Analysis of Deterministic Longest-Chain Protocols

Elaine Shi, Cornell University

Abstract

Most classical consensus protocols rely on a leader to coordinate nodes' voting efforts. One novel idea that stems from blockchain-style consensus is to rely, instead, on a "longest-chain" idea for such coordination. Such a longest-chain idea was initially considered in *randomized* protocols, where in each round, a node has some probability of being elected a leader who can propose the next block. Recently, well-known systems have started implementing the *deterministic* counterpart of such longest-chain protocols — the deterministic counterpart is especially attractive since it is even simpler to implement than their randomized cousins. A notable instantiation is the Aura protocol which is shipped with Parity's open-source Ethereum implementation.

Interestingly, mathematical analyses of deterministic, longest-chain protocols are lacking even though there exist several analyses of randomized versions. In this paper, we provide the first formal analysis of deterministic, longest-chain-style consensus. We show that a variant of the Aura protocol can defend against a Byzantine adversary that controls fewer than $\frac{1}{3}$ fraction of the nodes, and this resilience parameter is tight by some technical interpretation. Based on insights gained through our mathematical treatment, we point out that Aura's concrete instantiation actually fails to achieve the resilience level they claim.

Finally, while our tight proof for the longest-chain protocol is rather involved and non-trivial; we show that a variant of the "longest-chain" idea which we call "largest-set" enables a textbook construction that admits a simple proof (albeit with slower confirmation).

1 Introduction

A central abstraction in distributed systems is called *consensus* or *state machine replication* [13,29,42]. In state machine replication, a distributed system of nodes reach agreement on an ever-growing, linearly ordered log. State machine replication protocols have long been deployed in companies such as Google and Facebook to replicate their computing infrastructure; more recently they are deployed on an Internet scale enabling cryptocurrency systems such as Bitcoin and Ethereum. State machine replication protocols have been studied for three decades [13,29,32] and two major approaches exist:

1. classical-style consensus (e.g., PBFT [13], Paxos [29], and others [9,11,15–17,28,30,43]) and
2. longest-chain-style consensus (e.g., Bitcoin and Ethereum's proof-of-work blockchains and proof-of-stake blockchain constructions [18,19,27,40]).

In both of these approaches, nodes vote to reach a decision. From a technical perspective, a primary distinction between classical-style and longest-chain-style consensus lies in how the voting process is coordinated. In classical consensus, the voting process is typically coordinated by a leader (possibly rotating over time); and whenever the leader is honest and proposes the same item (e.g., block or batch of transactions) to everyone, everyone will vote on the same item and a decision can be made very soon by collecting majority or super-majority votes (possibly over multiple rounds of voting). In longest-chain-style protocols, *the leader election and voting processes are coalesced* *into one*: whenever one is elected leader it has the opportunity to cast a vote. To coordinate nodes voting effort, a new "longest-chain" idea is adopted — in some sense, people vote on the most-popular history observed so far. Such coordination is important for consistency: intuitively, if nodes fail to concentrate their voting efforts, it can lead to many forks and no chain will emerge as the dominant decision.

Obviously, the longest-chain idea first appeared in the celebrated Nakamoto consensus [36], i.e., the consensus protocol underlying Bitcoin. Since the original Nakamoto consensus employs a random leader election mechanism, *all* existing mathematical analyses of longest-chain-style consensus focus on the *randomized* setting [19, 22, 27, 37, 39, 40]. Note that although Nakamoto must assume proof-of-work, later schemes have shown how to design "permissioned" consensus protocols that emulate Nakamoto's stochastic process but dispense with proof-of-work entirely [18, 19, 27, 40]. A very natural simplification of longest-chain-style protocols is to replace the randomized leader election with a *deterministic, round-robin* policy. Perhaps due to the simplicity, some practical systems [4, 5] have started adopting this approach. Somewhat surprisingly, although analyses for randomized, longest-chain-style protocols are well-known by now, existing randomized proofs do not imply security of the deterministic variant (even though the latter seems simpler). Since deterministic longest-chain-style protocols are now being implemented and widely distributed in practice, it is imperative that we establish a correct mathematical understanding of its security properties.

A representative deterministic, longest-chain-style protocol is Aura [4] (short for Authority Round). Aura is well-known in the cryptocurrency community since it is one of the few blockchain consensus algorithms shipped with Parity's open-source Ethereum implementation [1] — this is one of the few major Ethereum implementations; and at the time of writing, the `github` project has had 11,978 commits and 192 contributors.

Roughly speaking, the Aura blockchain protocol works as follows:

- There are n nodes numbered $0, 1, \ldots, n-1$, each with a known public key. In round r, node $(r \mod n)$ is the leader.
- In every round r , the leader chooses the longest chain seen so far, and extends the chain by signing the next block containing 1) the current round number r, henceforth also called the block's *timestamp*, 2) a batch of transactions, and 3) the hash of the parent block.
- In a valid blockchain, the blocks' timestamps must strictly increase, and moreover honest nodes reject blocks with timestamps in the future.
- At any time, a node takes its longest chain and chops off some number of blocks at the end, and the prefix is considered finalized.

Specifically, Aura makes the following parameter choices and claims: for finalization, they choose to chop off $\lfloor \frac{n}{2} \rfloor + 1$ trailing blocks signed by distinct signers. Their documentation claims that the protocol defends against *minority* Byzantine corruptions (which we shall refute in this paper).

1.1 Our Results and Contributions

Formal analyses of deterministic blockchains. To the best of our knowledge we are the first to provide mathematical proofs for deterministic, longest-chain-style protocols. More concretely, we make the following contributions:

- We present a variant of Aura's protocol that is provably secure. In this variant, in round r, all blocks in the longest chain whose timestamps are at most $r - n$ are considered finalized (*c.f.* Aura chops off only $\lfloor \frac{n}{2} \rfloor + 1$ blocks from distinct signers).

- We prove that this variant ensures consistency and liveness in the presence of any adversary that can control fewer than $\frac{1}{3}$ fraction of the nodes; further, transaction confirmation takes $\Theta(n)$ rounds since we need to chop off $\Theta(n)$ trailing blocks for finalization. Note that the round complexity of this protocol is asymptotically optimal due to the well-known lower bound [10, 21, 23, 31, 33, 35] that any *deterministic* consensus protocol must incur at least $f + 1$ rounds where f is the number of corrupt nodes (and this lower bound holds even for crash fault).

- Unlike existing proofs for randomized blockchains that require stochastic analyses that reason about the good properties respected by an overwhelming fraction of execution traces (as opposed to the worst-case execution trace), our proof relies on elementary discrete math — nonetheless as we show, proving an asymptotically tight bound on transaction confirmation time is *non-trivial.* The most sophisticated part of the proof involves a combinatorics argument that a particular good event called a "pivot" appears every n rounds even against the worst-case attack.

- We show that the $\frac{1}{3}$ resilience parameter is tight for this protocol; specifically, we show that if the adversary can control how honest nodes break ties among multiple chains of the same length, then there is an explicit $\frac{1}{3}$ attack against this protocol (note that our security proof admits arbitrary tie-breaking).

We point out that it is an interesting observation that the most natural embodiment of the "deterministic, longest-chain" idea defends only against less than $\frac{1}{3}$ corruptions while in comparison, the randomized counterparts can secure against upto minority corruption [22, 27, 37, 40]. Thus for future work, a technically intriguing question is whether there exist other natural embodiments of the "deterministic, longest-chain" idea that achieves stronger resilience.

Analysis of Aura's instantiation. Based on our mathematical analysis, we reflect on Aura's specific choices and claims for their deterministic blockchain instantiation.

As mentioned, our deterministic, longest-chain protocol described above defends against any $< \frac{1}{3}$ attack, and this is tight when tie-breaking can be arbitrary (for scoring two chains of the same length). Aura's instantiation employs a specific tie-breaking rule: when two chains are of the same length, they prefer the "earlier" one, i.e., whose last block has an earlier timestamp. We show that even under their tie-breaking rule, there is a $\frac{3}{7}$ attack against consistency no matter how many trailing blocks honest nodes chop off for finalization. Further, if we actually chop off only $\lfloor \frac{n}{2} \rfloor + 1$ trailing blocks with distinct signers for finalization (like what Aura does and assuming their tie-breaking rule), there is still an explicit attack that breaks consistency when the adversary controls slightly more than $\frac{3}{8}$ fraction of nodes. We conclude that Aura's claim of defending against minority corruptions is incorrect under the classical notion of a Byzantine adversary. Our analysis also implies that their concrete instantiation actually achieves a resilience parameter that is between $\frac{1}{6}$ and $\frac{3}{8}$.

Disclosure and recommendations to Parity. We have emailed Parity regarding our findings, and have suggested that they chop off more blocks from the end for higher resilience, and weaken their public claim that they defend against upto minority Byzantine corruptions. Parity responded and thanked us for our analysis, but at the time of the publication, they have not updated their corrupt-minority claim [4] or their code [1] based on our suggestions.

A largest-set variant for pedagogical purposes. Last but not the least, inspired by the longest-chain idea, we construct a new consensus protocol that relies on a similar "largest-set" idea, i.e., nodes always vote on the item that has collected the most number of votes so far. This protocol is of particular interest for pedagogy (e.g., to illustrate why and how a longest-chain-type idea works), since it admits a simple and elementary proof; moreover the proof is in some sense a deterministic counterpart of the recent analyses of randomized blockchains [22, 27, 37, 40]. Our largest-set variant and its proof have been adopted for teaching in a major blockchain winter school in December 2017 [7].

Non-goals. We stress that how to achieve synchronous consensus tolerating even $n-1$ faults has been long known, assuming a public-key infrastructure and digital signatures [20]. It is *not* a goal of this paper to construct schemes that improve existing results in a *theoretical or asymptotical* sense. Instead, our work is primarily motivated by two nonetheless important factors: 1) understand from a mathematical perspective exactly what is novel about the "longest-chain" idea, and identifying a simplest-possible scheme and proof to facilitate pedagogy of the "longest-chain" idea; and 2) mathematically analyze protocols already deployed and shipped in practice, both for reassurance and for clarifying existing misconceptions. Having said this, we do point out one interesting fact about longest-chain-style protocols: in a long-running setting and considering an *amortized* notion of bandwidth: *confirming each block requires only one block-multicast from a single node.* In comparison, classical-style consensus require all nodes to vote to confirm each block and some works [14, 17, 24] have suggested random sub-sampling a smaller committee to vote on each block to reduce bandwidth.

Throughout this paper, we focus on the permissioned consensus setting where nodes' public keys are known a-priori. Recent works [17, 18] have shown how to add "robust committee reconfiguration" on top of a *permissioned* consensus protocol to build open-enrollment, proof-of-stake consensus. While these techniques are generic and can be applied to the protocols described in our paper, it is outside the scope of our paper to provide a detailed recount of these approaches.

1.2 Additional Related Work

Earlier this section, we have reviewed classical consensus [9, 11, 13, 15–17, 28–30, 34, 43] and longest-chain-style consensus [18, 19, 22, 27, 36, 37, 39, 40], and explained intuitively why the "longest-chain" idea appears to be a new innovation in the latter approach. Note that among the aforementioned classical-style consensus protocols, some achieve agreement in a *partially synchronous* network [11, 13, 15, 29, 30, 43] while others achieve security in a *synchronous* network [9, 16, 17, 34]. To the best of our knowledge, known longest-chain-style protocols [18, 19, 27, 36, 37, 39, 40] are secure in a *synchronous* model. We now review additional related works.

EOS is among the top five cryptocurrencies by market cap at the time of the writing [2]; their consensus protocol, called DPoS [6] is also a deterministic, longest-chain-style protocol. EOS's DPoS design has undergone several iterations and is still evolving — see the excellent online discussions regarding the history of DPoS consensus [6]. An earlier version, DPoS 2.0 appears to be a close relative of the Aura protocol but DPoS 2.0 chops off $\lfloor 2n/3 \rfloor + 1$ number of trailing blocks signed by distinct nodes for finalization. In their latest version DPoS 3.0, they seem to be using subtly modified version of the "longest-chain" rule: rather than picking the longest chain (measured by total number of blocks) to extend, they pick a chain whose "last irreversible block" is the largest, and break ties using the number of blocks — note that this important choice, which likely will matter to the protocol's consistency and liveness properties, seems undocumented and we only discovered it from inspecting their open-source implementation [3]. DPoS 3.0 also tries to incorporate ideas from classical Byzantine Fault Tolerance (BFT) [13, 29] and Casper-FFG [12] by adopting a new, more conservative finalization rule. To date there is no known proof of security for DPoS.

In this paper, we focus on analyzing Aura's instantiation due to its simpler and clearer specification [4] and more stabilized implementation [1]. It would be of value to do a similar analysis for DPoS's variant (once the protocol's design has stabilized) as future work — or alternatively, it might be worthwhile to simplify and improve their protocol.

Concurrent and independent work. A concurrent and independent work called Ouroboros-BFT [26], Kiayias and Russell describe a variant of the deterministic, longest-chain-style protocol similar to Aura [4], they also provide an independent proof that $\Theta(n)$ time is necessary for confirmation. While both papers have essentially the same basic result (but with quite different proofs), each work focused on different extensions to the basic result: Ouroboros-BFT suggested an elegant extension that allows optimistically fast confirmation, and an extension to the proof showing that although consistency can be lost during network partitions, consistency and liveness will restore when the partition heals. Our work focuses on refuting the corrupt-minority claim made by Aura and clarifying misconceptions; thus we demonstrate explicit attacks against the protocol (including against Aura's specific instantiation) when the adversary controls significantly fewer than minority number of nodes. Furthermore, we propose a textbook variant suitable for pedagogy — this variant is designed to mathematically illustrates why *longest-chain-style* protocols work a simplest-possible protocol and proof, demonstrating why the "longest-chain" idea behind Nakamoto is novel.

1.3 Paper Organization

We first present a formal model for protocol execution and formally define Byzantine Agreement in Section 2. Then, in Section 3, we present a *deterministic*, longest-chain protocol that is a natural analog of known *randomized* longest-chain protocols [27, 27, 37, 39, 40]. We then formally show that this protocol realizes Byzantine Agreement as long as the adversary controls fewer than 1/3 fraction of nodes; furthermore, the 1/3 resilience parameter is tight for this protocol — for ease of understanding we first present a simpler proof with looser liveness parameters in Section 3 and defer it to the end of the paper to show a tighter (but more complicated) version of the proof.

At this moment, we are ready to investigate Parity's Aura protocol (Section 4), which is a variant of the protocol described in Section 3. We refute Parity's claim that Aura tolerates up to minority Byzantine corruptions. For ease of understanding, our paper's formalism begins with Byzantine Agreement, i.e., a single-shot consensus abstraction. In Section 5, we show that longest-chain style protocols naturally allow us to reach agreement not just once, but repeatedly over time — thus realizing a "state machine replication" abstraction. Finally, in Section 6, we give a textbook largest-set analog which admits a very simple proof.

2 Preliminaries

2.1 Model

We consider a standard, *synchronous* model of execution, where an adversary $\mathcal{A}$ is allowed to *adaptively* corrupt nodes during the protocol execution. In this section, we describe the model in more detail and introduce relevant notations.

Communication model. We assume a synchronous communication model, that is, if an honest node sends a message m in round t, all honest recipients must have received the message m by the beginning of round $t + \Delta$; further, subject to this constraint, the adversary may delay or reorder messages arbitrarily. We refer to Δ as the maximum network delay, and Δ is known to a synchronous consensus protocol (i.e., hard-wired in the protocol description).

Without loss of generality, throughout the paper we shall simply assume that $\Delta = 1$ — if $\Delta > 1$, we can always rename Δ rounds as one super-round, and only process the next batch of incoming messages at the beginning of each super-round.

The protocols we describe adopt a *multicast* communication pattern — whenever an honest node multicasts a message, it is destined for all nodes including itself. Corrupt nodes, on the other hand, may send messages to only a subset of the nodes.

We assume a *rushing* adversary, i.e., the adversary $\mathcal{A}$ can observe messages sent by honest nodes in a round before deciding what messages corrupt nodes will send in the same round.

Corruption model. We assume that $\mathcal{A}$ may adaptively corrupt nodes during protocol execution. Specifically, during each round r, $\mathcal{A}$ can examine what each honest node wants to send, it can then adaptively decide to corrupt a subset of the honest nodes in round r. If a node i becomes corrupt in round r, the messages that i originally wanted to send in this round can be erased; further, in the same round r, the now-corrupt i can send arbitrary corrupt messages (that may depend on the messages that honest nodes send in round r).

Given a protocol that has a fixed termination time T_{end} (which may be a function dependent on total number of nodes), if a node becomes corrupt in or before round T_{end}, we say that the node is *eventually-corrupt*; else we say that the node is *honest-forever*. The protocols described in this paper will have provable security when $n \geq 3f + 1$ where f is the number of eventually-corrupt nodes, and n denotes the total number of nodes.

Notations. We assume that $\mathcal{A}$ is responsible for spawning the nodes that participate in the protocol and we typically use $0, 1, \ldots n-1$ to index the nodes. At the start of the protocol execution, $\mathcal{A}$ is responsible for choosing the inputs of (honest) nodes. We say that $\mathcal{A}$ is (n, f)-respecting iff $\mathcal{A}$ spawns n nodes among whom at most f can be adaptively corrupt.

Modeling ideal signatures. Throughout the paper, the only cryptographic primitive we adopt is digital signatures. Although for ease of presentation we assume the existence of ideal signatures, in a practical implementation we can replace the ideal signatures with any secure digital signature scheme and a public-key infrastructure (PKI). Below we elaborate on our use of ideal signatures and how to interpret our theorems when ideal signatures are replaced with a cryptographically secure digital signature.

We adopt the standard ideal signature model: assume that each node i has a unique public key pk_i that is included in a public-key infrastructure (PKI). Nodes can call an ideal signing algorithm to sign each message they sent; and the signing algorithm always outputs a globally unique string for each signing attempt made by any node. Furthermore, nodes can call a verification algorithm, which, upon receiving a tuple $(\mathsf{pk}_i, \mathsf{m}, \sigma)$, outputs 1 iff 1) there is some $i \in \{0, 1, \ldots, n-1\}$ whose public key matches pk_i, and 2) the signing algorithm was called earlier by i with the input m, and moreover σ was produced as the result of this call. Otherwise, the verification algorithm returns 0.

All of our theorems hold deterministically for even computationally unbounded adversaries in the ideal signature model. When the ideal signature is replaced with any cryptographically secure signature scheme, all of our theorems hold for all but a negligible fraction of the execution traces, in the presence of any probabilistic, polynomial-time adaptive adversary. More precisely, in all execution traces in which there are no successful signature forgeries in the union of honest nodes' views, the security properties we prove (including consistency, validity, and liveness) are respected. We stress that our technique of modeling signatures as ideal signatures and then instantiating with real-world signature schemes is *standard, cryptographically sound, and customarily adopted* in the literature [25].

2.2 Byzantine Agreement: Definitions

For simplicity, we begin our exposition not with state machine replication which aims to confirm a linearly ordered log through repeatedly reaching consensus, but with the 1-bit single-shot version, i.e., Byzantine Agreement [32]. We formally define Byzantine Agreement below. Later in Section 5, we describe how to naturally extend our protocol to support state machine replication.

In a Byzantine Agreement protocol, there is a designated sender that is part of the common knowledge. Henceforth, without loss of generality, *we may assume that node* 0 *is the designated sender.*

Syntax. Prior to protocol start, the sender receives an input $b \in \{0, 1\}$ from $\mathcal{A}$. At the end of the protocol, every node i (including the sender) outputs a bit b_i and the output is also sent to $\mathcal{A}$.

Security definition. A Byzantine Agreement protocol must satisfy consistency, validity, and T_{end}-termination. Specifically, let $T_{\text{end}}(n)$ be a function in n denoting the termination time. We say that a protocol Π satisfies consistency, validity, or T_{end}-termination w.r.t. $\mathcal{A}$ iff the following properties hold in an execution involving the adversary $\mathcal{A}$:

- *Consistency.* If an honest node outputs b_i and another honest node outputs b_j, then it must hold that $b_i = b_j$.
- *Validity.* If the sender remains honest till the end of the execution and the sender's input is b, then all honest nodes must output b.
- T_{end}*-termination.* By the end of round T_{end}, all honest nodes output a bit.

We say that a protocol Π satisfies consistency, validity, or T_{end}-termination in (n, f)-environments iff for every deterministic $\mathcal{A}$ that is (n, f)-respecting w.r.t. Π, Π satisfies consistency, validity, or T_{end}-termination w.r.t. $\mathcal{A}$. Note that *since we prove security in the ideal signature model for unbounded adversaries, without loss of generality it suffices to prove security against every deterministic adversary* (that respects some corruption budget).

3 A Determinisitic, Longest-Chain Protocol

As mentioned, for clarity, we begin our exposition with byzantine agreement (i.e., 1-bit single-shot consensus). Later in Section 5 we show how our protocol naturally extends to state machine replication (i.e., agreeing on a linearly ordered log through repeated consensus).

3.1 Useful Definitions and Notations

Leader. In every round i, node $(i \mod n)$ is the leader. Thus each round has a unique leader.

Valid blockchain. A blockchain of length L is an ordered list of blocks where each block is of the form $\mathsf{chain} := \{(t_\ell, \mathsf{data}_\ell, \sigma_\ell)\}_{\ell \in [L]}$ where $t_\ell \in \mathbb{N}$ denotes the purported round in which this block was created (also referred to the block's *timestamp*), data_ℓ denotes an arbitrary payload string, σ_ℓ denotes a signature of the chain upto length ℓ.

Henceforth given a blockchain denoted $\mathsf{chain} := \{(t_\ell, \mathsf{data}_\ell, \sigma_\ell)\}_{\ell \in [L]}$, we define $\mathsf{extract}(\mathsf{chain})$ to be the operator removes the signature from every block outputting $\{(t_i, \mathsf{data}_i)\}_{i \in [L]}$. We use $\mathsf{chain}[i]$ to denote the i-th block in chain and we use $\mathsf{chain}[: i]$ to denote the prefix of chain upto the i-th block (inclusive).

A blockchain $\mathsf{chain} := \{(t_\ell, \mathsf{data}_\ell, \sigma_\ell)\}_{\ell \in [L]}$ is said to be valid iff

- For any $1 \leq i < j \leq L$, it holds that $t_i < t_j$;
- For any $\ell \in [L]$, σ_ℓ is a valid signature of the prefix $\mathsf{extract}(\mathsf{chain}[: \ell - 1])$ under pk_i where $i = t_\ell \mod n$ is the leader of round t_ℓ.

Remark 1. *Note that a practical optimization is to build a hash chain: let $h_0 = 0$ and let $h_i := H(h_{i-1}, t_i, \mathsf{data}_i)$. In this case, the signature for the i-th block would be computed over h_i; and further we include h_i in the block* $\mathsf{chain}[i]$.

3.2 Protocol Π_{chain}

- Round 0: sender (i.e., node 0) signs its input and multicasts the input along with the signature.
- For every round $r = 1, 2, \ldots$, every node (including the sender) performs the following:
  1. Receive chains from the network. Discard any invalid chains and any chains containing timestamps greater than or equal to r. Chains that are not discarded are considered part of the nodes' view so far.
  2. Let chain be the longest valid blockchain that has been observed in the node's view so far[1]. If there are multiple such longest chains, break ties arbitrarily. If no such chain has been observed, let chain be the empty chain.
  3. If the current node is the leader of round r, then let data be the set

     $$\left\{ (b, \sigma_b) : \begin{array}{l} b \in \{0,1\} \text{ and at least one valid} \\ \text{signature } \sigma_b \text{ has been observed for } b \end{array} \right\}$$

     Sign $\mathsf{extract}(\mathsf{chain})||(r, \mathsf{data})$ thus obtaining the signature σ; and multicast the new chain

     $$\mathsf{chain}||(r, \mathsf{data}, \sigma)$$

- At the beginning of round T_{end} (the choice of T_{end} will be stated later), do the following. Let chain be the longest valid blockchain observed so far. In all blocks in chain whose timestamp is at most n, look for the first bit b with a valid signature from the sender, and output the bit b. If such a bit is not found, output 0.

Parameters. We will prove that the above protocol satisfies the definition of Byzantine Agreement for $T_{\text{end}} \geq 2n + 1$, as long as the adversary controls fewer than $\frac{n}{3}$ nodes.

Note that our consistency proof actually shows that in round $r > n$, all blocks in an honest node's chain with round numbers $r - n$ or smaller must have stabilized. However, here the termination parameter T_{end} is set to $2n+1$ to account for the fact that the earliest block containing the sender's signature may not be the first block in a blockchain.

[1]Henceforth this is referred to as the node's longest chain in the current round r. Note that here we refer to the longest chain at the *beginning* of the round.

Remark 2 (Clarification about late-arriving blocks). *If a* chain *ending at a block with timestamp t arrives late in round $r > t+1$, a node cannot simply discard it or deem the* chain *invalid — even though discarding late-arriving messages is a standard technique in some classical-style synchronous consensus protocols [9]. In our protocol, an adversary may extend a chain with a corrupt block with timestamp t^*, make some honest nodes (denoted S) receive it in round t^*+1 but make others (denoted S') receive it in later rounds. If S' simply discarded the late-arriving block, honest nodes would make inconsistent decisions regarding whether each* chain *is valid and our longest-chain-style consensus would then break.*

3.3 Formal Protocol Analysis

For clarity, in the main body of the paper, we first present a simpler proof but for worse termination parameters. Specifically, let n denote the total number of nodes; let f denote the number of corrupt nodes; and let $\epsilon := \frac{1}{3} - \frac{f}{n}$. In this looser but simpler proof, we will set $T_{\text{end}} := (1 + \lceil \frac{1}{\epsilon} \rceil)n$. In particular, if $n = 3f+1$, then the protocol will run for $\Theta(n^2)$ number of rounds.

Later in Section A, we will present a tighter proof for the termination parameter $T_{\text{end}} = 2n+1$. However, this tighter proof requires several additional non-trivial techniques.

3.3.1 Additional Terminology

We define the following helpful terminology.

- *Execution.* We use the notation exec to denote an execution that is determined by the adversary $\mathcal{A}$.
- *Blocks.* For convenience, when the context is clear, we also use the term "the i-th block in chain" as an alias for the prefix of chain upto the i-th block, i.e., chain[: i]. Given two blockchains chain and chain$'$, the two length-ℓ blocks chain[: ℓ] and chain$'$[: ℓ] are said to be distinct iff extract(chain[: ℓ]) and extract(chain$'$[: ℓ]) are distinct (i.e., we ignore the signature when defining distinctness for blocks).
- *Honest-forever and eventually-corrupt blocks.* A block whose timestamp corresponds to an honest-forever leader round is said to be an *honest-forever block*; otherwise it is said to be an *eventually-corrupt block.*
- *Honest chain.* If in some execution exec, some honest node's longest chain in round r is chain, then we say that chain is an *honest chain* in exec (or an honest chain in round r in exec).
- *Honest-forever and eventually corrupt leader round.* A round t is said to be an honest-forever leader round if node ($t \mod n$) remains honest forever in exec; else t is said to be an eventually-corrupt leader round.

All of our proofs below hold in the ideal signature model.

3.3.2 Validity and Liveness

We first prove validity and liveness.

Special boundary blocks. In the remainder of this section, we shall assume that an honest node's longest chain in some round r consists of *i)* an imaginary "end-of-chain block" whose timestamp is

r, and whose length is the original chain length plus 1; and *ii)* an imaginary genesis block whose timestamp is 0 and whose chain length is 0.

The special genesis and end-of-chain blocks are assumed to be honest-forever blocks.

Lemma 1 (Chain growth)**.** *Let* chain *denote some node's longest chain in some round. Suppose that* $\mathsf{chain}[\ell] := (t, \_, \_)$ *and* $\mathsf{chain}[\ell'] := (t', \_, \_)$ *are both honest-forever blocks (including genesis or end-of-chain blocks) in* chain *and moreover* $t' - t - 1 \geq kn$ *for some positive integer* k*. It holds that* $\ell' - \ell - 1 \geq k(n - f)$*.*

Proof. Let exec be an execution trace defined by $\mathcal{A}$. Suppose that in some round r in exec an honest node i is the leader and suppose that node i's longest chain is of length ℓ_r at the beginning of round r. By definition of the protocol, every honest-forever node's longest chain will be of length at least $\ell_r + 1$ at the beginning of round $r + 1$.

Now, observe that in every n consecutive rounds, at least $(n - f)$ rounds have leaders that are honest-forever. Now, between (t, t'), there are $t' - 1 - (t + 1) + 1 = t' - t - 1 \geq kn$ rounds. Therefore, between (t, t'), there are at least $k(n - f)$ rounds with honest-forever leaders. Since there is an honest-forever leader for round t and one for round t'. Thus between $[t, t']$ there are at least $k(n - f) + 2$ rounds with honest-forever leaders.

Observe that for the honest node $(t \mod n)$, its longest chain must be of length $\ell - 1$ at the beginning of round t (that is why it signed a block of length ℓ in round t); and similarly, for the honest node $(t' \mod n)$, its longest chain must be of length $\ell' - 1$ at the beginning of round t'. We have that

$$\ell' - (\ell - 1) \geq k(n - f) + 2$$

and thus $\ell' - \ell - 1 \geq k(n - f)$. □

Lemma 2 (Chain quality)**.** *Let* chain *denote an honest node's longest chain in round* $r + 1$*. Then, for any* n *consecutive rounds* $[t + 1, t + n] \subseteq [r]$*, there must be an honest-forever block in* chain *with a timestamp in the range* $[t + 1, t + n]$*.*

Proof. Consider any execution exec defined by $\mathcal{A}$. In some honest chain chain, there exist honest-forever blocks $\mathsf{chain}[\ell] := (t, \_, \_)$ and $\mathsf{chain}[\ell'] := (t', \_, \_)$ such that 1) there are no honest-forever blocks in between; and 2) $t' - t - 1 \geq n$. Let $k \geq 1$ be the largest integer such that $t' - t - 1 \geq kn$; obviously, $t' - t - 1 < (k+1)n$. By Lemma 1, $\ell' - \ell - 1 \geq k(n - f)$. Now, all blocks $\mathsf{chain}[\ell + 1..\ell' - 1]$ are eventually-corrupt blocks with distinct timestamps in the range (t, t') — here the notation $\mathsf{chain}[\ell + 1..\ell' - 1]$ means the part of chain from length $\ell + 1$ to $\ell' - 1$ inclusive. Since there are at most f eventually-corrupt nodes, we have that at most $(k + 1)f$ of the timestamps in the range (t, t') correspond to eventually-corrupt leaders.

To reach a contradiction, it suffices to observe that the following cannot be true given that $n \geq 3f + 1$ and $\ell' - \ell - 1 \geq k(n - f)$:

$$\ell' - \ell - 1 \leq (k + 1)f$$

□

Theorem 1 (Validity and liveness)**.** *Suppose that* $n \geq 3f + 1$*. Then, every honest-forever node will output a bit (i.e., liveness holds). Moreover, if the sender is honest-forever, then every honest-forever node must output the sender's input bit (i.e., validity holds).*

Proof. Liveness trivially holds by protocol definition. For validity, if the sender is honest-forever, it will sign its input bit and multicast it in round 0. Due to Lemma 2, at the beginning of round T_{end}, any honest node's longest chain must have an honest-forever block whose timestamp is between $[1, n]$. By protocol definition, this honest-forever block must contain the sender's signature on its input bit. □

3.3.3 Consistency

Fact 1. *Suppose that node i is an honest-forever leader in some round t and signed a chain of length ℓ in round t; further, suppose that node j is an honest-forever leader in some round $t' \neq t$ and it signed a chain of length ℓ' in round t': we have that $\ell \neq \ell'$.*

Proof. Straightforward by protocol definition. When node i signs a chain of length ℓ in round t, in every round $r > t$, every honest node's longest chain must be of length at least ℓ and thus no honest node will ever sign a chain of length ℓ in any round $r > t$. □

Fact 2. *Let chain and chain' denote two honest chains in some execution exec. Suppose that $\mathsf{chain}[\ell]$ and $\mathsf{chain}'[\ell]$ are both honest-forever blocks. Then, it holds that $\mathsf{chain}[:\ell] = \mathsf{chain}'[:\ell]$.*

Proof. Straightforward from Fact 1. □

Given two blockchains chain and chain', if $\mathsf{extract}(\mathsf{chain}[:\ell]) = \mathsf{extract}(\mathsf{chain}'[:\ell])$, we say that chain and chain' share the block $\mathsf{chain}[:\ell]$.

Lemma 3. *Let chain and chain' be the longest chains of two honest nodes at the beginning of round T_{end}. Suppose that the last honest-forever block shared by $\mathsf{extract}(\mathsf{chain})$ and $\mathsf{extract}(\mathsf{chain}')$ has the timestamp t^*. It must hold that $T_{\text{end}} - t^* - 1 < n \cdot \lceil \frac{1}{\epsilon} \rceil$.*

Proof. Suppose not, i.e., suppose that $T_{\text{end}} - t^* - 1 \geq n \cdot \lceil \frac{1}{\epsilon} \rceil$. Let $k \geq \lceil \frac{1}{\epsilon} \rceil$ be the largest integer such that $T_{\text{end}} - t^* - 1 \geq kn$. Recall that "block" is used as an alias of a prefix of a blockchain and that distinctness of two blocks is defined disregarding the signatures (see Section 3.3.1).

Due to chain growth (Lemma 1), there are at least $k(n-f)$ blocks after time t^* in either chain or chain' (not counting the end-of-chain special boundary block). Due to chain quality (Lemma 2) and Fact 2, chain and chain' must diverge in round $t^* + n$ — otherwise the block at t^* cannot be the latest shared honest-forever block of chain and chain'. This means that the number of distinct blocks in chain and chain' after time t^* must be at least $2k(n-f) - n$.

On the other hand, during every honest-forever leader round, only a unique block will be signed by the honest-forever leader. For every eventually-corrupt round t, chain and chain' can each contain a block with the timestamp t. Recall that due to the definition of k, it must be that $T_{\text{end}} - t^* - 1 < (k+1)n$, i.e., at most $(k+1)n$ rounds have elapsed between $t^* + 1$ and T_{end}. Therefore, we have that the total number of distinct blocks after time t^* in chain and chain' must be upper bounded by $(k+1)f + (k+1)n$.

Our goal is to show the following inequality which would suffice for reaching a contradiction:

$$2k(n-f) - n > (k+1)f + (k+1)n$$

Plugging in $f = (\frac{1}{3} - \epsilon)n$, it suffices to show that $2k(\frac{2}{3} + \epsilon)n - n \geq (k+1)(\frac{1}{3} - \epsilon)n + (k+1)n$. Simplifying the above, it suffices to show that $2k(\frac{2}{3} + \epsilon) - 1 \geq (k+1)(\frac{1}{3} - \epsilon) + (k+1)$, i.e., $\frac{4}{3}k + 2k\epsilon - 1 \geq \frac{1}{3}k + \frac{1}{3} - k\epsilon - \epsilon + (k+1)$, i.e., $\epsilon + 3k\epsilon > 2 + \frac{1}{3}$. The last inequality holds as long as $k \geq \lceil \frac{1}{\epsilon} \rceil$. □

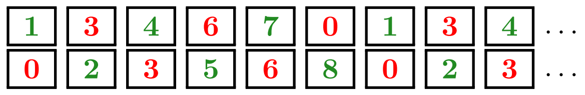
Theorem 2 (Consistency for Π_{chain}). *Let $T_{\text{end}} := (1 + \lceil \frac{1}{\epsilon} \rceil)n$. Then, the longest-chain protocol Π_{chain} satisfies consistency in (n, f)-environments as long as $n \geq 3f + 1$.*

Proof. Straightforward from Lemma 3 and the definition of the protocol. $\square$

Tighter proof. In Section A, we present a tighter but more sophisticated proof for $T_{\text{end}} := 2n+1$. The most technical part of the proof is a combinatorics argument that a certain good event called a "pivot" happens every n rounds.

3.4 An Explicit Attack with $\frac{1}{3}$ Corruption

We now show that our analysis is tight in resilience by demonstrating a $\frac{1}{3}$ attack against the deterministic blockchain protocol Π_{chain}. We consider 9 nodes numbered $0, 1, \ldots, 8$, and every node i that is a multiple of 3 is corrupt; every remaining node is honest. Note that exactly $\frac{1}{3}$ of the nodes are corrupt. The goal of the adversary is to maintain two equal-length chains forever; and he does so by diverging the honest nodes' mining attempts. The attack is graphically illustrated below where red blocks denote corrupt blocks and green blocks denote honest blocks — we will explain the attack in detail below.



We now explain how the attack works — we assume that when there are two chains of equal length, the adversary can control how honest nodes break ties, for example,

- imagine that the protocol prefers chains received earlier, the adversary can deliver one chain slightly earlier than the other within the same round[2].
- if the protocol prefers chains that end a block lexicographically smaller, the adversary can adjust the field `data` of a corrupt block to manipulate how honest nodes break ties;

We stress that our proofs earlier indeed hold even when the adversary can choose how honest nodes break ties arbitrarily.

For convenience, we number the rounds starting from 0 too. Henceforth, without risk of ambiguity, we use a block's timestamp to denote the block.

- Round 0: corrupt node 0 signs a block but he does not release this block yet.
- Round 1: the honest node 1 signs a block. At this moment, the adversary releases the 0-block.
- Round 2: the honest node 2 sees two longest chains, the "0"-chain and the "1"-chain respectively. He chooses to extend the "0"-chain and signs the next block 2.
- Round 3: corrupt node 3 extends the "1"-chain and releases this block. Additionally, he also extends the "0-2"-chain, but he withholds this block.
- Round 4: the honest node 4 extends the "1-3"-chain. In the same round, the adversary releases the "0-2-3"-chain.

[2]This makes more sense when we imagine that each round is in fact a super-round consisting of $\Delta > 1$ rounds. As discussed in Section 2.1, our protocols in fact work in a generalized synchronous model where Δ may be greater than 1; but assuming $\Delta = 1$ in our description is without loss of generality since we can always rename Δ rounds as one super-round and process all messages received only at super-round boundaries.

- Round 5: the honest node 5 sees two longest chains, the "1-3-4"-chain and the "0-2-3"-chain. He chooses to extend the "0-2-3"-chain.
- Round 6: corrupt node 6 extends the "1-3-4"-chain and releases this block. Additionally, he also extends the "0-2-3-5"-chain but he withholds this block.
- Round 7: the honest node 7 extends the "1-3-4-6"-chain. In the same round, the adversary releases the "0-2-3-5-6"-chain.
- Round 8: honest node 8 sees two longest chains "1-3-4-6-7" and "0-2-3-5-6" respectively. He chooses to extend the chain "0-2-3-5-6".
- Round 9: corrupt node 0 extends the "1-3-4-6-7" chain with the next block, and releases this block. Additionally, he extends the "0-2-3-5-6-8" chain with the next block but withholds this block.
- This goes on.

Summarizing, we have shown an attack in which both forks can be maintained forever and thus no matter how many blocks an honest node chops off from the end, consistency can be violated.

We note that one way to prevent the adversary from arbitrarily manipulating honest nodes' tie-breaking is to specify in the protocol certain tie-breaking rules. Indeed Aura's specific instantiation introduces such a tie-breaking rule. In the next section, we will show even though Aura's tie-breaking rule limits the ability of the adversary to affect honest nodes' tie-breaking behavior when encountering equal-length forks, we can nonetheless construct a $\frac{3}{7}$-attack that breaks consistency no matter how many blocks are chopped off for consistency.

4 Analysis of Aura's Instantiation

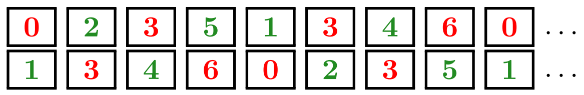
Based on the insights gained from our earlier mathematical analysis, we turn our attention to Aura's specific instantiation. In this paper, the specific version of the Aura implementation we analyzed is the snapshot that dates back to September, 2018. As mentioned, Aura's specific choice and claim are

1. they chop off only $\lfloor \frac{n}{2} \rfloor + 1$ blocks for finalization;
2. they employ the following tie-breaking rule: for two equal-length chains, they prefer the chain whose last block has an earlier timestamp and
3. they claim to defend against any minority, Byzantine adversary [4].

We now reflect on Aura's choices and claims.

A $\frac{3}{7}$-attack under Aura's specific tie-breaking rule. Recall that earlier we argued that $\frac{1}{3}$ is the best we can hope for when the adversary is allowed to arbitrarily choose how honest nodes breaks ties (when encountering equal-length chains). Aura adopts a specific tie-breaking rule (mentioned above) in their protocol that does not give $\mathcal{A}$ complete freedom to choose how honest nodes break ties.

We now demonstrate a $\frac{3}{7}$-attack against consistency even with Aura's specific tie-breaking rule. The attack is depicted below — based on this illustration, the detailed description of the attack is similar to Section 3.4. Note that this attack can maintain two equal length forks forever, thus even if honest nodes chop off an arbitrarily large number of blocks at the end for finalization, consistency can still be broken.



Reduced resilience when chopping off fewer blocks. We now consider what resilience Aura can achieve by chopping off only $\lfloor \frac{n}{2} \rfloor + 1$ blocks with distinct signers (and using their specific tie-breaking rule). We show that a consistency attack is possible if the adversary controls just slightly more than $\frac{3}{8}$ fraction of the nodes. The attack depicted above has the following generalization: $3f$ corrupt nodes can sustain two equal-length forks of length $4f$ (each fork having distinct signers). Thus with $3f$ corrupt nodes we have to chop off at least $4f$ blocks (if not more) for consistency. Now suppose $n = 8f - 3$, then Aura would actually chop off only $4f - 1$ blocks and this would lead to a consistency attack when $3f$ out of $8f - 3$ nodes are corrupt.

Finally, it is not difficult to generalize our proof to see that if one actually chops off only $\lfloor \frac{n}{2} \rfloor + 1$ trailing blocks with distinct signers, we can still guarantee consistency and liveness as long as $f < \frac{n}{6}$. This means that Aura's concrete instantiation actually gives a resilience parameter that is somewhere between $\frac{1}{6}$ and $\frac{3}{8}$, and they cannot defend against corrupt minority as they claim.

5 State Machine Replication

Byzantine Agreement allows us to reach agreement only once. In practical applications, we often need to reach agreement many times establishing a *linearly ordered log* (e.g., of transactions). A protocol that allows a set of nodes to agree on an ever-growing, linearly ordered log is called a *state machine replication* protocol [38, 39, 42].

5.1 Definitions: State Machine Replication

A state machine replication protocol can be defined as below [38, 39, 42].

Syntax. In a state machine replication protocol, in every round, an honest node receives as input a set of transactions txs from $\mathcal{A}$ at the beginning of the round, and outputs a LOG collected thus far to $\mathcal{A}$ at the end of the round.

Security. Henceforth let $T_{\text{confirm}}(n)$ be a function in n that denotes the confirmation time.

Definition 1 (Security of a state machine replication protocol)**.** We say that a state machine replication protocol Π satisfies consistency (or T_{confirm}-liveness resp.) w.r.t. some adversary $\mathcal{A}$, iff in an execution of Π involving $\mathcal{A}$ (henceforth denoted exec) consistency (or T_{confirm}-liveness resp.) is satisfied:

- *Consistency*: the execution exec satisfies consistency iff the following holds:
  - *Common prefix.* Suppose that in exec, an honest node i outputs LOG at time t, and an honest node j outputs LOG' at time t' (i and j may be the same or different), it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that in exec, a node i is honest during $[t, t']$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.

- *Liveness*: the execution exec satisfies T_{confirm}-liveness iff the following holds: if in some round $t \leq |\mathsf{exec}| - T_{\text{confirm}}$, some node honest in round t either received from $\mathcal{A}$ an input set txs that contains some transaction tx or has tx in its output log in round t, then, for any node i honest at any time $t' \geq t + T_{\text{confirm}}$, let LOG be the output of node i at time t', it holds that $\mathsf{tx} \in \mathsf{LOG}$.

  Intuitively, liveness says that transactions input to an honest node get included in honest nodes' LOGs within T_{confirm} time; and further, if a transaction appears in some honest node's LOG, it will appear in every honest node's LOG within T_{confirm} time.

We say that a state machine replication protocol Π satisfies consistency and T_{confirm}-liveness in (n, f)-environments iff for every deterministic $\mathcal{A}$ that is (n, f)-respecting w.r.t. Π, Π satisfies consistency and T_{confirm}-liveness w.r.t. $\mathcal{A}$.

5.2 A Longest-Chain-Based SMR Protocol Π_{smr}

Although our earlier Byzantine Agreement protocols are described for the binary case (i.e., when the sender's input is a bit), it is easy to modify the protocols to support *multi-valued Byzantine Agreement* (i.e., when the sender's input is an arbitrary string rather than a single bit). In the synchronous model, it is relatively easy to construct state machine replication (SMR) from *multi-valued Byzantine Agreement* (MV-BA), e.g., in each round, fork n instances of MV-BA where node i is the sender in the i-th instance; and the final log would be the concatenation of the output of all these MV-BA instances [41]. This simple transformation from MV-BA to SMR, however, incurs a rather large overhead.

One advantage of longest-chain-style protocols is that these protocols themselves naturally give rise to state machine replication protocols. For example, our longest-chain protocol described in Section 3 requires only a couple simple modifications to become a state machine replication protocol (henceforth denoted Π_{smr}):

- *Transaction propagation.* Upon receiving a set of transactions txs from $\mathcal{A}$, multicast txs to everyone.

- *Block content.* Whenever creating a block, instead of placing in the data field any bit that has a signature from the sender, place in the data field any transaction the node has observed but has not appeared in the current prefix of the blockchain.

- *Output log.* At the end of every round r, let chain be the longest chain at the beginning of this round — output the prefix of chain with timestamps at most $r - n$.

We shall prove that Π_{smr} satisfies the following theorem assuming ideal signatures.

Theorem 3. *The state machine replication protocol Π_{smr} satisfies consistency and $(2n+1)$-liveness in (n, f)-environments as long as $n \geq 3f + 1$.*

The proofs are a somewhat straightforward extension of our proofs for Byzantine Agreement. We present the proofs in the next sub-section.

5.3 Proofs for State Machine Replication

We now prove Theorem 3. Note that to obtain confirmation time $T_{\text{confirm}} := 2n + 1$, the proofs here build on the conclusions from our tighter analysis in Section A.

The consistency proof is implied by Theorem 4. We now prove the liveness of Π_{smr} for $T_{\text{confirm}} := 2n + 1$. It suffices to prove the following lemma assuming the ideal signature model.

Lemma 4. *The following claims hold:*

a) Suppose that $\mathcal{A}$ inputs a set containing the transaction tx *to some honest node in round r, then in round $r + 2n + 1$, every honest node's output must contain* tx*; and further,*

b) Suppose that in some round r some honest node's output includes the transaction tx*, then in any round $r' \geq r + n + 1$, any honest node's output must contain* tx *as well.*

Proof. Claim (a) holds due to the following. If some honest node i receives tx from $\mathcal{A}$ in some round r, it will multicast tx to everyone in round r. Thus all honest nodes will have received tx at the beginning of round $r + 1$. In round $r' = r + 1 + 2n$, let chain denote the longest chain of some honest node i, i will output to $\mathcal{A}$ the prefix of chain whose timestamps are at most $r + n + 1$ by protocol definition. By chain quality[3] (Lemma 2), in chain, there must be an honest block whose timestamp is in the range $[r + 1, r + n + 1]$ — by honest protocol definition, this honest block (or the prefix chain till this block) must contain tx.

Claim (b) holds due to the following. Suppose that some honest node's longest chain in round r is chain, and some block in chain with the timestamp $t \leq r - n$ contains tx (and thus the node outputs tx to $\mathcal{A}$ in round r). Now, in round $r' = r + n + 1$, every honest node will output a prefix of their longest chain then containing all blocks whose timestamps are no greater than $r + 1$. Due to chain quality, there is at least one honest-forever block whose timestamp is between $r - n$ and $r + 1$. The remainder of the proofs follows due to consistency. $\square$

6 A Largest-Set Protocol: Textbook Construction and Proof

In this section, we describe a variant of a deterministic, longest-chain protocol. In this variant, when a node is elected as a leader in a round, it votes on the item with the most number of votes seen so far. We call this protocol a "largest-set" protocol (as opposed to "longest-chain"). As we shall see, this variant enables a very simple proof and we recommend it for pedagogical purposes — and we use the single-shot version, Byzantine Agreement, for simplicity. Interestingly, the proofs for this protocol can be viewed as a simplified, deterministic variant of the analysis of Nakamoto's blockchain [36, 37, 39].

6.1 Useful Definitions

Leader. In every round i, node ($i \mod n$) is the leader. Thus each round has a unique leader.

Valid votes. A valid vote for a normal bit $b \in \{0, 1\}$ is a tuple (b, t, i, σ) where $i \in [0..n - 1]$ is a node identifier, $t \geq 1$ is a round number (also called the vote's *timestamp*), and σ is a valid signature under pk_i for the tuple (b, t). A valid vote for the dummy bit $\perp$ is a tuple $(\perp, t, i, \sigma)$ where $i \in [0..n - 1]$ is a node identifier, $t \geq T_\perp$ is a round number no earlier than $T_\perp$, and σ is a valid signature under pk_i for the tuple $(\perp, t)$.

Two votes (b, t, i, σ) and (b', t', i', σ') are said to be *distinct* iff $(b, t, i) \neq (b', t', i')$.

Vouch for. Given an execution trace exec, we define "vouch for" as below:

[3]Note that Lemma 2 holds for Π_{smr} too in the same way it holds for Π_{chain}.

- If $b \in \{0,1\}$ is a normal bit and node i does not have the sender's signature on $(\texttt{propose}, b)$ in its view at the beginning of round r, then we say that there are no votes vouching for b w.r.t. node i and round r.

- Else, the number of votes vouching for $\widetilde{b} \in \{0,1,\perp\}$ w.r.t. node i and round r is defined to be the number of distinct valid votes for $\widetilde{b}$ in node i's view at the beginning of round r.

If $\widetilde{b} \in \{0,1,\perp\}$ has X votes vouching for it w.r.t. node i and round r in some execution $\texttt{exec}$, we sometimes also say: at the beginning of round r, node i has X votes vouching for $\widetilde{b}$ in $\texttt{exec}$. If $\widetilde{b}$ has the most number of votes vouching for it (in comparison with any other bit) w.r.t. node i and round r, we say that node i perceives $\widetilde{b}$ as the *most popular* bit in round r.

6.2 A Largest-Set Protocol Π_{set}

We now describe a simple largest-set protocol that realizes Byzantine Agreement as defined in Section 2.2. In a longest-chain protocol such as Nakamoto's blockchain [36] (and see also Section 3), nodes vote by extending the longest chain; in this largest-set protocol, there is no such concept of a chain, and nodes vote on the bit that so far has the largest number of votes vouching for it (i.e., the most popular bit).

Imprecisely speaking, in every round, every node looks for a bit signed by the sender and carrying the most number of votes, and then signs that bit. At the end of the protocol, whichever bit has the most number of signatures will be the bit output (i.e., decided). One issue is that if the sender is corrupt, it may not sign any bit. To handle this case, we introduce a dummy bit denoted $\perp$ sometime $T_\perp$ into the protocol — nodes are allowed to sign $\perp$ $T_\perp$ time into the protocol start. $T_\perp$ is chosen to be large enough such that if the sender did sign a bit upfront, the dummy bit $\perp$ cannot outnumber a normal bit in terms of number of votes and thus honest nodes will agree on a normal bit. On the other hand, $T_\perp$ must be chosen to be small enough (w.r.t. to the protocol duration T_{end}) such that the protocol retains consistency.

More concretely, the protocol works as follows.

- *Round 0:* let b be the sender's input bit. The sender signs $(\texttt{propose}, b)$ and multicasts the bit and the resulting signature.

- *For round $r = 1, 2, \ldots, T_{\text{end}}$:*

  – Receive all messages from the network and discard every vote whose timestamp is r or greater (discarded votes do not become part of the node's view in the protocol).

  – If the current node is the leader of this round, perform the following steps (otherwise skip). Let $\widetilde{b} \in \{0,1,\perp\}$ be the most popular bit (w.r.t. the current node and round r), sign $(\widetilde{b}, r)$, and multicast 1) the resulting vote; 2) all valid votes the node has for $\widetilde{b}$ so far in its view; and 3) a sender's signature on $(\texttt{propose}, \widetilde{b})$ if applicable. If there are multiple most popular bits, break ties arbitrarily.

- At the beginning of round $T_{\text{end}} + 1$: Find the most popular bit $\widetilde{b} \in \{0,1,\perp\}$ breaking ties arbitrarily. If $\widetilde{b} \in \{0,1\}$, output $\widetilde{b}$. Otherwise output 0.

Parameters. Let $\epsilon = \frac{1}{3} - \frac{f}{n}$, we will choose $T_\perp = n+1$, and $T_{\text{end}} = 2n \cdot \lceil \frac{1}{\epsilon} \rceil$. For example, if $n = 3f+1$, i.e., $\epsilon = \frac{1}{3n}$, then the protocol will run for $\Theta(n^2)$ number of rounds.

6.3 Proofs

If a round's leader is honest-forever, we say that this round is an honest-forever leader round. Otherwise, we say that it is an eventually-corrupt leader round.

Lemma 5 (Vote growth)**.** *Suppose that $T_{\text{end}} - T_\perp + 1 \geq kn$. At the beginning of round $T_{\text{end}} + 1$, any honest node i's most popular bit must have at least $k(n-f)$ votes vouching for it.*

Proof. In every honest-forever leader round $r \geq T_\perp$, suppose that the leader ($r \mod n$) of round r's most popular bit at the beginning of r has X votes vouching for it, then, at the beginning of round $r+1$, every honest node's most popular bit has at least $X+1$ votes vouching for it — since in round r node ($r \mod n$) will add another vote to this most popular bit and multicast it to everyone else. The remainder of the proof follows in a straightforward manner by observing that there are at most f eventually-corrupt leader rounds among every n rounds. □

Lemma 6 (Consistency)**.** *Suppose that $f = (\frac{1}{3} - \epsilon)n$. Then, at the beginning of round $T_{\text{end}} + 1$, all honest nodes must output the same bit $\widetilde{b} \in \{0, 1, \perp\}$.*

Proof. Due to Lemma 5, at the beginning of round $T_{\text{end}} + 1$, every honest node's most popular bit must have at least $k(n-f)$ votes where $k = \lceil \frac{1}{\epsilon} \rceil - 1$ is the largest integer such that $T_{\text{end}} - T_\perp + 1 \geq kn$. Suppose, for the sake of contradiction, at the beginning of round $T_{\text{end}} + 1$, two honest node's most popular bit are different bits henceforth denoted b and b'. Then it holds that there are at least $2k(n-f) = 2k(\frac{2}{3} + \epsilon)n$ distinct votes whose timestamps are upper bounded by T_{end}.

On the other hand, note that in every honest-forever leader round, no more than one vote may be cast for either b or b' by the end of the protocol; whereas in every eventually-corrupt leader round, one vote can be cast for each of b and b' by the end of the protocol. Thus, the total number of votes cast for either b or b' by the end of the protocol is at most $(k+1)(n+f) = (k+1)(\frac{4}{3} - \epsilon)n$. We conclude that

$$(k+1)\left(\frac{4}{3} - \epsilon\right)n \geq 2k\left(\frac{2}{3} + \epsilon\right)n$$

Simplifying the above equation, we have that

$$\frac{4k}{3} + \frac{4}{3} - \epsilon k - \epsilon \geq \frac{4k}{3} + 2k\epsilon$$

Therefore,

$$\frac{4}{3} - \epsilon \geq 3\epsilon k = 3\epsilon\left(\lceil \frac{1}{\epsilon} \rceil - 1\right) \geq 3 - 3\epsilon \geq 2$$

This is impossible and thus we reach a contradiction.

□

Fact 3. *If the sender is honest in round 0, then at the beginning of round $t \geq kn+1$ in the protocol, any honest node's most popular bit must have at least $k(n-f)$ votes vouching for it.*

Proof. Due to the same argument as Lemma 5 and the fact that the sender is honest and will sign its input bit in round 0. □

Lemma 7 (Validity)**.** *If the sender is honest-forever, then, all honest nodes must output the sender's input bit.*

Proof. If the sender is honest-forever, due to Fact 3, we have that at the beginning of round $T_\perp$, any honest node's most popular bit must have at least $n - f$ votes vouching for it; moreover, its most popular bit must be the sender's input bit b^* since the sender will not sign $1 - b^*$.

Suppose for the sake of contradiction that some honest node did not output the sender's input bit b^* at the end of the protocol. Since the sender never signed $1 - b^*$, this differing output bit must be $\perp$. Let $t^* > T_\perp$ be the *first* round in which some honest node's most popular bit is $\perp$. By definition of the protocol, before round t^*, no honest node will vote for $\perp$. Thus all valid votes for $\perp$ at the beginning of round t^* must come from eventually-corrupt nodes and they must have timestamps between $[T_\perp, t^* - 1]$. The total number of eventually-corrupt votes for $\perp$ with timestamps between $[T_\perp, t^* - 1]$ must be upper bounded by the number of eventually-corrupt leader rounds between $[T_\perp, t^* - 1]$. Let k be the largest integer such that $t^* - T_\perp \geq kn$. Then, the total number of eventually-corrupt votes for $\perp$ with timestamps between $[T_\perp, t^* - 1]$ must be upper bounded by $(k + 1)f$.

On the other hand, due to Fact 3, at the beginning of round t^*, the most popular bit in any honest node's view must have at least $(k + 1)(n - f)$ votes vouching for it.

We have that

$$(k + 1)f \geq (k + 1)(n - f)$$

However, this is impossible if $n \geq 3f + 1$. □

7 Conclusion

In designing Bitcoin [36], Nakamoto proposed the first *longest-chain*-style consensus protocol, a new consensus paradigm that departs from standard techniques in classical distributed systems. Although Nakamoto's protocol relies on Proof-of-Work (PoW), subsequent works [18, 19, 27, 40] have shown how to remove PoW assuming that the initial consensus nodes' public keys are common knowledge. All of the aforementioned protocols are *randomized*, i.e., every time a *random* leader is elected to propose the next block extending the longest chain.

In this paper, we provide the *first* analysis for natural, *deterministic* embodiments of the longest-chain idea, i.e., block proposers, rotating in a deterministic fashion, propose the next block extending the current longest chain. Our results suggest that

1. the deterministic variant takes linear (in n) time to confirm transactions (*c.f.*, earlier randomized variants take only security parameter amount of time);

2. the deterministic variant resists upto 1/3 Byzantine corruptions whereas earlier randomized variants can tolerate $1/2 - \epsilon$ fraction where ϵ is an arbitrarily small constant margin.

In comparison with its randomized cousin, the deterministic variant incurs worse confirmation time; however we point out that the deterministic variant secures against a stronger adversary who is capable of "after-the-fact message removal" [8], i.e., the adversary can observe what an honest node i wants to send in some round r, adaptively corrupt node i, erase the message it originally wanted to send, and then insert arbitrary corrupt messages on behalf of node i in round r. In comparison, known randomized longest-chain protocols are secure only against a weaker adversary who cannot perform such "after-the-fact message removal" — in fact, due to a recent lower bound result by Abraham [8], protocols that reach agreement using only sublinear number of multicast messages (e.g., known randomized longest-chain protocols [18, 19, 27, 36, 39, 40]) inherently cannot tolerate "after-the-fact message removal".

We next turn our attention to deployed open-source implementations of deterministic, longest-chain-style protocols. We provide an in-depth analysis of Parity's specific instantiation called Aura [1,4]. We show that their specific embodiment and parameters can tolerate only between 1/6 and 3/8 Byzantine corruptions, thus refuting their public claim that Aura resists up to minority Byzantine corruptions.

Finally, we present a largest-set analog of the longest-chain idea: this variant admits an extremely simple proof and thus we recommend it for pedagogical purposes.

Acknowledgments

We thank Danlu Huang, Yu Xia, Yizhou Yu, Jiaheng Zhang, for helpful discussions. Through a course project in the "Distributed Consensus and Blockchains" class at Cornell, Danlu, Yizhou, Jiaheng helped the author investigate the Aura protocol and perform a security analysis. Yu Xia kindly helped the author understand the Aura implementation. We gratefully acknowledge Rafael Pass for insightful technical discussions. Finally, we thank the reviewers for the helpful comments.

References

[1] `https://github.com/paritytech/parity-ethereum/tree/master/ethcore/src/engines/authority_round`.

[2] `https://coinmarketcap.com/`.

[3] `https://github.com/EOSIO/eos/`.

[4] Aura - authority round. `https://wiki.parity.io/Aura`.

[5] DPOS consensus algorithm - the missing white paper. `https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper`.

[6] Fix dpos loss of consensus due to conflicting last irreversible block. `https://github.com/EOSIO/eos/issues/2718`.

[7] Tsinghua/Cornell joint blockchain winter school in Shenzhen, China, 2017.

[8] Ittai Abraham, T-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. *CoRR*, abs/1805.03391, 2018.

[9] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous byzantine consensus. In *Financial Crypto*, 2019.

[10] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999.

[11] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *DSN*, pages 355–362, 2014.

[12] Vitalik Buterin and Vlad Zamfir. Casper. `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/`, 2015.

[13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[14] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Communication-efficient byzantine agreement without erasures. *CoRR*, abs/1805.03391, 2018.

[15] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. Cryptology ePrint Archive, Report 2018/981, 2018. `https://eprint.iacr.org/2018/981`.

[16] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. Cryptology ePrint Archive, Report 2018/980, 2018. `https://eprint.iacr.org/2018/980`.

[17] Jing Chen and Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341.

[18] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. In *Financial Cryptography*, 2019. Cryptology ePrint Archive: Report 2016/919, `https://eprint.iacr.org/2016/919.pdf`.

[19] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. In *Eurocrypt*, 2018.

[20] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *Siam Journal on Computing - SIAMCOMP*, 12(4):656–666, 1983.

[21] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment i: Crash failures. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, TARK '86, pages 149–169, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[22] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[23] Vassos Hadzilacos. A lower bound for byzantine agreement with fail-stop processors, 1983.

[24] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series: Consensus system. `https://dfinity.org/tech`.

[25] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, February 2009.

[26] Aggelos Kiayias and Alexander Russell. Ouroboros-bft: A simple byzantine fault tolerant consensus protocol. Cryptology ePrint Archive, Report 2018/1049, 2018. `https://eprint.iacr.org/2018/1049`.

[27] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Crypto*, 2017.

[28] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 45–58, 2007.

[29] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[30] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[31] Leslie Lamport and Michael Fischer. Byzantine generals and transaction commit protocols. Technical report, 1982.

[32] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[33] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[34] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. `https://dspace.mit.edu/bitstream/handle/1721.1/107927/MIT-CSAIL-TR-2017-004.pdf?sequence=1`, 2017.

[35] Yoram Moses and Sergio Rajsbaum. The unified structure of consensus: A *Layered Analysis* approach. In *PODC*, pages 123–132. ACM, 1998.

[36] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[37] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[38] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[39] Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *CSF*, 2017.

[40] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Asiacrypt*, 2017.

[41] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.

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

[43] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

A A Tighter Proof for the Longest-Chain Protocol

In this section, we present a tight proof for our deterministic longest-chain protocol. In the main body, we presented a much simpler proof that requires chopping off upto $O(\frac{n}{\epsilon})$ round numbers for finalization where $\epsilon := \frac{1}{3} - \frac{f}{n}$. In particular, if $n = 3f + 1$, then we need to chop off quadratic number of blocks. In this section, we prove that in fact, chopping off the most recent n round numbers from the end of the chain is sufficient. This tighter proof is obviously more technical than the earlier proof in Section 3.3.

We begin by defining a good event called a *pivot*. We then argue why a pivot contributes to consistency. Finally, we present a combinatorics argument why a pivot happens every n rounds.

Pivot. Given an execution exec, a pivot is a point of time t such that for any window $[t_0, t_1]$ that contains t, there are strictly more honest-forever leader rounds than eventually-corrupt leader rounds. Our definition of a pivot is inspired by the earlier work by Pass and Shi [40].

Note that by definition, if t is a pivot in exec, then node $t \mod n$ (i.e., the leader for round t) must be an honest-forever node. Further, it is easy to see that the following fact holds:

Fact 4. *The following holds for any execution exec where $n \geq 2f+1$: t is a pivot in exec iff for any window $[t_0, t_1]$ that contains t and $t_1 - t_0 + 1 \leq n$, there are strictly more honest-forever leader rounds than eventually-corrupt leader rounds in exec.*

Proof. Straightforward due to the following observation: for any consecutive n rounds, there must be more honest-forever leader rounds than eventually-corrupt leader rounds. □

Lemma 8 (Pivot leads to convergence)**.** *Consider any execution denoted exec: suppose that t is a pivot in exec; further, suppose that chain is the longest chain belonging to an honest node in round $r > t$ in exec and chain' is the longest chain belonging to an honest node in round $s > t$ in exec. It holds that $\mathsf{extract}(\mathsf{chain})$ and $\mathsf{extract}(\mathsf{chain}')$ must agree for the prefix whose timestamps are at most t.*

Proof. Suppose that the honest-forever node ($t \mod n$) signed a block at length ℓ in exec. It suffices to prove that $\mathsf{chain}[\ell]$ and $\mathsf{chain}'[\ell]$ must both be this honest-forever block. We prove it for chain since the argument for chain' is identical.

If $\mathsf{chain}[\ell]$ is an honest-forever block, the proof follows directly by Fact 2. We thus focus on the case when $\mathsf{chain}[\ell]$ is an eventually-corrupt block. In this case, let $\mathsf{chain}[\ell_L] := (t_L, \_, \_)$ be the latest honest-forever block to the left of $\mathsf{chain}[\ell]$; if no such block exists, simply let $\ell_L := 0$ and $t_L = 0$. Similarly, let $\mathsf{chain}[\ell_R] := (t_R, \_, \_)$ be the earliest honest-forever block to the right of $\mathsf{chain}[\ell]$; if no such block exists, simply let $\ell_R := |\mathsf{chain}|+1$ and let $t_R := r$. Henceforth, we imagine that chain has two imaginary blocks at length 0 and $|\mathsf{chain}|+1$ denoting beginning and end of chain respectively. For any $\widetilde{t} \in (t_L, t_R)$, if some honest-forever leader signed a block at length ℓ in round $\widetilde{t}$, then $\mathsf{chain}[\ell]$ must be an eventually-corrupt block. Since all blocks in chain must have distinct timestamps, we have that between (t_L, t_R), there are at least as many eventually-corrupt leader-rounds as there are honest-forever leader-rounds. This contradicts the fact that t is a pivot. □

Lemma 9 (Existence of a pivot every n rounds)**.** *Suppose that $n \geq 3f+1$. Then, for every n consecutive rounds $(t, t+n]$, there must exist a pivot $t^* \in (t, t+n]$.*

Proof. **Placing numbers on a circle.** Imagine we place m numbers, either $+1$ or -1 on a circle such that fewer than $\frac{m}{3}$ of them are -1 — such a placement is henceforth said to be a *valid configuration.*

Pivot. For any $i, j \in [0..m-1]$ such that $i \neq j$, we use the notation $s^{\rightarrow}[i..j]$ to denote the cumulative sum going from the i-th number clockwise until we reach the j-th number (inclusive of both the i-th and the j-th number). We define $s^{\rightarrow}[i..i]$ to be the i-th number. $s^{\leftarrow}[i..j]$ is similarly defined but going counter-clockwise.

The i-th number where $i \in [0..m-1]$ is said to be a *pivot* iff for any $j \in [0..m-1]$, we have that $s^{\rightarrow}[i..j] > 0$ and $s^{\leftarrow}[i..j] > 0$. It is not difficult to see that this definition is equivalent to the following: the i-th number is said to be a pivot iff for any clockwise segment spanning positions $a..b$ that includes i $s^{\rightarrow}[a..b] > 0$.

Configuration induced by an execution. An execution exec will induce a configuration as explained below. We place n numbers on a circle based on exec: for $i \in [0..n-1]$, if node i is honest-forever in exec, the i-th number is $+1$; otherwise it is -1. Since we assume that $n \geq 3f+1$, it must hold that the configuration induced by exec is a valid configuration.

To prove the above lemma, it suffices to prove that the configuration induced by any exec has a pivot. We prove a generalization of the statement:

Claim 1. *Suppose that we place m numbers, either $+1$ or -1, on a circle as mentioned above. In any valid configuration (i.e., fewer than $\frac{m}{3}$ of them are -1), there must exist a pivot.*

Proof. We now prove the above claim. A *negative group* is a sequence of consecutive -1s on the circle sandwiched by $+1$s on both sides. A *positive group* is sequence of consecutive $+1$s on the circle sandwiched by -1s on both sides. If all numbers on the circle are $+1$ (or -1 resp.), all the numbers form a single positive (or negative resp.) group. If a positive (or negative resp.) group does not contain all numbers on the circle, it is said to be a *proper* positive (or negative resp.) group.

Any *proper* positive (or negative resp.) has a *left-most* number and a *right-most* number: essentially the group spans the left-most number going clockwise to the right-most number.

We now prove by induction.

Base case. All numbers form a single positive group. In this case the proof is trivial.

Induction step. If a valid configuration is not the base case, then it must contain a sequence of alternating proper positive and proper negative groups — let T denote the total number of such alternating positive and negative groups (the base case can be regarded as $T = 1$). We need to prove the following: if a pivot exists for every valid configuration where $T \leq \tau$ (i.e., induction hypothesis), then a pivot exists for every valid configuration where $T \leq \tau + 1$.

We now perform the following elimination procedure:

- *Start:* Say we start from a valid configuration C that is not all positive.
- *Find negative group:* Find an arbitrary proper negative group, let its right-most number be position i_R and the left-most number be position i_L.
- *Counter-clockwise scan:* Scan counter-clockwise from i_R until we reach the first position j such that $s^{\rightarrow}[j..i_R] = 0$.
- *Clockwise scan:* Start from the left-most number of the negative group, say the i_L-th number, and scan clockwise until we reach the first position k such that $s^{\rightarrow}[i_L..k] = 0$.
- *Elimination:* Eliminate all numbers between positions $[j..k]$, and the ending configuration is called C'.

Henceforth all numbers eliminated due to the counter-clockwise scan together comprise the "counter-clockwise elimination"; and all numbers eliminated due to the clockwise scan together comprise the "clockwise elimination". The clockwise and counter-clockwise eliminations overlap in the negative group chosen.

We now prove that the following facts hold for the above elimination procedure.

Fact 5. *A number that is pivot in C cannot be eliminated by the above procedure.*

Proof. It suffices to argue that a position p is a pivot in C cannot be eliminated during a counter-clockwise elimination (the argument is symmetric for clockwise). For p to be eliminated, it must hold that $s^{\rightarrow}[p..i] \leq 0$ where i is the right-most number of some proper negative group. This cannot hold by the definition of a pivot. □

Fact 6. *A number that is pivot in C' must be a pivot in C.*

Proof. For simplicity, we assume that the elimination simply rewrites eliminated numbers as 0s — this way, we can use the same position labels for C and C'.

Henceforth, we denote the cumulative sum $s^{\rightarrow}[a..b]$ in C as $S[a..b]$, and the cumulative sum $s^{\rightarrow}[a..b]$ in C' as $S'[a..b]$. Due to Fact 5, it suffices to prove that for any position p that is not eliminated during the elimination procedure, for any p^*, we have that $S[p..p^*] \geq S'[p..p^*]$ (and the other direction, i.e., counter-clockwise, is symmetric). We consider the following cases:

- Case 1: p^* is not among the eliminated positions. In this case, the proof is obvious since the eliminated positions sum up to 0.

- Case 2: p^* is among the eliminated positions. Recall that j is the left-most among the eliminated positions; and that i_R is the right-most number of the negative group involved in the elimination procedure. It suffices to prove that $S[j..p^*] \geq 0$. This holds since otherwise j cannot the first number scanning counter-clockwise from i_R such that $S[j..i_R] = 0$ — there must exist such a number in between positions $[p^* + 1..i_R]$.

□

Fact 7. *If the elimination procedure eliminates x number of -1s, it can eliminate no more than $2x$ number of $+1$s.*

Proof. Straightforward by observing that in both the clockwise and the counter-clockwise eliminations, there are as many -1s elimanted as there are $+1$s. Further, the -1s eliminated in both directions can overlap whereas the $+1$s cannot overlap in both directions (since it is not difficult to see that there must be numbers left after the elimination). □

Due to Facts 6 to prove the induction step, it suffices to prove that in C' there is a pivot — the latter holds due to Fact 7 and the induction hypothesis. □

□

Theorem 4 (Consistency). *Let* chain *be some honest node's longest chain in round r; and let* chain' *be some honest node's longest chain in round $t \geq r$, it holds that* chain *and* chain' *agree in the part of their prefixes with timestamps $r - n$ or smaller.*

Proof. Straightforward due to Lemma 8 and Lemma 9. □