing Proof-of-Stake Nakamoto 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

Satoshi Nakamoto's Proof-of-Work (PoW) longest chain (LC) protocol was a breakthrough for Internet-scale open-participation consensus. Many Proof-of-Stake (PoS) variants of Nakamoto's protocol such as Ouroboros or Snow White aim to preserve the advantages of LC by mimicking PoW LC closely, while using PoS for Sybil resistance. Previous works have proven these PoS LC protocols secure assuming all network messages are delivered within a bounded delay. However, this assumption is not compatible with PoS when considering bandwidth constraints in the underlying communication network, because PoS enables the adversary to spam the network with equivocating blocks, which is impossible in PoW. The bandwidth constraint necessitates that nodes choose carefully which blocks to spend their limited download budget on. We show that 'download along the longest header chain', a natural download rule for PoW LC, emulated by PoS variants, is insecure for PoS LC. Instead, we propose 'download towards the freshest block' and prove that PoS LC with this download rule 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 our PoS LC protocol in parallel, we obtain a PoS consensus protocol with improved worst-case throughput, even in the presence of a spamming adversary. Our result can be viewed as a first step towards the co-design of consensus and network layer protocols.

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 non-trivial by an adversary who has some control over message delays and who 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 [34], 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 way of using *proof-of-work* (PoW) puzzles as a Sybil resistance mechanism [16, 27]. 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 [20, 35]. 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, 12, 29] and Sleepy Consensus [11, 36] 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, 12], 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.

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

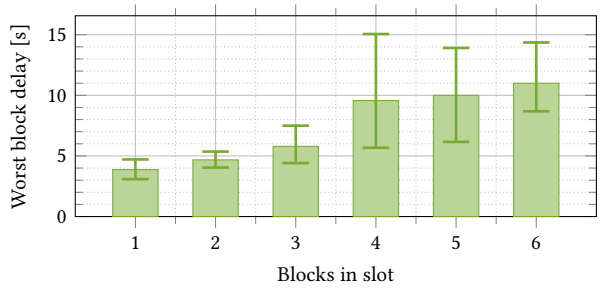

Figure 1: Worst block propoagation delay (10-th percentile, mean, 90-th percentile) in a slot measured in experiments with Cardano's Ouroboros implementation. The delay increases as the number of blocks produced and broadcast in the network per slot is increased, showing that network delay cannot be treated as independent of network load. Details of the experimental setup are given in Appendix C.1.

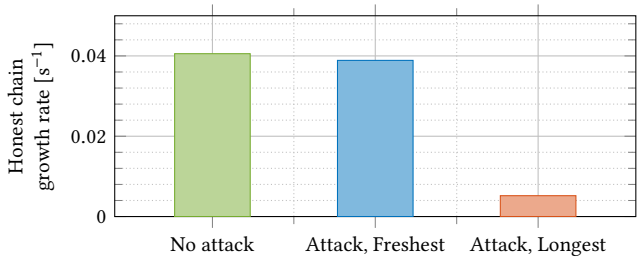

Figure 2: The honest chain growth rate in three scenarios: without spamming attack; under attack and downloading the longest header chain first (priority rule in Cardano's block download logic); under attack and downloading the freshest block first (introduced in this work). Details of the experimental setup are given in Section 5.2. For a trace of the chain growth in the same experiment, see Figure 6.

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 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 [12, 15, 36] shows that this difference is immaterial in the synchronous network model where message propagation delay between honest nodes is controlled by the adversary, but below a known upper bound Δ. 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, irrespective of whether few or many messages are being transmitted simultaneously. 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 [14]. 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 [24]. 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.

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 Δ_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 [9, 10], 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 [9, 10, 13] for chain selection [26] and block downloads [25]. As long as the block production rate is low relative to the

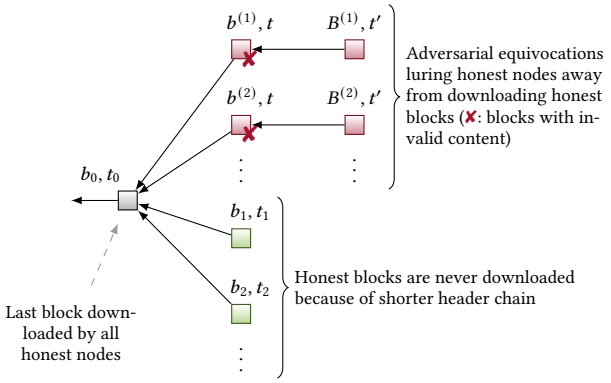

Figure 3: In PoS LC with 'download along the longest header chain' rule, an adversary can stall consensus indefinitely if it has two block production opportunities $t < t'$ at which it creates infinitely many equivocating chains $b_0 \leftarrow b^{(i)} \leftarrow B^{(i)}$ where $b^{(i)}$ have invalid 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.

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 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. The impact of this attack is seen in our experiments with a PoS LC node implementing this download rule (Figure 2).

Mitigation Heuristics. In practice, implementations show awareness of and attempt to mitigate such and similar 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 [9, 10, 13]. By the time an honest node detects misbehavior, however, the adversary has already wasted some of the node's resources. Furthermore, an adversary can boost the impact of its attack by creating more Sybil network peers (recall that there is no relation between consensus stakeholders 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 [38–40].

Another common approach, exemplified by PoS Ethereum, is for nodes to propagate only the first copy they have received of a block production opportunity [2]. As a result, an equivocating adversary leaves the network in a state of (at least temporary) split view. An honest block can then get built on top of blocks that parts of the network have not received before. Full consideration of the new honest block is delayed by the additional time required to locate and fetch its prefix from peers. This side effect of the heuristic is neither covered by the synchronous network model nor part of many formal consensus protocol security analyses.

A related proposal is to blocklist equivocating stakeholders and ignore their blocks or even consider them invalid. However, by the time an equivocation is detected, some honest nodes might have already taken consensus decisions based on earlier blocks produced by that stakeholder, and it is not straightforward to reconcile honest nodes' views without undoing consensus decisions.

While particular heuristics may or may not be susceptible to spamming/equivocation attacks depending on implementation details, a rigorous analysis is often missing. Furthermore, this discussion suggests fundamental flaws with the 'download along the longest header chain' philosophy and prompts a search for a conceptually simple and provably secure alternative. In the attack on the 'download along the longest header chain' rule described in Figure 3, we observe that even though new honest blocks are being proposed, the highest download priority is given to the older adversarial blocks. If nodes downloaded the fresher blocks proposed in more recent time slots $t_1, t_2, \ldots$ instead, then this attack would not succeed. This observation motivates the following 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.*, at every instant an honest node identifies the block from the most recent block production opportunity based on the header information, and downloads any missing blocks in the prefix and including that freshest block. This rule ensures that every now and then when an honest node proposes a block, other honest nodes make progress towards downloading that block and its prefix. This way, honest nodes have a chance to produce blocks extending it in the future, and align their block production efforts toward a particular chain, which is arguably the key stepping stone of prior security analysis techniques [15, 36] on which we build. In particular, our download rule avoids the attack of Figure 3 in which no honest block will ever be downloaded. In fact, the honest chain's growth rate remains unaffected by this spamming attack under the freshest block download rule (Figure 2).

Our 'download towards the freshest block' rule for PoS LC has another interesting property in comparison to the 'download along the longest header chain' rule of PoW LC. 'Download along the longest header chain' uses no information about the timing of blocks, which is natural as such information is not reliably available in PoW LC. Yet, this rule does not work when simply copied for PoS LC. In contrast, 'download towards the freshest block' works for PoS LC and leverages the block production timing information which is particular to PoS LC but not reliably available in PoW LC.

Our Contributions. We prove that LC PoS with our drop-in 'download towards the freshest block' rule is secure in networks with bandwidth constraints in which the adversary can spam the network with equivocating blocks at an arbitrary rate, the adversary can withhold blocks, and the adversary can release blocks with invalid content that honest nodes discard after downloading. Our analysis extends and refines proof techniques for PoS LC in synchronous networks without bandwidth constraint [36]. Moreover, spamming becomes an even more serious concern in high throughput systems such as [6, 18, 43] where honest messages already consume substantial bandwidth, leaving little margin to deal with spam. We show that parallel composition of multiple instances of PoS LC with our download rule (inspired by [19]) 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, 7]. Eclipse attacks on peer-to-peer networks, where an adversary uses several IP addresses to occupy all connections maintained by a victim node, have been studied in [23, 38–40] and in the context of Bitcoin in [8]. The temporary shutdown of a PoS LC protocol Solana [42] in September 2021, was reportedly due to an increase in the transaction load in the network, and "lack of prioritization of network-critical messaging caused the network to start forking" [33]. These examples indicate that network-level threats are a serious concern and raise the question of careful co-design of consensus and network layer protocols.

The need for careful modelling of bandwidth constraints in the context of high throughput systems was identified in [6, 19]. For PoW, [6, 14, 41] 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, [19] extends this 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 [19, Theorem 1] still assumes a bounded network delay. This assumption is only shown to hold when the adversary does not corrupt any nodes and does not send or delay any messages [19, 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 [19] 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 security of PoS LC under bandwidth constraints, our analysis builds on tools from several years of security analysis for LC protocols [5, 12, 15, 20, 22, 35–37], particularly the concept of pivots [36] (cf. Nakamoto blocks [15]).

Algorithm 1 Idealized PoS LC consensus protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ (helper functions/procedures: Appendix A, $\mathcal{F}^{\rho}_{\text{headertree}}$: Algorithm 2)

on INIT(genesisHeaderChain, genesisTxs)
  $\mathsf{h}\mathcal{T}, \mathsf{d}C \leftarrow \{\text{genesisHeaderChain}\}, \text{genesisHeaderChain}$ ▷ *Initialize header tree and longest downloaded chain with genesis block*
  $\text{blkTxs}[\mathsf{d}C] \leftarrow \text{genesisTxs}$ ▷ *Default of* blkTxs *entries:* unknown
on RECEIVEDHEADERCHAIN(C)
  assert $\mathcal{F}^{\rho}_{\text{headertree}}$.VERIFY($C$) = true ▷ *Verify block production lottery and header block chain structure, see Algorithm 2*
  $\mathsf{h}\mathcal{T} \leftarrow \mathsf{h}\mathcal{T} \cup \text{prefixChainsOf}(C)$
  $\mathcal{Z}$.BROADCASTHEADERCHAIN(C)
on RECEIVEDCONTENT(C, txs) ▷ *Callbacks are queued until* $C \in \mathsf{h}\mathcal{T}$ *and* $\text{blkTxs}[C'] \notin \{\text{unknown}, \texttt{invalid}\}$ *for all prefix chains* $C' \preceq C$
  assert C.txsHash = Hash(txs) ▷ *Verify* txs *match header*
  if txsAreSemanticallyValidWrtPrefixesOf(C, txs) ▷ *Store and relay* txs *for block if* txs *are valid, otherwise mark block as* invalid
    $\text{blkTxs}[C] \leftarrow \text{txs}$
    $\mathcal{Z}$.UPLOADCONTENT(C, txs)
  else
    $\text{blkTxs}[C] \leftarrow \texttt{invalid}$
  $\mathcal{T}' \leftarrow \mathsf{h}\mathcal{T} \setminus \{C' \in \mathsf{h}\mathcal{T} \mid \text{blkTxs}[C'] \in \{\text{unknown}, \texttt{invalid}\}\}$
  $\mathsf{d}C \leftarrow \text{longestChain}(\mathcal{T}')$ ▷ *Update longest downloaded chain*
on SCHEDULECONTENTDOWNLOAD() ▷ *Called when download idle*
  $\mathcal{T}' \leftarrow \{C \in \mathsf{h}\mathcal{T} \mid \forall C' \preceq C : \text{blkTxs}[C'] \neq \texttt{invalid}\}$ ▷ *Discard chains with invalid blocks*
  $C \leftarrow \arg\max_{C' \in \mathcal{T}'} C'.\text{time}$ ▷ *Find freshest block*
  $C \leftarrow \arg\min_{C' \in \{C'' \preceq C \mid \text{blkTxs}[C''] = \text{unknown}\}} |C'|$ ▷ *Find first block with missing content in prefix of freshest block, if any; else* $C = \bot$
  if $C \neq \bot$
    $\mathcal{Z}$.REQUESTCONTENT(C) ▷ *If the requested content is available, callback* RECEIVEDCONTENT(.) *will be triggered by* $\mathcal{Z}$
for time slots $t \leftarrow 1, ..., T_{\text{h}}$ of duration τ
  $\text{txs} \leftarrow \mathcal{Z}$.RECEIVEPENDINGTXSSEMANTICALLYVALIDWRT($\mathsf{d}C$)
  if $C' \neq \bot$ **with** $C' \leftarrow \mathcal{F}^{\rho}_{\text{headertree}}$.EXTEND($t, \mathsf{d}C$, txs) ▷ *Check eligibility to produce a new block, and if so do so, see Algorithm 2*
    $\mathcal{Z}$.UPLOADCONTENT(C', txs)
    $\mathcal{Z}$.BROADCASTHEADERCHAIN(C')
  $\mathcal{Z}$.OUTPUTLEDGER($\mathsf{d}C^{\lceil T_{\text{conf}}}$) ▷ *Ledger of node* i *at time* t*:* LOG_i^t

Outline. We recapitulate the details of Nakamoto's LC protocol and introduce our formal model for bandwidth constrained networks in Section 2. We provide an overview of the security argument of PoS LC with the 'download towards the freshest block' rule under bandwidth constraint in Section 3. Technical details of the security analysis are provided in Section 4. We present experimental evidence for the robustness and superior performance of our download rule in Section 5. Finally, we sketch in Section 6 how to use PoS LC with our download rule as a building block to obtain a consensus protocol with high worst-case throughput in bandwidth constrained networks.

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 easily extended to the case of heterogeneous and dynamic stake using tools from [11, 12]. Nodes' cryptographic identities are common knowledge.

Algorithm 2 Idealized functionality $\mathcal{F}^{\rho}_{\text{headertree}}$: block production lottery and header block chain structure (cf. Appendix A)

1: **on** INIT(genesisHeaderChain, numParties)
2: $N \leftarrow$ numParties
3: $\mathcal{T} \leftarrow$ {genesisHeaderChain}
4: **on** ISLEADER(P, t) **from** $\mathcal{A}$ or $\mathcal{F}^{\rho}_{\text{headertree}}$
5: **if** lottery$[P, t] = \bot$
6: lottery$[P, t] \xleftarrow{\$}$ (true with probability ρ/N, else false)
7: **return** lottery$[P, t]$
8: **on** EXTEND(t', C, txs) **from** party P **at** time slot t
9: $C' \leftarrow C \,\|\, \text{newBlock}(\text{time} = t', \text{party} = P, \text{txsHash} = \text{Hash}(\text{txs}))$
10: **if** $C \in \mathcal{T} \wedge$ ISLEADER$(P, t') =$ true $\wedge\ C.\text{time} < t' \leq t$
11: $\mathcal{T} \leftarrow \mathcal{T} \cup \{C'\}$
12: **return** C'
13: **return** $\bot$
14: **on** VERIFY(C)
15: **return** $C \in \mathcal{T}$

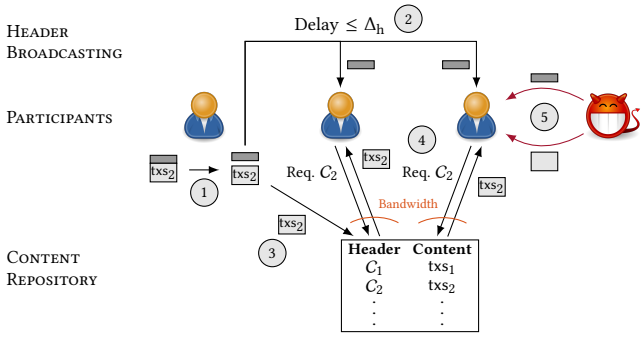

Figure 4: In our model, block headers are propagated with a known delay upper bound Δ_h, while block content is subject to a bandwidth constraint. ① An honest node produces a new valid block, consisting of header and content. ② Block headers are broadcast and arrive at honest nodes within at most Δ_h delay. ③ Block content is submitted to an idealized 'repository'. ④ A hash of the corresponding block content is included in the block header. Upon request, the content of a certain block is obtained from the 'repository', 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.

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 β of all nodes, where β 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. Thus, for simplicity, we have assumed that all nodes are always *awake*. Our analysis builds on techniques from [36] and the refined machinery therein can be used to extend our analysis to the setting of asleep/awake honest nodes.

Protocol Main Features. Pseudocode of an idealized PoS LC Nakamoto consensus protocol employing the 'download towards the freshest block' rule is provided in Algorithm 1 (cf. [36, Figure 3]). 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 2 (cf. [36, Figure 2]). An index of the helper functions used in the pseudocode is provided in Appendix A. 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, 12, 29] and the Sleepy Consensus [11, 36] family. In the main loop of the PoS LC protocol (Algorithm 1, lines 23ff.) the node attempts in every time slot (which is of duration τ) to produce a new block containing transactions txs and to extend the longest downloaded chain (denoted $\mathsf{d}C$) 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. As 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 Δ_h that is common knowledge. Once an honest node receives a new valid block header chain (Algorithm 1, lines 4ff.), the node adds it to its local header tree $\mathsf{h}\mathcal{T}$. Block content uploaded via $\mathcal{Z}$.UPLOADCONTENT(.) is kept by $\mathcal{Z}$ in an idealized repository. 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 (Algorithm 1, lines 8ff.). Each node has a bandwidth constraint of C blocks per second. With a slot duration of $\tau > \Delta_h$ and $\bar{C} \triangleq (\tau - \Delta_h)/C$, all nodes receive the headers of blocks proposed in the current slot, and thereafter $\mathcal{Z}$ delivers at most $\bar{C}$ block contents requested from the repository to each honest node per time slot.[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 any of its descendants in the future. Finally, the node updates its longest downloaded chain.

'Download Towards The Freshest Block' Rule. Whenever no block content download is in progress, the 'download towards the freshest block' rule is used to determine which block's content to request from $\mathcal{Z}$ next (Algorithm 1, lines 17ff.). For this purpose, the header tree $\mathsf{h}\mathcal{T}$ 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. The callback RECEIVEDCONTENT(.) is triggered if and only if the requested block content is found in $\mathcal{Z}$'s repository.

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 chain C); extending any chain using past block production opportunities as long as the purported block production

[1] Unlike [19], we do not model the upload bandwidth because honest nodes only send very few messages in our protocol.

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 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 2, line 6) 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 [36]; against adaptive adversaries, a verifiable random function (VRF) is used [29]. Although the ideal functionality $\mathcal{F}^{\rho}_{\text{headertree}}$ relies on the knowledge of N to tune the threshold ρ/N, in PoS realizations such as in [12] 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 2, line 9) is established using digital signatures.

3 HIGH LEVEL SECURITY ARGUMENT

The security proof is divided into two sections. In Section 4.3, we prove that the protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ achieves safety and liveness whenever the random sequence specifying the block proposal opportunities satisfies two properties. In the next section, we show that these properties hold throughout the execution time horizon with high probability (detailed proof in Appendix E.2). Our proof follows the techniques from [36] and [15]. 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 Δ. In our case, we first prove that if the bandwidth is sufficiently larger than the block production rate and majority of the nodes are honest, then a large fraction of honestly proposed blocks are downloaded by all nodes within a bounded delay, with high probability.

To this effect, we consider *unique* time slots, in which there is exactly one honest block proposal (any other slots with block proposals are called *adversarial*). In a unique slot, the proposed block is the unique freshest block, hence all honest nodes download towards it. If the prefix of the freshest block contains at most $\bar{C}$ blocks that have not been downloaded yet, then the freshest block can be downloaded within one time slot. In Proposition 2, we prove inductively as follows, that the blocks proposed in all unique slots are downloaded by all nodes within one slot. Assuming that this hypothesis holds up to a certain time, the only blocks in the prefix of the next freshest block that are yet to be downloaded, belong to adversarial slots. Moreover, this prefix must be the longest chain in the view of the honest node that proposed the freshest block, and this requires adversarial slots to outnumber unique slots over some interval. In Definition 4, we define a property wherein all such intervals contain less than $\bar{C}$ adversarial slots. Since the freshest block builds on a valid chain, its prefix can contain at most one block from each time slot, therefore the number of blocks to be downloaded is bounded by the number of adversarial slots, which in turn in less than $\bar{C}$ as per the above property. This shows that the freshest block can be downloaded within one slot, and concludes the induction proof. In Lemma 3, we show that the property from Definition 4 holds with high probability, if the rate of unique slots exceeds that of adversarial slots, and the per slot bandwidth $\bar{C}$ is sufficiently larger than the per slot block proposal rate ρ. The download of blocks from unique slots implies that the minimum chain growth rate of all honest nodes is at least the rate of unique slots (Proposition 1).

Following [36], we define a *pivot* as a slot such that in every interval containing the pivot, there are more unique slots than adversarial slots or there are no adversarial slots (see Definition 2). In Lemma 4, we show that when a slot is both unique and a pivot, the block proposed therein stays in the longest downloaded chain of every honest node thereafter, irrespective of the adversary's actions. In short, this is because the honest nodes' chains grow at least at the rate of unique slots, hence the adversary requires a greater number of adversarial slots (than unique slots) to produce a different longest chain, which contradicts the pivot condition. Finally, since the block from a unique pivot slot stays in all nodes' longest chains, this ensures that honest transactions will be included in this block or its prefix, and the transactions in the prefix of this block remain in every honest node's ledger forever. Therefore, the occurrence of unique pivots leads to safety and liveness of the ledger output by the consensus protocol (proved in Lemma 1). In Lemma 2, we show that with high probability, unique pivot slots occur frequently if the rate of unique slots exceeds that of adversarial slots. This proof follows the technique in [36].

Finally in Theorem 1, we identify the parameter values under which the protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ is secure for a given desired resilience β and security parameter κ. For the rate of unique slots to exceed the rate of adversarial slots, we require that the rate of block production ρ be bounded as a function of β, so that most slots with honest block proposals are also unique slots. A similar constraint exists in the synchronous model [12, 15, 36] that the product of the block production rate and network delay Δ is bounded by a function of β. Next, we require that the per slot bandwidth $\bar{C} = \Omega(\kappa)$ so that the property in Definition 4 is satisfied and in turn blocks from unique slots are downloaded within one slot with high probability. This implies that the time slot $\tau = \Delta_{\text{h}} + \frac{\bar{C}}{C} = \Omega(\kappa)$. This is similar to [19] where under a bandwidth constrained model, the probabilistic delay

[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.

bound scales with the security parameter. Finally, the confirmation time $T_{\text{conf}} = \Omega(\kappa^2)$ slots so that at least one unique pivot slot occurs within this time. This dependence is similar to that in [36] in the synchronous model.

Duration Of Time Slots. In prior analyses of PoW and PoS LC in the bounded delay model, we require loners [12, 15, 37] or convergence opportunities [36] (which are honest block proposals with no other block proposals in Δ slots before or after them) to outnumber adversarial blocks. Otherwise, a centralized adversary can keep running lotteries for the next block immediately after proposing one block, while honest nodes wait for their blocks to be propagated and downloaded by the other nodes. In our protocol, we attempt to bridge this difference between honest and adversarial opportunities by increasing the slot duration so that all honest nodes download the freshest block and its prefix by the end of the time slot (with high probability). Then the adversary does not get any more block production opportunities while honest nodes wait on their download. Thus, we only require unique slots, which is a weaker requirement than loners, hence allows a larger block production rate ρ. This makes the analysis more similar to a lock-step model, as in [20], and thus simplifies the analysis as well.

4 SECURITY PROOF

4.1 Definitions

The PoS LC protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ has three parameters. The length of each time slot is τ seconds, the average number of nodes eligible to propose a block per time slot is ρ, and the confirmation latency is T_{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 will fix $\tau = \Delta_{\text{h}} + \frac{\bar{C}}{C}$ such that each honest node can download the content of $\bar{C}$ blocks in one time slot after receiving the headers proposed in that slot. The adversary controls β fraction of the stake. We denote by κ the security parameter.

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. An execution $\mathcal{E}^{\beta,T_{\text{h}}}$ of time horizon T_{h} is specified by the sequence $\{H_t, A_t\}_{t \le T_{\text{h}}}$. 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 $H_t \overset{\text{i.i.d.}}{\sim} \text{Poisson}((1-\beta)\rho)$ and $A_t \overset{\text{i.i.d.}}{\sim} \text{Poisson}(\beta\rho)$, all independent of each other.

Denote by $\text{d}C_i(t)$ 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 $L_i(t) = |\text{d}C_i(t)|$ and $L_{\min}(t) = \min_i L_i(t)$. At the end of each slot, honest node i outputs the ledger $\text{LOG}_i^t = \text{d}C_i(t)^{\lceil T_{\text{conf}}}$, which consists of a list of transactions as ordered in all blocks in $\text{d}C_i(t)$ with time slot up to $t - T_{\text{conf}}$.

An execution of the consensus protocol is *secure* if it satisfies:

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

A consensus protocol is *secure* if it satisfies safety and liveness with high probability over its executions.

Definition 1. A slot t is called *unique* if $A_t = 0$ and $H_t = 1$. A slot t is called adversarial if either $A_t > 0$ or $H_t > 1$.

Define the predicates $\text{Unique}(t)$ to be true iff slot t is unique and $\text{Adv}(t)$ to be true iff slot t is adversarial.

We refer to a slot t as a block proposal slot if $A_t + H_t > 0$. For $s > r$, denote by $\mathcal{B}(r,s]$ the number of block proposal slots in the interval $(r,s]$, denote by $\mathcal{U}(r,s]$ the number of unique slots in the interval $(r,s]$ and denote by $\mathcal{A}(r,s]$ the number of adversarial slots in $(r,s]$.

$$\mathcal{B}(r,s] \triangleq \sum_{t=r+1}^{s} \mathbb{1}\{A_t + H_t > 0\}, \tag{1}$$

$$\mathcal{U}(r,s] \triangleq \sum_{t=r+1}^{s} \mathbb{1}\{\text{Unique}(t)\}, \tag{2}$$

$$\mathcal{A}(r,s] \triangleq \sum_{t=r+1}^{s} \mathbb{1}\{\text{Adv}(t)\}. \tag{3}$$

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:

$$p \triangleq \Pr[A_t + H_t > 0] = 1 - e^{-\rho}, \tag{4}$$

$$p_{\text{U}} \triangleq \Pr[\text{Unique}(t)] = (1-\beta)\rho e^{-\rho}, \tag{5}$$

$$p_{\text{A}} \triangleq \Pr[\text{Adv}(t)] = p - p_{\text{U}} \tag{6}$$

Definition 2. A *pivot* is a slot t such that

$$\forall (r,s] \ni t: \ (\mathcal{U}(r,s] > \mathcal{A}(r,s]) \vee (\mathcal{A}(r,s] = 0). \tag{7}$$

The predicate $\text{Pivot}(t)$ is true iff t is a pivot.

We will say that a slot t is a unique pivot slot iff $\text{Pivot}(t) \wedge \text{Unique}(t)$.

Definition 3. An execution $\mathcal{E}^{\beta,T_{\text{h}}}$ satisfies $\text{FrequentPivots}_{\gamma}$ iff

$$\forall t \le T_{\text{h}} - \gamma: \exists t' \in (t, t+\gamma]: \ \text{Pivot}(t') \wedge \text{Unique}(t'). \tag{8}$$

Definition 4. An execution $\mathcal{E}^{\beta,T_{\text{h}}}$ satisfies $\text{ShortPrefixes}_{\bar{C},\gamma}$ iff

$$\forall t \le T_{\text{h}}: \max_{t' < t: \ \text{Unique}(t') \wedge (\mathcal{A}(t',t] \ge \mathcal{U}(t',t])} \mathcal{A}\big(t', t\big] < \bar{C}. \tag{9}$$

Definition 5. Define the predicate $\text{ChainGrowth}_{(r,s]}$ to be true iff for all m unique slots $t_1, \ldots, t_m \in (r,s]$, the block proposed in t_j is downloaded by all honest nodes by the end of slot t_j.

Definition 6. A *great block* is an honest block b from slot t such that for all honest nodes i and $t' \ge t$: $b \in \text{d}C_i(t)$.

Note that the genesis block, which is defined to be from the unique slot $t = 0$ and 'downloaded' by honest nodes from the start, is a great block.

4.2 Proof Overview

Lemma 1. *The protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ with $T_{\text{conf}} = \gamma$ achieves safety and liveness with $T_{\text{live}} = 2\gamma$ in any execution $\mathcal{E}^{\beta,T_{\text{h}}}$ which satisfies $\mathsf{FrequentPivots}_{\gamma}$ and $\mathsf{ShortPrefixes}_{\bar{C},\gamma}$.*

Lemma 1 is proved in Section 4.3.

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

$$\Pr\left[\mathcal{E}^{\beta,T_{\text{h}}} \colon \neg\mathsf{FrequentPivots}_{\gamma}\right] \leq 2T_{\text{h}}^2 e^{-\alpha_1 p w} + T_{\text{h}} e^{-\alpha_3 \gamma / w}, \quad (10)$$

where $\alpha_1 = \epsilon_1^2/36$, $w > \max\left\{\frac{2}{\alpha_1 p}\ln\left(\frac{4(1+e^{-\alpha_1 p})}{(1-e^{-\alpha_1 p})^2}\right), \frac{2\ln(\sqrt{2}T_{\text{h}})}{\alpha_1 p}\right\}$ and α_3 is a constant that depends on ϵ_1 and ρ.

Lemma 3. *If $p_{\text{U}} = \frac{1}{2}p(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$ and $\bar{C} = p_{\text{A}}T(1+\epsilon_2)$ for some $\epsilon_2 > 0$ and $T > \max\left\{\frac{2}{1-e^{-\alpha_1 p}}, \frac{2\ln(\sqrt{2}T_{\text{h}})}{\alpha_2 p}\right\}$, then*

$$\Pr\left[\mathcal{E}^{\beta,T_{\text{h}}} \colon \neg\mathsf{ShortPrefixes}_{\bar{C},\gamma}\right] \leq 2T_{\text{h}}^2 e^{-\alpha_2 p T}, \quad (11)$$

where $\alpha_2 = \min\left\{\alpha_1, \frac{\epsilon_2^2}{\epsilon_2+2}\frac{p_{\text{A}}}{p}\right\}$.

Lemma 2 and Lemma 3 are proved in Appendix E.2.

Theorem 1. *Except with probability $\text{negl}(\kappa)$ over the executions $\mathcal{E}^{\beta,T_{\text{h}}}$, the protocol $\Pi^{\rho,\tau,T_{\text{conf}}}$ with parameters ρ such that $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$, $\tau = \Omega(\kappa+\ln T_{\text{h}})$ and $T_{\text{conf}} = \Omega\left((\kappa+\ln T_{\text{h}})^2\right)$, achieves safety and liveness with $T_{\text{live}} = \Omega\left((\kappa+\ln T_{\text{h}})^2\right)$.*

Proof. From Lemma 1, safety and liveness holds except with probability

$$\Pr\left[\mathcal{E}^{\beta,T_{\text{h}}} \colon \neg\mathsf{FrequentPivots}_{\gamma} \vee \neg\mathsf{ShortPrefixes}_{\bar{C},\gamma}\right] \leq 2T_{\text{h}}^2 e^{-\alpha_1 p w} + 2T_{\text{h}}^2 e^{-\alpha_2 p T} + T_{\text{h}} e^{-\alpha_3 \gamma / w} \quad (12)$$

by Lemma 2, Lemma 3 and a union bound. Let $\kappa' = \kappa + \ln T_{\text{h}}$. Given the desired fraction of adversarial stake β to defend against, pick ρ such that $p_{\text{U}} = \frac{1}{2}p(1+\epsilon_1)$, *i.e.*, $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$. Next, we pick w such that $w = \frac{2\ln(\sqrt{2}T_{\text{h}})+\Omega(\kappa)}{\alpha_1 p}$. This ensures that the probability $2T_{\text{h}}^2 e^{-\alpha_1 p w}$ corresponding to having more adversarial than honest slots in some interval of size at least w, is $\text{negl}(\kappa)$. This results in $w = c_1\kappa'$ where c_1 depends on ρ and ϵ_1.

Next, we choose $T = \frac{2\ln(\sqrt{2}T_{\text{h}})+\Omega(\kappa)}{\alpha_2 p} = c_2\kappa'$ so that the probability $2T_{\text{h}}^2 e^{-\alpha_2 p T}$ corresponding to $\neg\mathsf{ShortPrefixes}_{\bar{C},\gamma}$, is $\text{negl}(\kappa)$. Hence, the bandwidth required in blocks per slot is $\bar{C} = p_{\text{A}}T(1+\epsilon_2) = c_3\kappa'$. In order to interpret this in terms of bandwidth in blocks per second, we must set $\tau = \Delta_{\text{h}} + \frac{\bar{C}}{C} \approx \frac{c_3}{C}\kappa'$, allowing nodes to download $\bar{C}$ blocks per slot with a constant capacity C blocks per second.[3]

Finally, we pick γ so that the probability $T_{\text{h}} e^{-\alpha_3\gamma/w}$ corresponding to not finding a pivot slot in some interval of γ slots, is $\text{negl}(\kappa)$. Therefore we get $\gamma \geq \frac{\ln(T_{\text{h}})+\Omega(\kappa)}{\alpha_3} w = c_4\kappa'^2$. Finally, $T_{\text{conf}} = \gamma = c_4\kappa'^2$ time slots and $T_{\text{live}} = 2\gamma$ accordingly. Note that T_{conf} corresponds to $\gamma\tau = c_5\kappa'^3$ in units of real time. □

[3] Δ_{h} and C are parameters that depend on the physical network conditions, and hence these are independent of the security parameter.

4.3 Combinatorial Analysis

Proposition 1. *Let t_0, s with m unique slots $t_1, \ldots, t_m \in (t_0, s]$ be such that $\mathsf{ChainGrowth}_{(t_0,s]}$ holds. 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 $t_j \leq t \leq s$,

$$L_{\min}(t) - L_{\min}(t_j) \geq \mathcal{U}\left(t_j, t\right] \quad (13)$$

Proof. Part (1) is easily seen by the fact that honest nodes propose on their longest downloaded chain and b_{j-1} is downloaded before b_j is proposed. 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 unique slots in $(t_j, t]$ are downloaded before the end of slot t, we conclude that $L_{\min}(t) \geq |b_{j+1}| + \mathcal{U}\left(t_j, t\right] - 1 \geq L_{\min}(t_j) + \mathcal{U}\left(t_j, t\right]$. □

Proposition 2. *If $\mathcal{E}^{\beta,T_{\text{h}}}$ satisfies $\mathsf{ShortPrefixes}_{\bar{C},\gamma}$, then $\mathsf{ChainGrowth}_{(0,T_{\text{h}}]}$ holds.*

Proof. Let $t_1, \ldots, t_m$ be the m unique slots in $(0, T_{\text{h}}]$. Let b_j be the block from t_j for $1 \leq j \leq m$. The header of b_j is received by all honest nodes within Δ_{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 $\bar{C}$ blocks whose contents have not been downloaded.

For induction, assume that $\mathsf{ChainGrowth}_{(0,t_j-1]}$ holds. Using this, we will show that $\mathsf{ChainGrowth}_{(0,t_{j+1}-1]}$ holds. For the base case, this is true for $j = 1$ since t_1 is the first unique 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 t'_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, $t'_j \leq t_j - 1$. Then,

$$|\mathrm{d}\mathcal{C}_i(t_j - 1)| \leq |b'_j| + \mathcal{A}\left(t'_j, t_j - 1\right] \quad (14)$$

since blocks after b'_j are from adversarial slots by definition of t'_j. From the assumption of $\mathsf{ChainGrowth}_{(0,t_j-1]}$ and part (1) of Proposition 1,

$$|b_{j-1}| \geq |b'_j| + \mathcal{U}\left(t'_j, t_j - 1\right]. \quad (15)$$

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 (14) and (15) that $\mathcal{A}\left(t'_j, t_j - 1\right] \geq \mathcal{U}\left(t'_j, t_j - 1\right]$. 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 $\mathcal{A}\left(t'_j, t_j - 1\right]$. Since b_j extends $\mathrm{d}\mathcal{C}_i(t_j - 1)$, the number of block contents to be downloaded in the prefix of b_j is at most $\mathcal{A}\left(t'_j, t_j - 1\right] + 1$. As per $\mathsf{ShortPrefixes}_{\bar{C},\gamma}$, this is at most $\bar{C}$ (note that $t'_j \leq t_j - 1$). Therefore, b_j is downloaded within one slot. Since there are no more unique slots in (t_j, t_{j+1}), this completes the induction step by showing that $\mathsf{ChainGrowth}_{(0,t_{j+1}-1]}$. For $j = m$, we would conclude with $\mathsf{ChainGrowth}_{(0,T_{\text{h}}]}$ as required. □

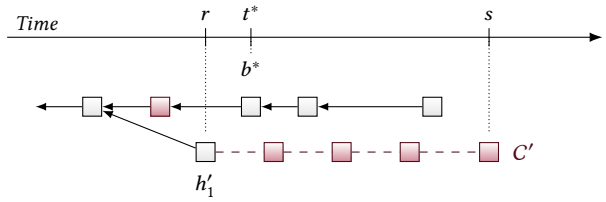

Figure 5: An illustration of one example of the blocks and time slots defined in the proof of Lemma 4. The block b^* is proposed in the unique pivot slot t^*. At the end of slot $s \geq t^*$, the chain $C' \not\ni b^*$ is the longest chain in some node's view. The last block from a unique slot in C' 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 $\mathsf{Pivot}(t^*)$.

Lemma 4. *Suppose that $\mathcal{E}^{\beta,T_h}$ satisfies $\mathsf{ShortPrefixes}_{\hat{C},\gamma}$. Let t^* be a time slot such that $\mathsf{Pivot}(t^*) \wedge \mathsf{Unique}(t^*)$. Let b^* be the block proposed in slot t^*. Then b^* is a great block.*

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

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

$$|C'| \leq |h'| + \mathcal{A}(r,s] \tag{16}$$

From Proposition 1 and Proposition 2,

$$L_{\min}(s) \geq L_{\min}(r) + \mathcal{U}(r,s]\,. \tag{17}$$

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 unique slot r and $\mathsf{ChainGrowth}_{(0,T_h]}$ holds, so $L_{\min}(r) \geq |h'|$. Using the above observations with (16) and (17), we get

$$\mathcal{U}(r,s] \leq \mathcal{A}(r,s] \tag{18}$$

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

Lemma 4 shows that the block from every unique pivot slot is a great block. Therefore, under $\mathsf{FrequentPivots}_\gamma$, every interval of γ slots brings at least one great block. To conclude with the proof of Lemma 1, one needs to show that the occurrence of great blocks leads to safety and liveness. This is done using standard techniques from [15, 36], and is shown in Appendix E.1.

5 EXPERIMENTS

5.1 Implementation Details

We implemented our PoS LC node in 800 lines of Golang.[4] For all of our experiments, the slot duration τ is set to 1 second, and the 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 [9, 10]. 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 [31]. 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 our '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 the honest nodes choose to download these spam chains.

Figure 6 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

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

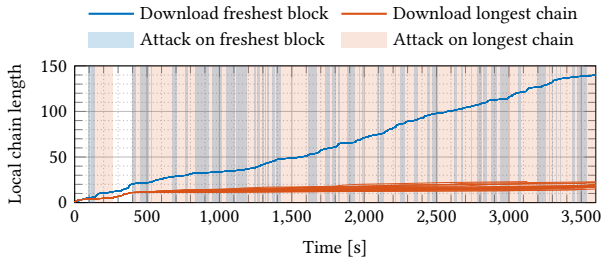

Figure 6: Traces of honest chain growth under spamming attack (cf. Figure 3) when using different download rules and an in-flight cap of 2. Each curve represents one honest node. Shaded areas represent time periods when nodes are suffering from the attack and are downloading invalid blocks. PoS LC downloading longest chain stalls. PoS LC downloading freshest blocks is robust.

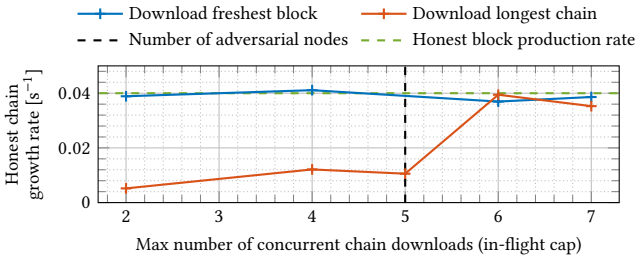

Figure 7: Honest chain growth rate under spamming attack (cf. Figure 3) while allowing concurrent block downloads from different number of peers. With in-flight cap below the number of adversarial peers, PoS LC downloading longest chain shows performance degradation; PoS LC downloading freshest block is robust.

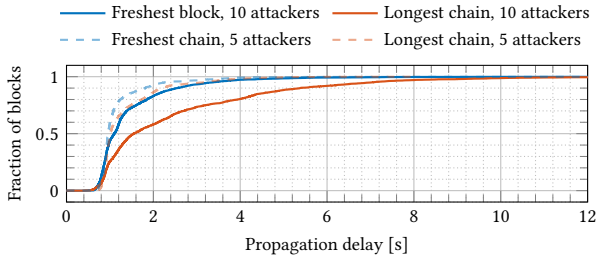

Figure 8: Empirical cumulative density function of block propagation delay under different download rules, facing different number of attackers, and an infinite in-flight cap.

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 chain 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 6) until the experiment ends.

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 6), 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 C.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 7 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 [8, 23, 38–40]: 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 7 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 8 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 our rule only slight so that security is unaffected.

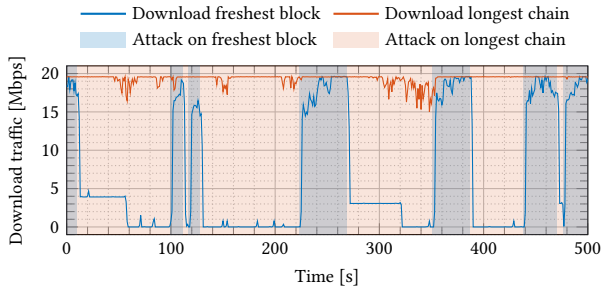

Figure 9: Traces of download traffic over a 500-second period at one of the victim nodes when using different download rules and in-flight cap of 4. Shaded areas represent time periods when the node is suffering from the attack and downloading invalid blocks.

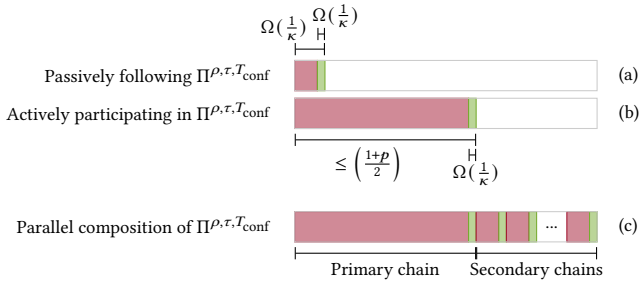

Figure 10: Worst-case throughput and bandwidth consumption, as a fraction of the total bandwidth. Green portions represent bandwidth consumption that contributes to throughput, while red portions represent bandwidth consumption that is caused by the adversary and may not contribute to throughput (*e.g.*, empty/invalid blocks, spamming).

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 9 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.

6 HIGH THROUGHPUT UNDER BANDWIDTH CONSTRAINT

From our analysis in Theorem 1, we parametrize $\Pi^{\rho,\tau,T_{\text{conf}}}$ 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 [19]. Thus, the throughput of $\Pi^{\rho,\tau,T_{\text{conf}}}$ decreases with increasing security parameter. Indeed, we show in Section 6.2 that the worst-case throughput of $\Pi^{\rho,\tau,T_{\text{conf}}}$ is lower bounded by $\frac{2p_{\text{U}}-p}{\tau} = \frac{1}{\Omega(\kappa)}$.

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 10(a)). In fact, the ratio between throughput and the bandwidth required to download the confirmed blocks is lower bounded by 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 κ. This suggests to invoke the idea of Parallel Chains [18, 19]: 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 κ increases.

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 participating in $\Pi^{\rho,\tau,T_{\text{conf}}}$ make the protocol secure, which is what allows other nodes to download the confirmed chain with little bandwidth consumption. However, even under spamming attacks, we show in Section 6.2 that the worst-case bandwidth consumption is only a little more than half the available bandwidth C (shown in Figure 10(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 secondary chains to occupy the remaining bandwidth (shown in Figure 10(c)). So, we modify the parallel chains construction from [18, 19] as described in the following section.

6.1 Parallel Chains Construction

The protocol consists of m parallel instances of $\Pi^{\rho,\tau,T_{\text{conf}}}$. 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 the freshest blocks 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_{conf}-deep LC confirmation rule based on the block headers (see Figure 11). Transactions from the confirmed portion of all the chains are first ordered by their time slots and then by the index

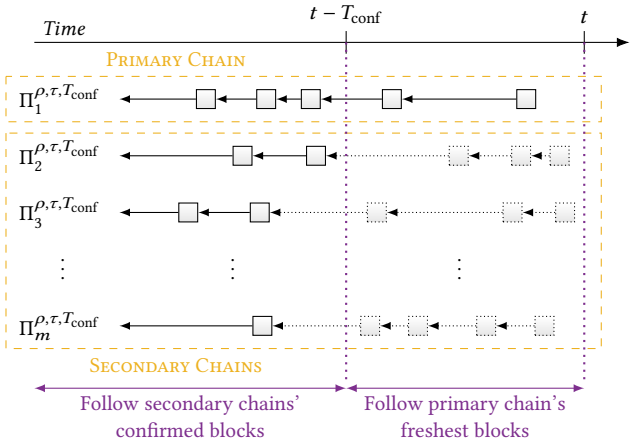

Figure 11: In the parallel chains construction using $\Pi^{\rho,\tau,T_{\text{conf}}}$, each node is assigned one primary chain; the other $(m-1)$ chains are secondary. Nodes participate actively in their primary chain using the freshest block download rule, and follow their secondary chains passively by downloading confirmed blocks only.

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, so as to avoid duplicating transactions across different $\Pi^{\rho,\tau,T_{\text{conf}}}$ instances. See Appendix D for pseudocode for the above parallel chains construction.

Each instance of $\Pi^{\rho,\tau,T_{\text{conf}}}$ is secure if at most β 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 like close to the overall adversarial power, rendering the construction secure against non-adaptive adversaries that corrupt at most β fraction of the nodes. See Appendix F for a more detailed security analysis.

6.2 Analysis of 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 unique slots, p_{U} blocks per slot (Proposition 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 5. *(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 $T \geq T_1$, $\mathsf{dC}_i(t_2) \setminus \mathsf{dC}_i(t_1)$ 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 5, the throughput of each chain is at least $\text{TP}_1 = \frac{\theta}{\tau}$ blocks per second.[5] 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 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 6. *(Passive Bandwidth Consumption) There exists a constant T_3 such that for any honest nodes i, i' and time slots $t_1, t_2 \geq t_1 + T$ such that $T \geq T_3$, $\mathsf{LOG}_{i'}^{t_2} \setminus \mathsf{LOG}_i^{t_1}$ contains at most $\phi_{\text{p}} T(1+\epsilon_6)$ blocks, with probability at least $1 - \exp(-\alpha_6 T)$, where $\phi_{\text{p}} = p$.*

Finally, we analyze the worst-case bandwidth consumption of active nodes in $\Pi^{\rho,\tau,T_{\text{conf}}}$. As per the freshest block download rule (see Algorithm 1, lines 17ff.), 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 20). Since in every unique slot, each node downloads the freshest block within one slot (Proposition 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 9 for a matching observation in our experiments.)

Lemma 7. *(Active Bandwidth Consumption) There exists a constant T_2 such that for any honest node i and time slots $t_1, t_2 \geq t_1 + T$ with $T \geq T_2$, node i does not download any blocks during at least $\phi_{\text{idle}} T(1-\epsilon_5)$ slots in the interval $(t_1, t_2]$, with probability at least $1 - \exp(-\alpha_5 T)$, where $\phi_{\text{idle}} = \frac{p_{\text{U}}(1-p)}{p} \geq \frac{1-p}{2}$.*

Note that this gives only a simple upper bound on the bandwidth consumption of active nodes. The freshest block rule may be further refined to reduce bandwidth consumption, *e.g.*, by not considering equivocating blocks as 'fresh'. Lemmas 5, 6 and 7 are proved in Appendix E.3.

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

$$\begin{aligned} \text{TP}_m = m\,\text{TP}_1 &= \left(1 + \frac{\phi_{\text{idle}}}{\phi_{\text{p}}} C\tau\right) \frac{\theta}{\tau} \\ &\geq \frac{(1-p)(2p_{\text{U}} - p)}{2p}\, C \\ &= \frac{(1-p)\epsilon_1}{2}\, C \text{ blocks per second} \end{aligned} \tag{19}$$

using $p_{\text{U}} = \frac{1}{2}p(1+\epsilon_1)$ from Theorem 1. The throughput of a single chain goes to zero as the security parameter increases (since τ is proportional to the security parameter), but this is compensated by increasing m, so that the aggregate throughput of the parallel chains remains within a constant fraction of the optimal throughput

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

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.

The worst-case throughput of a single chain and that of the parallel construction are limited by the chain quality factor $\frac{2p_U - p}{p}$ due to the possibility of selfish mining attacks [17]. Using the Conflux inclusion rule from [32] (which is also employed in [19]), this factor can be improved to $\frac{p_U}{p}$. In this rule, each block includes pointers to orphaned blocks that have not been pointed to. However, to use this rule in a bandwidth constrained network, one must 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. This is achieved by tuning parameters based on chain quality.

ACKNOWLEDGMENT

We thank Dan Boneh and Ertem Nusret Tas for fruitful discussions. JN is supported by the Protocol Labs PhD Fellowship 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] 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

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

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

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

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

[23] 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

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

[25] 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

[26] 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

[27] 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.

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

[29] 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.

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

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

[32] 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).

[33] 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

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

[35] 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*.

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

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

[38] 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

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

[40] 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.

[41] 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.

[42] 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
[43] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *2020 IEEE Symposium on Security and Privacy (SP)*. 90–105. https://doi.org/10.1109/SP40000.2020.00008

A HELPER FUNCTIONS FOR ALGORITHM 1 AND ALGORITHM 2

- Hash(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'
- prefixChainsOf(C): Set of prefixes of C
- longestChain($\mathcal{T}$): Determine longest chain among set $\mathcal{T}$ of chains. Ties are broken by the adversary.
- $C^{\lceil T_{\text{conf}}}$: Prefix of chain C consisting of all blocks with time slots up to T_{conf} less than the current time slot
- txsAreSemanticallyValidWrtPrefixesOf(C, 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)
- newBlock(time = t, party = P, txsHash = Hash(txs)): Produces a new block header with the given parameters
- $\mathcal{Z}$.broadcastHeaderChain(C): Broadcasts header chain C via $\mathcal{Z}$ to other nodes
- $\mathcal{Z}$.uploadContent(C, txs): Uploads content txs for the block identified by chain C into the block content repository of $\mathcal{Z}$ ($\mathcal{Z}$ only stores the content txs if its hash matches the transaction hash in C)
- $\mathcal{Z}$.requestContent(C): Requests content associated with block identified by chain C from the repository of $\mathcal{Z}$
- $\mathcal{Z}$.receivePendingTxsSemanticallyValidWrt(C): Retrieves a set of pending transactions that are not included in but semantically valid (see above) with respect to C
- $\mathcal{Z}$.outputLedger(C): Declares C as the node's ledger to the $\mathcal{Z}$ (this constitutes LOG_i^t for which consistency and liveness are required for a secure consensus protocol)

B ADDITIONAL HELPER FUNCTIONS FOR ALGORITHM 3 (SEE ALSO APPENDIX A)

- 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.
- $\mathcal{Z}$.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.

C SUPPLEMENTAL EXPERIMENTAL MATERIAL

C.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 mine blocks at the beginning of the same slot, and report the time for all 17 nodes to download all N blocks.

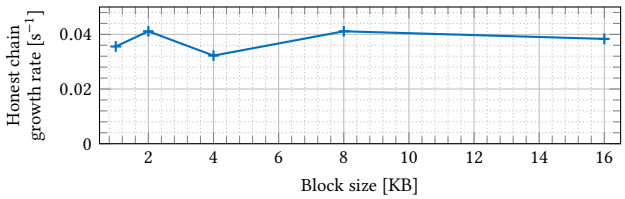

Figure 12: Honest chain growth rate under spamming attack when using different block sizes and the download freshest block rule. Despite the increasing network load (through the increasing block size), there is no performance deterioration when downloading the freshest block.

C.2 Chain Growth with Larger Blocks

In this experiment, we look at the robustness of our scheme when we increase the block size. The topology is the same as previous experiments, but the in-flight cap is fixed to 1. Figure 12 shows that the 'download towards the freshest block' rule maintains the chain growth rate, despite the increasing network load.

D PARALLEL CHAINS PSEUDOCODE

Algorithm 3 gives pseudocode for the parallel chains construction using our PoS LC protocol with the "download freshest block" 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 $\mathcal{F}_{\text{parallel}}^{\rho,m}$. 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 freshest block rule as in Algorithm 1. If there are no freshest blocks left to be downloaded, 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 20, 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 $\mathcal{F}_{\text{parallel}}^{\rho,m}$ (Algorithm 4) 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.

Algorithm 3 Parallel Chains PoS LC consensus protocol $\Pi_{\text{pc}}^{\rho,\tau,T_{\text{conf}},m}$ (helper functions: Appendix B, $\mathcal{F}_{\text{parallel}}^{\rho,m}$: Algorithm 4, $\Pi^{\rho,\tau,T_{\text{conf}}}$: Algorithm 1)

1: **on** INIT(genesisHeaderChain, genesisTxs)
2: pri ← $\mathcal{F}_{\text{parallel}}^{\rho,m}$.PRIMARYCHAININDEX()
3: **for** idx = 1, ..., m
4: Π_{idx} ← new $\Pi^{\rho,\tau,T_{\text{conf}}}$ ▷ *Initialize m instances of* $\Pi^{\rho,\tau,T_{\text{conf}}}$
5: Π_{idx}.INIT(genesisHeaderChain, genesisTxs)
6: **on** RECEIVEDHEADERCHAIN(idx, C)
7: Π_{idx}.RECEIVEDHEADERCHAIN(C)
8: **on** RECEIVEDCONTENT(idx, C, txs)
9: Π_{idx}.RECEIVEDCONTENT(C, txs)
10: **on** SCHEDULECONTENTDOWNLOAD() ▷ *Called when download idle*
11: Π_{pri}.SCHEDULECONTENTDOWNLOAD() ▷ *First priority for primary*
12: **if** no content requested by Π_{pri}
13: $\mathcal{S}$ ← {longestChain(Π_{idx}.h$\mathcal{T}$)$^{\lceil T_{\text{conf}}}$ | idx ∈ {1, ..., m}\{pri}}
14: C ← $\arg\min_{C'' \preceq C' \in \mathcal{S}, \text{blkTxs}[C'']=\text{unknown}} C''$.time ▷ *Pick earliest unknown block from any confirmed secondary chain*
15: $\mathcal{Z}$.REQUESTCONTENT(C)
16: **for** time slots t ← 1, ..., T_{h} of duration τ
17: txs ← $\mathcal{Z}$.RECEIVEPENDINGTXSSEMANTICALLYVALIDWRT(Π_{pri}.dC) ▷ *Only include valid txs whose accounts belong to the primary chain*
18: **if** $C' \neq \perp$ **with** C' ← $\mathcal{F}_{\text{parallel}}^{\rho,m}$.EXTEND(pri, t, Π_{pri}.dC, txs) ▷ *Check eligibility to produce a new block, and if so do so, see Algorithm 4*
19: $\mathcal{Z}$.UPLOADCONTENT(pri, C', txs)
20: $\mathcal{Z}$.BROADCASTHEADERCHAIN(pri, C')
21: tmax ← max{t: Π_{idx}.d$C^{\lceil T_{\text{conf}}}$.time ≥ t, idx ∈ {1, ..., m}} ▷ *Find the maximum time slot of all downloaded and confirmed chains*
22: LOG ← sortBySlotThenIndex({C | $C \preceq \Pi_{\text{idx}}$.d$C^{\lceil T_{\text{conf}}}$, C.time ≤ tmax, idx ∈ {1, ..., m}}) ▷ *Arrange confirmed and downloaded chains in increasing order of time slots, then chain index*
23: $\mathcal{Z}$.OUTPUTLEDGER(LOG)

Algorithm 4 Idealized functionality $\mathcal{F}_{\text{parallel}}^{\rho,m}$ for parallel chains (see also $\mathcal{F}_{\text{headertree}}^{\rho}$: Algorithm 2)

1: **on** INIT(genesisHeaderChain, numParties)
2: $\mathcal{P}_1, ..., \mathcal{P}_m$ ← random equi-partition of {1, ..., numParties}
3: **for** idx = 1, ..., m
4: **for** $P \in \mathcal{P}_{\text{idx}}$
5: pri[P] ← idx
6: $\mathcal{F}_{\text{idx}}$ ← new $\mathcal{F}_{\text{headertree}}^{\rho}$ ▷ *Initialize m instances of* $\mathcal{F}_{\text{headertree}}^{\rho}$
7: $\mathcal{F}_{\text{idx}}$.INIT(genesisHeaderChain, numParties/m)
8: **on** PRIMARYCHAININDEX() **from** party P
9: **return** pri[P]
10: **on** EXTEND(idx, t', C, txs) **from** party P **at** time slot t
11: **if** pri[P] ≠ idx
12: **return** $\perp$
13: **return** $\mathcal{F}_{\text{idx}}$.EXTEND(t', C, txs)

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 [21, 28, 30] 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 of block production (Algorithm 3 line 16), 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.

E PROOF DETAILS

E.1 Combinatorial Analysis

E.1.1 Proof of Lemma 1.

Proof. 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 unique 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 $\text{ShortPrefixes}_{\bar{C},\gamma}$ and Lemma 4, this also means that there are no unique pivot slots in the interval $(t-\gamma, t]$, which is a contradiction to $\text{FrequentPivots}_{\gamma}$.

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, tx $\in \text{LOG}_i^{t'}$ for all $t' \geq t^* + T_{\text{conf}}$, which is at most $t + 2\gamma$. □

E.2 Probabilistic Analysis

E.2.1 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

$$\forall (r,s] \ni t, s-r < w\colon\ \text{PivotCondition}_{(r,s]} \vee (\mathcal{A}(r,s] = 0)\,. \quad (20)$$

Proposition 3. *If $p_{\text{U}} = \frac{1}{2}p(1+\epsilon_1)$ for some $\epsilon_1 \in (0,1)$,*

$$\forall (r,s]\colon\ \Pr\left[\neg\text{PivotCondition}_{(r,s]}\right] \leq 2\exp\left(-\alpha_1 p(s-r)\right), \quad (21)$$

with $\alpha_1 = \eta\epsilon_1^2$ and $\eta = 1/36$.

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

$$\Pr\left[\mathcal{B}(r,s] \geq p(s-r)(1+\epsilon)\right] \leq \exp\left(-\frac{\epsilon^2 p(s-r)}{2+\epsilon}\right). \quad (22)$$

Also, by a Chernoff bound for $\epsilon > 0$,

$$\Pr\left[\mathcal{U}(r,s] \le p_{\mathrm{U}}(s-r)(1-\epsilon)\right] \le \exp\left(-\frac{\epsilon^2 p_{\mathrm{U}}(s-r)}{2}\right) \quad (23)$$

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

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

except with probability

$$\exp\left(-\frac{\epsilon^2 p(s-r)}{2+\epsilon}\right) + \exp\left(-\frac{\epsilon^2 p_{\mathrm{U}}(s-r)}{2}\right) \quad (24)$$

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

$$2\exp\left(\frac{\epsilon_1^2 p(s-r)}{36}\right) \quad (25)$$

□

E.2.2 Proof of Lemma 3.

Proof. Define the event F_t as

$$\max_{t'<t\colon \mathsf{Unique}(t')\wedge(\mathcal{A}(t',t]\ge\mathcal{U}(t',t])} \mathcal{A}\big(t',t\big] \ge \bar{C}. \quad (26)$$

This event can be equivalently expressed as

$$\exists t' < t\colon \mathsf{Unique}(t') \wedge \left(\mathcal{A}\big(t',t\big] \ge \mathcal{U}\big(t',t\big]\right) \wedge \left(\mathcal{A}\big(t',t\big] \ge \bar{C}\right). \quad (27)$$

The event $\{\neg\mathsf{ShortPrefixes}_{\bar{C},\gamma}\}$ can be expressed as $\bigcup_{t\le T_{\mathrm{h}}} F_t$. Then for some fixed T,

$$\begin{aligned}\Pr[F_t] &\le \Pr\left[\bigcup_{t'=0}^{t-1}\left\{\mathcal{A}\big(t',t\big] \ge \mathcal{U}\big(t',t\big] \wedge \mathcal{A}\big(t',t\big] \ge \bar{C}\right\}\right]\\ &\le \sum_{t'=0}^{t-T}\Pr\left[\mathcal{A}\big(t',t\big] \ge \mathcal{U}\big(t',t\big]\right] + \sum_{t'=t-T}^{t-1}\Pr\left[\mathcal{A}\big(t',t\big] \ge \bar{C}\right]\\ &\le \sum_{k=T}^{\infty} 2\exp\left(-\alpha_1 pk\right) + T\exp\left(-\frac{\epsilon_2^2}{2+\epsilon_2}p_A T\right)\\ &= \frac{2\exp(-\alpha_1 pT)}{1-\exp(-\alpha_1 p)} + T\exp\left(-\frac{\epsilon_2^2}{2+\epsilon_2}p_A T\right)\\ &\le 2T\exp\left(-\alpha_2 pT\right), \end{aligned} \quad (28)$$

for $T \ge \frac{2}{1-\exp(-\alpha_1 p)}$ 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_{h}, we get

$$\Pr\left[\neg\mathsf{ShortPrefixes}_{\bar{C},\gamma}\right] \le 2T_{\mathrm{h}}T\exp(-\alpha_2 pT) \le 2T_{\mathrm{h}}^2\exp(-\alpha_2 pT) \quad (29)$$

Note that we also require $T > \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_2 p}$ so that the probability bound is smaller than 1. □

E.2.3 Proof of Lemma 2.

Proposition 4. *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for an execution horizon T_{h} and $w > \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1 p}$,*

$$\Pr\left[\exists (r,s], s-r \ge w\colon \neg\mathsf{PivotCondition}_{(r,s]}\right] \le 2T_{\mathrm{h}}^2\exp\left(-\alpha_1 pw\right). \quad (30)$$

Proof. Using a union bound and Proposition 3,

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

□

Proposition 5. *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for a time horizon T_{h} and $w > \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1 p}$,*

$$\Pr\left[\exists t\colon \mathsf{WeakPivot}_w(t) \wedge \neg\mathsf{Pivot}(t)\right] \le 2T_{\mathrm{h}}^2\exp(-\alpha_1 pw). \quad (31)$$

Proof. If some t is a weak pivot (with $w \ge \frac{2\ln(\sqrt{2}T_{\mathrm{h}})}{\alpha_1 p}$) and t is not a pivot, then $\exists (r,s] \ni t$ with $s-r \ge w$ such that $\neg\mathsf{PivotCondition}_{(r,s]}$. But the probability for this is bounded accordingly by Proposition 4. □

Proposition 6. *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for time horizon T_{h},*

$$\forall t\colon \quad \Pr\left[\mathsf{WeakPivot}_w(t) \mid \mathsf{Unique}(t)\right] \ge p_1 \quad (32)$$

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

$$E_1 \triangleq \{\mathcal{A}(t-v, t+v] = 0\}, \quad (33)$$

$$E_2 \triangleq \{\forall (r,s] \ni t, s-r < w, (r,s] \notin (t-v,t+v]\colon \mathsf{PivotCondition}_{(r,s]}\}. \quad (34)$$

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

For bounding $\Pr[\neg E_2]$, 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 $u \le v$, note that $(r,s] \ni t$ implies that $(r,s] \in (t-v,t+v]$. One can check that for $v+1 \le u \le 2v$, there are $2(u-v)-1$ intervals $(r,s] \ni t$ such that $(r,s] \notin (t-v,t+v]$. For $2v+1 \le u < w$, 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 3 and a union bound,

$$\begin{aligned}\Pr[\neg E_2] &\le \sum_{u=v+1}^{w-1}\sum_{\substack{(r,s]\ni t\colon\\ s-r=u\wedge\\ (r,s]\notin(t-v,t+v]}}\Pr[\neg\mathsf{PivotCondition}(r,s]]\\ &\le \sum_{u=v+1}^{2v}(2(u-v)-1)2e^{-\alpha_1 pu} + \sum_{u=2v+1}^{w-1}u2e^{-\alpha_1 pu}\\ &\le \sum_{k=1}^{v}2(2j-1)e^{-\alpha_1 p(v+j)} + \sum_{u=2v+1}^{w-1}2ue^{-\alpha_1 pu}\end{aligned}$$

$$
\begin{aligned}
&\le \sum_{k=1}^{v} 2(2j-1)e^{-\alpha_1 p(v+j)} + \sum_{u=2v+1}^{\infty} 2ue^{-\alpha_1 pu} \\
&= \frac{2e^{-\alpha_1 p(v+1)}(1-(2v+1)e^{-\alpha_1 pv})}{1-e^{-\alpha_1 p}} \\
&\quad + \frac{4e^{-\alpha_1 p(v+2)}(1-e^{-\alpha_1 pv})}{(1-e^{-\alpha_1 p})^2} \\
&\quad + \frac{2(2v+1)e^{-\alpha_1 p(2v+1)}}{1-e^{-\alpha_1 p}} + \frac{2e^{-\alpha_1 p(2v+2)}}{(1-e^{-\alpha_1 p})^2} \\
&= \frac{2e^{-\alpha_1 p(v+1)}}{1-e^{-\alpha_1 p}} + \frac{4e^{-\alpha_1 p(v+2)} - 2e^{-\alpha_1 p(2v+2)}}{(1-e^{-\alpha_1 p})^2} \\
&\le \frac{2e^{-\alpha_1 p(v+1)}}{1-e^{-\alpha_1 p}} \left(1 + \frac{2e^{-\alpha_1 p}}{1-e^{-\alpha_1 p}}\right) \\
&\le \frac{2e^{-\alpha_1 pv}(1+e^{-\alpha_1 p})}{(1-e^{-\alpha_1 p})^2} \qquad (35)
\end{aligned}
$$

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 $\Pr[\neg E_2] \le \frac{1}{2}$.

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

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

for the given choice of v. □

Proposition 7. *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for horizon T_{h} and $w > \frac{2}{\alpha_1 p} \ln\left(\frac{4(1+e^{-\alpha_1 p})}{(1-e^{-\alpha_1 p})^2}\right)$,*

$$
\begin{aligned}
&\forall t\colon \Pr\left[\exists t' \in (t, t+\gamma]\colon \mathsf{WeakPivot}_w(t') \wedge \mathsf{Unique}(t')\right] \\
&\quad \ge 1 - \exp(-\alpha_3 \gamma / w), \qquad (36)
\end{aligned}
$$

with $\alpha_3 = \frac{p_1 p_{\mathrm{U}}}{2}$.

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

$$
\begin{aligned}
E_i &\triangleq \{\mathsf{WeakPivot}_w(t_i) \wedge \mathsf{Unique}(t_i)\} \qquad (37) \\
E &\triangleq \{\exists t' \in (t, t+\gamma]\colon \mathsf{WeakPivot}_w(t') \wedge \mathsf{Unique}(t')\}. \qquad (38)
\end{aligned}
$$

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

$$
\begin{aligned}
\Pr[E] &\ge \Pr\left[\bigcup_{i=0}^{k-1} E_i\right] = 1 - \Pr\left[\bigcap_{i=0}^{k-1} \neg E_i\right] \\
&\ge 1 - (1 - p_1 p_{\mathrm{U}})^k \\
&\ge 1 - \exp(-p_1 p_{\mathrm{U}} k) \\
&= 1 - \exp(-p_1 p_{\mathrm{U}} \gamma / 2w), \qquad (39)
\end{aligned}
$$

where we have used Proposition 6. □

Proposition 8. *If $p_{\mathrm{U}} = \frac{1}{2}p(1+\epsilon_1)$, then for horizon T_{h}, $w > \frac{2}{\alpha_1 p} \ln\left(\frac{4(1+e^{-\alpha_1 p})}{(1-e^{-\alpha_1 p})^2}\right)$ and $\gamma > \frac{w \ln(T_{\mathrm{h}})}{\alpha_3}$,*

$$
\begin{aligned}
&\Pr\left[\forall t\colon \exists t' \in (t, t+\gamma]\colon \mathsf{WeakPivot}_w(t') \wedge \mathsf{Unique}(t')\right] \\
&\quad \ge 1 - T_{\mathrm{h}} \exp(-\alpha_3 \gamma / w). \qquad (40)
\end{aligned}
$$

Proof. By a union bound over all T_{h} possible time slots, and using Proposition 7. □

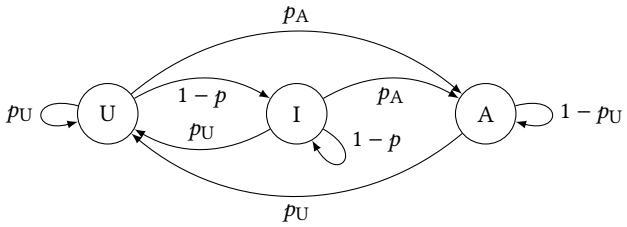

Figure 13: An upper bound on the bandwidth utilization of our protocol can be calculated from the stationary distribution of this Markov chain

Proof of Lemma 2. Finally, to prove Lemma 2, let

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

Note that $E_1 \cap E_2 \subseteq E$. Then we apply a union bound on the probabilities from Propositions 8 and 5. □

E.3 Proofs for Throughput and Bandwidth Consumption

E.3.1 Proof of Lemma 5.

Proof. Due to Proposition 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

$$
\begin{aligned}
&\Pr[\mathcal{U}(t_1, t_2] \ge p_{\mathrm{U}} T(1-\epsilon)] \\
&\ge \Pr[\mathcal{U}(t_1, t_2] \ge p_{\mathrm{U}}(t_2 - t_1)(1-\epsilon)] \ge 1 - \exp\left(\frac{\epsilon^2}{2} p_{\mathrm{U}} T\right). \qquad (41)
\end{aligned}
$$

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 α_4. Finally, we note that $\theta = p_{\mathrm{U}} - p_{\mathrm{A}} = 2p_{\mathrm{U}} - p$. □

E.3.2 Proof of Lemma 7.

Proof. Consider the Markov chain shown in Figure 13 with three states—U corresponding to a unique slot, I corresponding to a slot without a block proposal such that the most recent block proposal was a unique 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

$$\pi_{\mathrm{U}} = p_{\mathrm{U}}, \quad \pi_{\mathrm{I}} = \frac{p_{\mathrm{U}}(1-p)}{p}, \quad \pi_{\mathrm{A}} = \frac{p_{\mathrm{A}}}{p}. \tag{42}$$

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 ϕ_{idle} fraction of time slots, every honest node's bandwidth remains idle, where

$$\begin{aligned}\phi_{\mathrm{idle}} \geq \pi_{\mathrm{I}} &= \frac{p_{\mathrm{U}}(1-p)}{p} \\ &= \frac{1}{2}(1-p)(1+\epsilon_1) \\ &\geq \left(\frac{1-p}{2}\right). \end{aligned} \tag{43}$$

(For ϵ_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 $\phi_{\mathrm{idle}}T(1-\epsilon_5)$ slots in the I state in the interval $(t_1, t_2]$ is at least $1-\exp\left(-\frac{\epsilon_5^2}{2}\phi_{\mathrm{idle}}T\right)$. □

E.3.3 Proof of Lemma 6.

Proof. Consider time slots t_1 and $t_2 \geq t_1 + T$. 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 $t_1' \geq t_1 - 2T_{\mathrm{conf}}$ 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 $p(t_2' - t_1') \leq p(t_2 - t_1 + T_{\mathrm{conf}}) = p(T + T_{\mathrm{conf}})$. Then by a Chernoff bound,

$$\Pr\left[\mathcal{B}(t_1', t_2'] > pT(1+\epsilon_6)\right] \leq \exp\left(-\alpha_6 T\right) \tag{44}$$

for sufficiently large $T > T_{\mathrm{conf}}$ and some constant α_6. □

F SECURITY OF PARALLEL CHAINS

Theorem 2. *Except with probability* $\mathrm{negl}(\kappa)$ *over executions* $\mathcal{E}^{\beta,T_{\mathrm{h}}}$, *the protocol* $\Pi_{\mathrm{pc}}^{\rho,\tau,T_{\mathrm{conf}},m}$ *with parameters* ρ *such that* $(1-\beta)\rho e^{-\rho} = \frac{1-e^{-\rho}}{2}(1+\epsilon_1)$, $\tau = \Omega(\kappa + \ln T_{\mathrm{h}})$, $T_{\mathrm{conf}} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$ *and* $m = 1 + \frac{p_{\mathrm{U}}(1-p)}{p^2}C\tau(1-\epsilon_7)$ *achieves safety and liveness with parameter* $T_{\mathrm{live}} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$.

Proof. Consider a particular protocol instance Π_{idx}. Define $\mathrm{d}C_{i,\mathsf{idx}}$ to be the longest downloaded chain of node i for protocol instance Π_{idx}. From Theorem 1, for the given ρ, τ and $T_{\mathrm{conf}} = \gamma$, each protocol instance Π_{idx} satisfies safety and liveness with respect to the ledger defined by $\mathrm{d}C_{i,\mathsf{idx}}(t)^{\lceil T_{\mathrm{conf}}}$ and for nodes i for which Π_{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 Π_{idx}, $\mathrm{d}C_{i,\mathsf{idx}}(t)^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{j,\mathsf{idx}}(t')^{\lceil T_{\mathrm{conf}}}$ or $\mathrm{d}C_{j,\mathsf{idx}}(t')^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{i,\mathsf{idx}}(t)^{\lceil T_{\mathrm{conf}}}$ for all time slots t, t' and all honest nodes i, j for which Π_{idx} is the primary chain. However, this holds even if Π_{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 Π_{idx} is a secondary chain, can also do so with headers and contents for a node j' for which Π_{idx} is the primary chain, thus causing a safety violation, which contradicts the earlier observation. Since all nodes have consistent confirmed chains (*i.e.* $\mathrm{d}C_{i,\mathsf{idx}}(t)^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{j,\mathsf{idx}}(t')^{\lceil T_{\mathrm{conf}}}$ or $\mathrm{d}C_{j,\mathsf{idx}}(t')^{\lceil T_{\mathrm{conf}}} \preceq \mathrm{d}C_{i,\mathsf{idx}}(t)^{\lceil T_{\mathrm{conf}}}$) 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 7, 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 6, 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_{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 $m - 1 = \frac{\phi_{\mathrm{idle}}\tilde{T}(1-\epsilon_5)}{\phi_{\mathrm{p}}\tilde{T}(1+\epsilon_6)}C\tau = \frac{p_{\mathrm{U}}(1-p)}{p^2}C\tau(1-\epsilon_7)$ for some ϵ_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 D, 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 1 in Appendix E.1) 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 $\gamma + T_{\mathrm{conf}} + \tilde{T} = \Omega((\kappa + \ln T_{\mathrm{h}})^2)$. □