PiLi: A Simple, Fast, and Robust Family of Blockchain Protocols[*]

T-H. Hubert Chan Rafael Pass Elaine Shi

Abstract

In every epoch, an eligible proposer proposes a next block extending from the freshest notarized chain it has seen. Everyone votes on the first proposal heard if 1) the proposed block extends from a parent block that is not too "stale", and 2) no conflicting notarizations have been observed in the recent past. When a block collects majority votes, it becomes *notarized* but not yet *final.* If a notarized chain ends at 13 consecutive epochs the trailing 8 blocks and the prefix is *final.* If a block of the present epoch collects $3/4$ fraction (and not just majority) votes, a node may advance to the next epoch immediately. Otherwise a node sends a timeout message when an epoch exipres and advances to the next epoch when majority timeout messages have been heard.

This very simple protocol can achieve the following properties where n denotes the total number of nodes: 1) during a period in which $3n/4$ honest nodes remain online, we can confirm transactions at raw network speed (i.e., asynchronously) as soon as we rotate to an honest and online proposer; 2) if more than $n/2$ of honest nodes have online presence for sufficiently long (w.r.t. the protocol's confirmation delay), transactions get confirmed in expected constant number of synchronous rounds; and 3) we guarantee consistency for *all* honest nodes — including those who might have unstable network connections and drop offline every now and then — as long as at any point of time the total number of offline and corrupt nodes is less than $n/2$. In this sense, our protocol is strictly more robust than classical, *synchronous* honest-majority consensus.

1 Introduction

At the core of decentralized cryptocurrencies is a beautiful abstraction called blockchain or state machine replication. In a blockchain protocol, nodes seek to agree on an ever-growing, linearly ordered log of transactions, such that two important security properties are satisfied: 1) *consistency*, i.e., all honest nodes' logs must be prefixes of each other and no node's log should ever shrink; and 2) *liveness*, i.e., a transaction will appear in all honest nodes' logs within a(n ideally small) bounded amount of time.

1.1 A Tale of Synchronous Consensus

The classical synchronous model assumes that messages sent by honest nodes are delivered in at most Δ rounds, and this network delay upper bound Δ must be pre-determined and hardwired in the protocol. This very strong assumption has caused two significant "folklore" barriers towards deploying synchronous consensus protocols in practice.

1. *Slowness.* With the exception of very recent works [18, 24], almost all existing synchronous protocols' confirmation delays are constrained by this pre-determined delay parameter Δ.

[*]PiLi means the sound of thunder in Chinese, it also means fast, furious, and streamlined.

Even if in reality, the network actually makes progress much faster, a typical synchronous protocol, unfortunately, is unable to reap the benefits. To make matters worse, the delay parameter Δ must be set conservatively to ensure that the assumptions needed for safety are not violated. Partly for this reason, synchronous consensus is considered "slow" in the folklore.

2. *Lack of robustness.* If an honest node has even a short-term outage, the classical synchronous model immediately treats it as corrupt — at this point, a protocol proven secure in the classical synchronous model is no longer obligated to provide consistency for this node, even if it comes back online shortly afterwards and wishes to continue participating [13]. This mismatch becomes even greater for long-running protocols like blockchains: no one can guarantee 100% up-time (e.g., Gmail has outages every couple of years [1]), and thus in the course of a few years, it could be that everyone has had an outage at some point.

   This is not just a theoretical concern. A recent work [13] has demonstrated what seems to be a serious flaw in a real-world blockchain protocol [24] being deployed by a cryptocurrency company — despite the fact that the protocol was proven secure under classical synchrony! Specifically, even when everyone is benign and a few nodes crash in a specific timing pattern, confirmed transactions can become undone (see Guo et al. [13] for more details).

Partly due to the above reasons, until decentralized cryptocurrencies such as Bitcoin emerged, synchronous consensus had remained (almost) only on paper. Decentralized cryptocurrencies such as Bitcoin adopted *synchronous* consensus protocols in practice for the first time (since Nakamoto-style consensus must set the expected block interval to be commensurate with the maximum network delay for safety [12, 21, 23]) but their slowness is evident and in fact has become a pain point that is most often debated among the cryptocurrency community.

What about partial synchrony/asynchrony? Both pain points (i.e., slowness and brittleness) can be overcome by adopting partial synchrony [7, 11] or asynchrony [5, 6], where the consensus protocol makes no assumptions about the network's maximum delay. Unfortunately, it is well-known that no partially synchronous (or asynchronous) consensus protocol can tolerate more than 1/3 corruptions [11] (*c.f.*, synchronous protocols are not subject to such a lower bound). In decentralized environments where players are mutually distrustful, a higher degree of resilience may be important.

1.2 PiLi: A New Paradigm for Synchronous Consensus

We introduce PiLi, a new family of "dream" blockchain protocols that resist *minority* corruptions. PiLi is not only conceptually extremely simple but also overcomes the aforementioned two significant "folklore" barriers.

Our protocols follow a streamlined "propose-vote" paradigm with no special execution paths for recovery (also called view change [7]). Finalization requires chopping off a few trailing blocks from a qualifying notarized chain. We provide a slightly informal description of this paradigm:

- In every epoch, a proposer proposes a block extending the freshest notarized chain it has seen, and everyone votes on the first valid proposal heard as long as 1) the proposed block extends from a parent that is not too "stale"; and 2) no conflicting notarizations have been seen recently.
- If a block gains votes from majority voters, it becomes *notarized.*

- If a notarized chain ends at X number of consecutive epochs, chop off the trailing $Y < X$ blocks and the prefix is *final*.
- If an epoch-e block gains votes from at least $3/4$ fraction of the voters, an epoch-$(e+1)$ block may be immediately proposed and voted on (and the node enters epoch $e+1$ immediately too). Otherwise, a node sends a time-out message upon expiration of the present epoch, and enters the next epoch upon receiving time-out messages from majority nodes.

We prove that this very simple protocol achieves the following properties where n denotes the total number of nodes:

1. *Optimistic responsiveness:* whenever $3n/4$ nodes are not only honest but also online for sufficiently long, then (as soon as an honest proposer is elected) transactions get confirmed "responsively" at raw network speed and moreover in $O(1)$ number of roundtrips — henceforth we denote the optimistic conditions necessary for responsiveness as $\mathbf{O}$.

2. *Liveness during periods of synchrony:* a synchronous notion of liveness is guaranteed whenever sufficiently many honest nodes have online presence for sufficiently long (relative to the confirmation time). Concretely, in any period of time during which more than $n/2$ honest nodes can deliver messages within Δ delay, these nodes can make progress in expected constant number of *synchronous* rounds — henceforth the set of conditions necessary for liveness is denoted $\mathbf{S}$.

3. *Consistency and best-possible partition tolerance:* consistency is preserved as long as the network, at any point of time, exhibits synchrony among more than $n/2$ honest nodes called the "honest and online set" — importantly, here the honest and online set may even *change rapidly* over time (and thus everyone's online presence may be transient relative to the confirmation time). Since consistency holds even for honest nodes who may be offline at times, our protocol is also said to be *best-possible partition tolerant* [13]. Henceforth the conditions necessary for consistency are denoted $\mathbf{W}$.

It is not hard to see that $\mathbf{O} \subset \mathbf{S} \subset \mathbf{W}$, i.e., the conditions for responsiveness are the most stringent and those for safety (i.e., consistency) are the weakest. In a practical deployment, the protocol should almost always operate in the optimistic regime $\mathbf{O}$ and thus one could enjoy asynchronous, fast performance. Yet we can resist the failures and node churns that are inevitable in practice (as long as majority number of honest nodes do not fail at the same time); and the minority nodes who are unstable at any point of time should never have to suffer from inconsistency.

1.3 Technical Contributions

For the first time, with an extremely simple paradigm, we overcome the two significant barriers which we believe to have significantly hindered the practicality of "synchronous consensus" in the past. Thus we hope that our work will pave the way for deploying "honest-majority" consensus protocols in the real world. We now explain our technical contributions in light of most closely related works.

Achieving optimistic responsiveness in a "streamlined-BFT" paradigm. The notion of optimistic responsiveness in synchronous consensus was first proposed in the recent work of Thunderella [24] and later extended by Loss and Moran [18] but in the context of single-shot consensus. It is known that $3/4$ honest and online is necessary for optimistic responsiveness for any protocol that aims to resist upto minority corruptions [24].

First, even without the "best-possible partition tolerance" property, our work is the first protocol in a streamlined-BFT paradigm that ensures security under honest majority and achieves optimistic responsiveness. In comparison, existing blockchain constructions that achieve optimistic responsiveness [24] have a dedicated execution path (called the "fallback" mechanism) for switching between the fast-path and the slow path. The fallback mechanism in existing works [24] is not only complicated to implement in practice [24] but also requires re-posting all notarized but uncheckpointed transactions on the fast path again to the slow path and thus is not bandwidth-efficient.

More importantly, our result is also a fundamental theoretical improvement over Thunderella [24]: as pointed out in recent work [13], Thunderella's safety property relies critically on *full* synchrony which makes their protocol alarmingly flawed in practice.

How to use "best-possible partition tolerance" in practical blockchain design. To overcome the brittleness of classical synchronous consensus, our work aims to prove safety under the aforementioned conditions $\mathbf{W}$ — this modeling approach was recently proposed by Guo, Pass and Shi [13] and they refer to such a network as a *weakly synchronous* network. In a weakly synchronous network, roughly speaking, at any point of time we allow minority nodes to have unstable network connections (or be corrupt); and these flaky nodes can rotate over time. Viewed from every single node's perspective, it could be that *every node was at some point offline and thus the network may have been asynchronous for every node*! As mentioned, a protocol secure under weak synchrony is also said to be *best-possible partition tolerant.*

Guo et al. [13] was the first to explore the theoretical feasibility of consensus protocols that are best-possible partition tolerant. The protocols proposed in their work, however, are of theoretical interest only and not recommended for implementation. In comparison, our work takes a practically grounded approach, and in this sense we are the first to show how one might use the notion of "best-possible partition tolerance" in the design of practical blockchain protocols. Since simplicity and strong-enough practical security are our first-order concerns, we in fact introduce a new modeling relaxation in comparison with Guo et al. [13]. Guo et al. [13]'s theoretical constructions achieve liveness under the same set of conditions $\mathbf{W}$ as are needed for safety. More concretely, since obviously liveness cannot be guaranteed for offline nodes for as long as they remain offline, Guo et al. [13] ensure "best-effort liveness", i.e., nodes should make progress as soon as they come online. We observe that Guo et al. [13]'s notion of liveness is in fact even stronger than the liveness promised by standard partially synchronous protocols [7, 11] — the latter commonly promise liveness during *periods of synchrony* in which a super-majority number of honest nodes are online and have good network connectivity. Inspired by this observation, we introduce an analogous *period of synchrony* notion in a weakly synchronous network. We aim to achieve worst-case liveness only during periods of synchrony in which there exists a majority set of honest nodes which are online and have good (i.e, Δ-respecting) network connectivity. We believe that this modeling relaxation could be of value in future (especially practically minded) works too since insisting on an overly stringent liveness notion may accidentally preclude practically useful protocols.

1.4 Additional Related Work

Byzantine agreement and blockchains. Byzantine Agreement (BA) is a form of *single-shot* consensus and has been studied in the distributed computing literature for more than 30 years [16]. Blockchains [12] or state machine replication protocols [25] are closely related to BA, but a blockchain protocol seeks to reach agreements repeatedly over time to establish an ever-growing, linearly-ordered-log. Although (under common assumptions) blockchains can be constructed from composition of (multi-valued) Byzantine Agreement, prior work has established the importance of studying blockchains as a separate abstraction due to various reasons: 1) while BA cares mostly about achieving consistency and liveness, additional practical security properties such as fairness or censorship resistence may be additionally stated for blockchains [6, 9, 12, 20–22]; and 2) direct constructions of blockchain protocols can sometimes be conceptually simpler and have better asymptotical or concrete efficiency than composition of single-shot BA [7, 10, 12, 14, 15, 24] — for this reason direct blockchain constructions are often more desirable in practical implementations too.

Pipelined-BFT. PiLi is inspired by a beautiful idea which we call "pipelined-BFT". This idea first implicitly appeared in the elegant work Casper-FFG [26] where the authors considered how to design a proof-of-stake finality gadget for Ethereum, and later was improved in works by others [3,8] — all prior works considered instantiation of this idea only for the partially synchronous setting and thus could only tolerate fewer than 1/3 corruptions.

At a high level, the "pipelined-BFT" idea is as follows. In classical BFT protocols, to confirm every batch of transactions, nodes must vote perform two or more rounds of voting. For example, in PBFT, the voting rounds are commonly referred to as "prepare" and "commit" (and for many synchronous protocols [2, 19], a common pattern occurs). A beautiful idea is the following: why don't we piggyback the present block's commit-round on the next block's notarization?

Although we are inspired by the pipelined-BFT paradigm, unlike all prior work, we aim to tolerate minority corruptions and meanwhile we would like to achieve a notion of robustness called "best-possible partition tolerance" which has been shown to be crucial for practical cryptocurrency systems [13]. We show that not only can we streamline the entire consensus protocol, we can also seamlessly embed any implicit view change that might be necessary in a unified "propose-vote" paradigm and thus the protocol does not have any special execution paths for recovery.

2 A Weakly Synchronous Execution Model

Earlier works [4, 13] have pointed out that the classical synchronous model is a mismatch for the real-world concerns when deploying distributed protocols. Specifically, if an honest node has even a short-term outage, the classical synchronous model immediately treats it as corrupt — at this point, a protocol proven secure in the classical synchronous model is no longer obligated to provide consistency for this node, even if it comes back online shortly afterwards and wishes to continue participating. This mismatch becomes even greater for long-running protocols like blockchains: no one can guarantee 100% up-time, and thus in the course of a few years, it could be that everyone has had an outage at some point.

To overcome this mismatch and yet not subject ourselves to the strong 1/3-resilience lower bounds pertaining to asynchrony and partial synchrony [11], Guo, Pass, and Shi [13] suggest a new model called "weak synchrony", which can be viewed as a relaxation of classical synchrony. The weak synchrony model allows us to tease out exactly which subset of classical, synchronous protocols enjoy the robustness properties that are important for practical deployment. A consensus protocol secure in a weakly synchronous network provides "best-possible partition tolerance" as long as the network has "sufficient, quantifiable" synchrony. More concretely, we want consistency for all honest nodes, even those who might have unstable network connections, as long as at any point of time, there are sufficiently many nodes that have good network connections — and this set of nodes may even change rapidly over time.

Although Guo et al. [13] demonstrate the theoretical existence of best-possible partition tolerant consensus protocols, their protocols are only of theoretical interest and are not recommended for practical implementation. Guo et al. [13] also aim to achieve both consistency and liveness under weak synchrony and as mentioned in Section 1.3, we observe that Guo et al. [13] liveness may be unnecessarily strong for practice. Specifically, in a weakly synchronous network, obviously offline nodes can get stuck for as long as they remain offline, and thus Guo et al. [13] require that honest nodes make progress when are online and as soon as they come back online. Note that such a notion of liveness is even stronger than standard partially synchronous protocols [7, 11]: standard partially synchronous protocols typically provide liveness only during "periods of synchrony", when super-majority number of honest nodes are online and have good network connectivity.

In this section, we first review the weakly synchronous model by Guo et al. [13], we then introduce a notion of global standardization time (GST) for weakly synchronous networks. Our GST notion is inspired by the modeling approach of classical partial synchrony [7, 11]: roughly speaking, we require progress only during a period of time in which a majority set of honest nodes are online and have good network connectivity for sufficiently long (where sufficiently long is defined relative to the protocol's confirmation delay).

2.1 Execution Model

We directly adopt the definitions of Guo et al. [13]. A protocol execution is formally modeled as a set of interactive Turing machines (ITMs). The execution proceeds in *rounds*, and is directed by a non-uniform probabilistic polynomial-time (p.p.t.) environment denoted $\mathcal{Z}(1^\kappa)$ parametrized by a security parameter $\kappa \in \mathbb{N}$. Henceforth we refer to ITMs participating in the protocol as *nodes* and we number the nodes from 1 to $n(\kappa)$ where n is chosen by $\mathcal{Z}$ and may be a polynomial function in κ.

Modeling corruption and network communication. We assume that there is a non-uniform p.p.t. adversary $\mathcal{A}(1^\kappa)$ that may communicate with $\mathcal{Z}$ freely at any time during the execution. The adversary $\mathcal{A}$ controls a subset of nodes that are said to be *corrupt*. All corrupt nodes are fully within the control of $\mathcal{A}$: $\mathcal{A}$ observes a node's internal state the moment it becomes corrupt and henceforth all the messages received by the corrupt node are forwarded to $\mathcal{A}$; further, $\mathcal{A}$ decides what messages corrupt nodes send in each round. In this paper, we assume that corruption is *static*, i.e., the adversary $\mathcal{A}$ decides which nodes to corrupt prior to the start of the protocol execution.

Nodes that are not corrupt are said to be *honest*, and honest nodes faithfully follow the prescribed protocol for as long as they remain honest. In each round, an honest node can either be *online* or *offline*.

Definition 2.1 (Honest and online nodes)**.** Throughout the paper, we shall use the notation $\mathcal{O}_r$ to denote the set of honest nodes that are online in round r. The set $\mathcal{O}_r$ is also called the "honest and online set" of round r. For $i \in \mathcal{O}_r$, we often say that i is honest and online in round r.

We make the following assumption about network communication. — note that our protocol is in the *multicast* model, i.e., every protocol message is sent to the set of all nodes:

Assumption 1 (Message delivery assumption)**.** *We assume that if someone in $\mathcal{O}_r$ multicasts a message* m *in round r, then everyone in $\mathcal{O}_t$ where $t \geq r + \Delta$ will have received* m *at the beginning of round t.*

In other words, an honest and online node is guaranteed to be able to deliver messages to the honest and online set of nodes Δ or more rounds later. The adversary $\mathcal{A}$ may delay or erase honest messages arbitrarily as long as Assumption 1 is respected.

Remark 2.2 (Offline nodes' network communication)**.** *Note that the above message delivery assumption implies that messages sent by honest but offline nodes can be arbitrarily delayed or even completely erased by the adversary. Further, the adversary can control which subset of honest messages each offline node receives in every round; it can omit an arbitrary subset of messages or even all of them from the view of honest offline nodes for as long as they remain offline.*

Remark 2.3. *We stress that a node is not aware whether it is online or offline. This makes protocol design in this model more challenging since the adversary can carefully choose a subset of messages for an offline (honest) node to receive, such that the offline node's view can appear perfectly "normal" such that it is unable to infer that it is offline. Jumping ahead, a consensus protocol secure in our model should guarantee that should an offline node make a decision while it is offline, such decisions would nonetheless be safe and would not risk inconsistency with the rest of the network.*

Our protocol needs to be aware of the parameters Δ and n. Throughout we shall assume that Δ and n are polynomial functions in κ. Formally, we can imagine that $\mathcal{Z}$ inputs Δ and n to all honest nodes at the start of the execution. Throughout the paper, we always shall assume that $(\mathcal{A}, \mathcal{Z})$ respects the following constraints:

> $\mathcal{Z}$ always provides the parameters n and Δ to honest nodes at the start of the execution such that n is the total number of nodes spawned in the execution, and moreover, the adversary $\mathcal{A}$ respects Assumption 1.

Schedule within a round. More precisely, in each round r, the following happens:

1. First, each honest node receives inputs from $\mathcal{Z}$ and receives incoming messages from the network; note that at this moment, $\mathcal{A}$'s decision on which set of incoming messages an honest node receives will have bearings on whether this honest node can be included in $\mathcal{O}_r$.
2. Each honest node then performs polynomially bounded computation and decides what messages to send to other nodes — these messages are immediately revealed to $\mathcal{A}$. Further, after the computation each honest node may optionally send outputs to $\mathcal{Z}$.
3. At this moment, $\mathcal{A}$ decides which nodes will belong to $\mathcal{O}_r$ where r denotes the current round. Note that $\mathcal{A}$ can decide the honest and online set $\mathcal{O}_r$ of the present round after seeing what messages honest nodes intend to send in this round.
4. $\mathcal{A}$ now decides what messages each corrupt node will send to each honest node. Note also that $\mathcal{A}$ is *rushing* since it can see all the honest messages before deciding the corrupt nodes' messages.
5. Honest nodes send messages over the network to other nodes (which may be delayed or erased by $\mathcal{A}$ as long as Assumption 1 is satisfied).

Definition 2.4 (χ-weak-synchrony)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects χ-weak-synchrony (or that $\mathcal{A}$ respects χ-weak-synchrony), iff in every round r, $|\mathcal{O}_r| \geq \lfloor \chi \cdot n \rfloor + 1$.

To aid understanding, we make a couple of remarks regarding this definition. First, the set of honest and online nodes need not be the same in every round. This allows us to model churns in the network: nodes go offline and come online; and we wish to achieve consistency for *all* honest nodes, regardless of whether they are online or offline, as long as sufficiently many nodes are online in each round. Second, the requirement of χ-weak-synchrony also imposes the fraction of corrupted nodes.

As an example, consider the special case when $\chi = 0.5$ and n is an even integer: if $(\mathcal{A}, \mathcal{Z})$ respects 0.5-weak-synchrony, it means that the adversary controls at most $n/2 - 1$ nodes. It could be that the adversary in fact controls fewer, say, $n/3$ number of nodes. In this case, up to $n/2 - 1 - n/3$ honest nodes may be offline in each round, and jumping ahead, in a consensus protocol we will require that consistency hold for these honest but offline nodes as well.

Finally, note also that our weakly-synchronous model is a generalization of the classical synchronous model: in the classical synchronous model, it is additionally required that for every r, $\mathcal{O}_r$ must be equal to the set of all nodes that remain honest till the end of round r (or later).

Notations for randomized executions. Throughout the paper, we use the notation $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ to denote the randomized experiment of running the protocol Π with $\mathcal{A}$ and $\mathcal{Z}$, invoked with the security parameter $\kappa \in \mathbb{N}$. The randomness in the experiment comes from honest nodes, $\mathcal{A}$, and $\mathcal{Z}$. Each sampling of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ produces an execution (also referred to as an execution trace) often denoted view. We would like that the fraction of executions that fail to satisfy relevant security properties be negligibly small in the security parameter κ. A function $\mathsf{negl}(\cdot)$ is said to be negligible if for every polynomial $p(\cdot)$, there exists some κ_0 such that $\mathsf{negl}(\kappa) \leq 1/p(\kappa)$ for every $\kappa \geq \kappa_0$.

2.2 Defining Global Standardization Time (GST)

We now formally define a notion of global standardization time (GST) to capture "periods of synchrony" during which the protocol should make progress. Later when we formally define consensus (see Section 2.3), we will require that liveness holds after the GST:

Definition 2.5 (Global Standardization Time)**.** Fix an execution denoted view, $\mathsf{GST}(\mathsf{view})$ is defined to be the first round after which all honest nodes are online, i.e., can communicate with each other within at most Δ delay.

At first sight, this might seem like an overly restrictive notion: in the real world, network conditions can alternate between good (i.e., a period of synchrony) and mildly bad (i.e., weak synchrony), and there may never be a round such that afterwards the network behaves perfectly forever. Nonetheless, the notion of GST is without loss of generality — it implies that whenever there is a period of synchrony that is sufficiently long (i.e., longer than the protocol's confirmation time), honest nodes are guaranteed to make progress during that good period. Note that a similar notion of GST is adopted in the classical partially synchronous model [11].

Remark 2.6 (An alternative definition of GST)**.** *An alternative and arguably better way to define GST is to require not that every honest node must be online afterwards, but rather, that there exist more than $n/2$ honest nodes (denoted $\mathcal{O}$) who must be persistently online afterwards. Under the latter definition, we can ask for liveness for those in $\mathcal{O}$ after the GST. Our liveness proofs later in fact directly hold for this alternate definition too, but for conceptual simplicity we adopt the same GST notion as classical partial synchrony [11] (but we achieve better resilience than partial synchrony).*

2.3 Weakly Synchronous Blockchain Protocols

In a blockchain protocol (also called state machine replication), a set of nodes seek to agree on an ever-growing log over time. In this section, we formally define the notion of a blockchain protocol for a weakly synchronous network. Roughly speaking, as long as the network conditions respect 0.5-weak-synchrony, we require *consistency*, i.e., all honest nodes' logs agree with each other although some nodes may progress faster than others. During periods of synchrony , we additionally require *liveness*, i.e., transactions observed by honest nodes get confirmed in all honest nodes' logs in a bounded amount of time. We formalize the definitions below.

Syntax. In a blockchain protocol, at the beginning of every round, a node receives as input a set of transactions txs from $\mathcal{Z}$, and outputs a LOG collected thus far to $\mathcal{Z}$ at the end of the round.

Security. Let $T_{\text{confirm}}(\kappa, n, \Delta)$ be a polynomial function in the stated parameters. We say that a blockchain protocol Π satisfies consistency (or T_{confirm}-liveness resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$, iff there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$, consistency (or T_{confirm}-liveness resp.) is satisfied:

- *Consistency*: A view satisfies consistency iff the following holds:
  - *Common prefix.* Suppose that in view, an honest node $i \in [n]$ outputs LOG to $\mathcal{Z}$ in round t, and an honest node $j \in [n]$ outputs LOG' to $\mathcal{Z}$ in round t' (i and j may be the same or different), it holds that either $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$. Here the relation $\preceq$ means "is a prefix of". By convention we assume that $\emptyset \preceq x$ and $x \preceq x$ for any x.
  - *Self-consistency.* Suppose that in view, an honest node i outputs LOG and LOG' in rounds t and t' respectively where $t \leq t'$, it holds that $\mathsf{LOG} \preceq \mathsf{LOG}'$.
- *$T_{confirm}$-liveness*: A view satisfies T_{confirm}-liveness iff the following holds: if an honest node has observed a transaction tx in or prior to some round $t \geq \mathsf{GST}(\mathsf{view})$, then, for any honest node i let LOG be its output log in some round $t' \geq t + T_{\text{confirm}}(\kappa, n, \Delta)$, it holds that $\mathsf{tx} \in \mathsf{LOG}$.

  Intuitively, liveness says that during periods of synchrony, every transaction observed by an honest node appears in every honest nodes' output logs within T_{confirm} time.

We say that Π is a *χ-weakly-synchronous blockchain protocol with $T_{confirm}$ confirmation time*, iff Π satisfies consistency and T_{confirm}-liveness w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony.

Remark 2.7. *Looking ahead, later in our protocol, the output log is actually structured as a sequence of blocks. Each block will be of the format $(e, \mathsf{TXs}, h_{-1})$ where e and h_{-1} are metadata relevant to the consensus protocol and TXs denotes the application-specific payload. Without loss of generality, we may assume that TXs is a set of transactions where each transaction tx is a string of fixed (polynomial) length. When LOG is a sequence of such blocks, $\mathsf{tx} \in \mathsf{LOG}$ means that tx is contained in the TXs field of some block in LOG.*

We additionally define a notion of progress called *chain growth* that is slightly weaker than the liveness notion above but sometimes easier to work with. Indeed, in the technical sections later, we will first prove chain growth (Sections 3 and 4). We then show a small modification (Section 5.2) that can lift the protocol to the liveness notion defined above.

Definition 2.8 (Chain growth)**.** Let $T(\kappa, n, \Delta)$ be a polynomial function in κ, n, and Δ. Formally, we say that a blockchain protocol satisfies T-growth w.r.t. $(\mathcal{A}, \mathcal{Z})$, iff for every $\kappa \in \mathbb{N}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: let $t_1 - T(\kappa, n, \Delta) \geq t_0 \geq \mathsf{GST}(\mathsf{view})$, then every honest node's output LOG in round t_1 is longer than its output LOG in round t_0.

Intuitively, chain growth requires that honest nodes' output logs grow at a steady pace during periods of synchrony, but it does not impose any requirements on the quality of the blocks (e.g., blocks produced should not selectively censor any transactions). To go from chain growth to liveness, we additionally need to ensure an appropriate notion of "block quality", i.e., the blocks confirmed do not censor any outstanding transaction.

3 The PiLi Family of Blockchain Protocols

3.1 Intuition

3.1.1 Warmup: Basic PiLi for Classical Synchrony

We present a brief warmup: an extremely simple blockchain protocol that satisfies consistency and liveness under honest majority, in a classical, fully-synchronous network. This warmup protocol is not really much simpler than our final construction (Section 3); it is nonetheless interesting to contrast our final construction with this warmup protocol to tease out how we achieve best-possible partition tolerance.

Warmup protocol PiLi. We assume the existence of a public-key infrastructure. Nodes sign every protocol message before sending it and moreover, any message is tagged with a purported sender. Upon receiving a message whose signature does not verify, the message is discarded immediately. For simplicity we also assume that a node always *echoes* (i.e., multicasts) every fresh message it has observed and thus if an honest node observes some information m in round r, all honest nodes will observe it in round $r + 1$ (if not earlier).

The protocol proceeds in epochs and every epoch contains two synchronous rounds called "propose" and "vote" respectively. For the time being, simply assume that in each epoch e, node $(e \mod n) + 1$ is the eligible proposer (we will discuss other proposer election policies later in Section 6.2). We use the convention that a smaller epoch e is *older* and a larger one is *fresher*.

- *Propose.* At the beginning of each epoch e, an eligible proposer proposes a block of the form $(e, \mathsf{TXs}, h_{-1})$ extending the freshest *notarized* chain it has seen so far, where TXs denotes the transactions to be confirmed, and h_{-1} denotes the parent chain's hash.

  As we shall see in the next step, nodes send votes on blocks; if a block receives at least votes from a majority of nodes, it is said to be *notarized*.

- *Vote.* Upon receiving a valid epoch-e proposal, a node i votes on the block $(e, \mathsf{TXs}, h_{-1})$ iff 1) i has seen a parent chain whose hash matches h_{-1} as well as the parent block's notarization; and 2) the parent block's epoch is not older than the freshest notarized chain i has seen at the beginning of the previous epoch $e - 1$.

If a node observes any notarized that ends with 6 blocks at consecutive epochs, chop off the trailing 5 blocks and the prefix is considered *final*.

Proof of security. In Section C, we give a more formal description of the above protocol, and prove that it satisfies consistency and liveness in a classical synchronous model assuming honest majority.

3.1.2 Making it Optimistically Responsive and Best-Possible Partition Tolerant

The above warmup protocol is not optimistically responsive since nodes must wait for epochs to timeout to make progress even when in reality, the network delivers messages much faster. The warmup protocol is also not best-possible partition tolerant: in particular, it could be that there exist two notarized blocks B and B' at the same epoch; but since offline nodes may be receiving an arbitrary subset of messages selected by the adversary, some offline nodes see B notarized and others see B' notarized; and thus they can reach different decisions each thinking there is no conflict.

Fortunately, a couple simple modifications can fix the above problems. First, for *optimistic responsiveness*, we allow a node to advance to the next epoch $e + 1$ immediately without waiting for the current epoch e to timeout, iff the node has observed not just majority, but 3/4 fraction of nodes' votes on an epoch-e block. Second, to achieve *best-possible partition tolerance*, we need that nodes rely on not just their own view in the protocol to decide "no-conflict"; instead, they should only believe in "no-conflict" if majority number of nodes confirm this belief. Thus our final protocol, referred to as PiLi*, works as follows:

- *Normal and skip blocks.* In a valid blockchain, we allow only two types of blocks, *normal* blocks and *skip* blocks. A normal block's epoch number must be the parent's epoch number plus 1, and a skip block's epoch number must be a multiple of 16 and moreover must be at least 16 epochs apart from the parent's epoch.
- *Propose.* At the beginning of every epoch e, an eligible proposer proposes a block denoted $(e, \mathsf{TXs}, h_{-1})$ extending the freshest notarized chain it has seen (if this does not violate the blockchain validity rule). Henceforth for simplicity we assume that a valid proposal $(e, \mathsf{TXs}, h_{-1})$ must carry the following information: 1) the parent $\mathtt{chain}$ whose hash must match h_{-1}; and 2) the parent's notarization $\mathcal{N}$ with the proposal.
- *Vote.* In every epoch e, a node votes on the first valid proposal from an eligible proposer as long as 1) the proposed block extends from a parent block whose epoch is not older than the freshest notarized chain the node has seen at the beginning of the previous epoch; and 2) either the proposed block is a skip block itself or else no conflicting notarizations have been seen since the last skip block in the parent chain (that the proposed block extends from).
- *Notarization and finalization.* If a block gains votes from majority voters, it becomes *notarized.* If a notarized chain ends at 13 number of consecutive epochs, chop off the trailing 8 blocks and the prefix is *final.*
- *Epoch advancement.* Normally, a node sends a time-out message denoted $\mathtt{clock}(e + 1)$ upon expiration of the present epoch e, and enters the next epoch upon receiving $\mathtt{clock}(e + 1)$ messages from majority nodes. However, if an epoch-e block gains votes from, not just the majority, but 3/4 fraction of the voters, an epoch-$(e + 1)$ block may be immediately proposed and voted on (and the corresponding proposer or voter enters epoch $e + 1$ immediately).

We stress that in the new protocol, nodes check conflicts during voting rather than at the time of finalization (like in the warmup protocol) and this is important for achieving best-possible partition tolerance. By voting on an epoch-e block, the node is simultaneously attesting to the fact that it has not seen any recent conflicts. Thus a notarization on a block vouches for the fact that many nodes have not seen a recent conflict.

However, with this modification, we need to additionally defend against a potential DoS attack: if corrupt nodes start to double vote, they can cause honest nodes to stop voting due to seeing conflicting notarizations. We thus patch this problem by introducing *skip* blocks: this allows honest nodes to "complain" by refusing to vote; and even though chain growth will temporarily halt at this point, progress will ensue at the next opportunity when an honest node proposes a skip block (during a period of synchrony).

3.2 Formal Description of PiLi*

3.2.1 Additional Preliminaries

Echo mechanism and stronger network delivery assumption. For convenience, we will make a slightly stronger assumption on the network — but in fact this stronger assumption can be realized from Assumption 1 described earlier.

Assumption 2 (Strong message delivery assumption)**.** *If $i \in \mathcal{O}_r$ and i has multicast or received a message m before the end of round r, then everyone in $\mathcal{O}_t$ where $t \geq r + \Delta$ will have received m at the beginning of round t.*

A brute-force way to realize Assumption 2 from Assumption 1 is to have every node echo (i.e., multicast) all messages they have seen so far in every round. Later in Section B, we will describe a more efficient echo mechanism that realizes Assumption 2 from Assumption 1 — roughly speaking, nodes continue to echo each message until they have heard majority nodes echo it [13]. We note that if we adopt a peer-to-peer diffusion mechanism like the network layer of Bitcoin or Ethereum, then the network layer should already handle the necessary retries for the stronger Assumption 2 to hold.

Setup assumptions. We assume that a hash function H is randomly chosen from a collision-resistant family a-priori. Further, depending on the proposer-eligibility policy, we may also choose a random oracle H^* a-priori for electing random proposers. We also assume the existence of a PKI and denote each node i's public/secret key pair as $(\mathsf{pk}_i, \mathsf{sk}_i)$. The hash function, the random oracle, and the PKI are chosen after the adversary submits the choice of who to corrput.

3.2.2 Protocol

Blocks and blockchains are defined in the most natural manner.

Block. Each block is of the format $(e, \mathsf{TXs}, h_{-1})$ where $e \in \mathbb{N}$ denotes an epoch number, $\mathsf{TXs} \in \{0,1\}^*$ denotes the block's payload (e.g., a batch of transactions to confirm), and $h_{-1} \in \{0,1\}^\kappa$ denotes the parent hash, i.e., hash of the blockchain the block extends from.

Valid blockchain. A valid blockchain denoted chain is a sequence of blocks where for $1 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i]$ denotes the i-th block in chain. We often use the following useful blockchain notation:

- $\mathsf{chain}[-1]$ denotes the last block in chain; $\mathsf{chain}[: i]$ denotes the first i blocks of chain; and $\mathsf{chain}[i :]$ denotes the suffix of chain starting at the i-th block;
- assume that a block of epoch e exists in chain, we use $\mathsf{chain}\langle e\rangle$ to denote this block at epoch e in chain; and use $\mathsf{chain}\langle: e\rangle$ to denote the prefix of chain ending at the block $\mathsf{chain}\langle e\rangle$.

For a blockchain chain to be valid, the following must be respected:

1. *Strictly increasing epochs.* For all $1 \leq i < j \leq |\mathsf{chain}|$, $\mathsf{chain}[i].e < \mathsf{chain}[j].e$; and

2. *Valid parent hash.* It must be that $\mathsf{chain}[1].h_{-1} = \bot$, and for every $2 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i].h_{-1} = H(\mathsf{chain}[: i-1])$ where H is randomly sampled from a collision-resistant hash family[1].

3. For every $2 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i]$ must either be a *normal* block, or a *skip* block, i.e., it must satisfy one of the following two rules:

   - either $\mathsf{chain}[i].e = \mathsf{chain}[i-1].e + 1$ — in this case we say that $\mathsf{chain}[i]$ is a *normal* block; or
   - $\mathsf{chain}[i].e \geq \mathsf{chain}[i-1].e + 16$ and $\mathsf{chain}[i].e$ is a multiple of 16 — in this case we say that $\mathsf{chain}[i]$ is a *skip* block.

Without loss of generality, we may assume that every valid blockchain denoted chain is prefaced by an imaginary genesis block denoted $\mathsf{chain}[0] := (e = 0, \bot, \bot)$.

Vote and notarization. A pair (h, σ) from a purported voter $i \in [n]$ is said to be a valid vote for some chain, iff $h = H(\mathsf{chain})$ and moreover $\Sigma.\mathsf{Verify}_{\mathsf{pk}_i}(h, \sigma) = 1$.

A collection of at least $\frac{3n}{4}$ valid votes for chain from distinct voters (i.e., nodes) is said to be a *strong notarization* for chain. A collection of stricly more than $\frac{n}{2}$ valid votes for chain from distinct voters is said to be a *notarization* for chain. Clearly, a strong notarization is also a notarization[2] for chain from distinct voters is said to be a *notarization* for chain.

Remark 3.1. *Note that since our blockchain validity definition requires that each block specifies the parent hash, assuming no hash collision, a block may be considered an alias for a chain. Thus, we often use the term "a vote or (strong) notarization for* chain*" and the term "a vote or (strong) notarization for the block* $\mathsf{B} := \mathsf{chain}[-1]$*" interchangeably.*

"Fresher than" relation. We say that chain is an epoch-e chain iff $\mathsf{chain}[-1].e = e$. A chain is said to be *fresher* than another chain' iff $\mathsf{chain}[-1].e > \mathsf{chain}'[-1].e$.

Stability-favoring proposer eligibility. For concreteness, we will first describe a stability-favoring proposer election policy, i.e., we only switch proposer if the current one stops working. Such a policy might be somewhat more suited for a permissioned deployment environment. Later in Section 6.2, we describe other policies, e.g., a democracy-favoring policy that seems suitable for a *decentralized*, proof-of-stake setting.

Consinder the following stability-favoring proposer eligibility policy: we say that a node $i \in [n]$ is an eligible proposer for chain (or equivalently, the last block in chain) iff the following holds: let $\mathsf{chain}[\ell]$ be the last *skip* block in chain, it must be that $i = (H^*(\mathsf{chain}[\ell].e) \mod n) + 1$ where H^* is a hash function (modeled as a random oracle) used for proposer election.

Protocol. The formal description of our protocol is in Figure 1. For conceptual simplicity, in Figure 1 we require that a proposer always include the parent chain and its notarization in any proposal it makes. Equivalently, the proposer need not explicitly attach the parent chain and its notarization, and a node could request/receive these from anyone — a proposal is regarded as complete only when a node has seen the parent chain and its notarization.

[1]In practice, H should be incrementally evaluated using a base hash function H_0: let $H(\mathsf{chain}) := H_0(H(\mathsf{chain}[:-1])||\mathsf{chain}[-1])$ if $|\mathsf{chain}| > 1$; and let $H(\mathsf{chain}) := H_0(\mathsf{chain}[1])$ if $|\mathsf{chain}| = 1$.

[2]In practice, we can employ an aggregate signature to compress the signatures.

PiLi*

Epoch advancement. If a node has been in an epoch e for 5Δ rounds, multicast a signed $\texttt{clock}(e+1)$ message. A node currently in epoch e advances to epoch $e+1$ as soon as either of the following conditions occurs:

1. *fast-forward:*
   - either the node is ready to propose epoch-$(e+1)$ block extending from a strongly notarized epoch-e block (i.e., it has seen a strong notarization for an epoch-e chain and the node is an eligible proposer for proposing an epoch-$(e+1)$ block extending from chain), or
   - it has seen a valid epoch-$(e+1)$ proposal extending from a strongly notarized epoch-e block.
2. *timeout:* the node has observed strictly more than $n/2$ valid $\texttt{clock}(e+1)$ messages from distinct nodes.

Henceforth we may assume that an honest node always invokes the above epoch advancement procedure at the beginning of a round immediately after receiving incoming messages and before taking other actions. Note that at the beginning of a round, a node may advance multiple epochs.

Propose. Upon entering epoch e, let TXs be the outstanding pool of unconfirmed transactions; let chain be the node's current freshest notarized chain (pick a strongly-notarized chain for tie-breaking); and let $\mathsf{B} := (e, \mathsf{TXs}, H(\mathsf{chain}))$. If $\mathsf{chain}||\mathsf{B}$ forms a valid chain and moreover the node is eligible for proposing $\mathsf{chain}||\mathsf{B}$, multicast the proposal (B, σ) where σ is the node's signature on B, tagged with $(\mathsf{chain}, \mathcal{N})$ where $\mathcal{N}$ is the set of all votes the proposer has seen for the parent chain. Henceforth, a proposal (B, σ) tagged with $(\mathsf{chain}, \mathcal{N})$ is said to be valid if it is signed by an eligible proposer, chain matches the parent hash in B, $\mathcal{N}$ is a correct notarization for chain, and finally $\mathsf{chain}||\mathsf{B}$ satisfies blockchain validity rules.

Vote. If the node fast-forwarded to the current epoch e, vote on the proposal that triggered the fast forward. Else, perform the following: let $\mathsf{B} := ((e, \mathsf{TXs}, h_{-1}), \sigma)$ tagged with $(\mathsf{chain}, \mathcal{N})$ be the *first* valid epoch-e proposal that was observed, vote on the proposal if the following conditions are satisfied:

1. *parent not too stale:* unless the current epoch $e = 1$, the aforementioned parent chain must be at least as fresh as the freshest notarized chain in the node's view at the beginning of the *previous* epoch;
2. *no recent conflicts:* if the proposed block $(e, \mathsf{TXs}, h_{-1})$ is not a skip block, then the following must hold: let $\mathsf{chain}[\ell]$ be the last skip block (or the imaginary genesis block) in the aforementioned parent chain, then for every block in $\mathsf{chain}[\ell :]$, the node must not have observed a notarization for a conflicting block with the same epoch number.

To vote on $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$, let chain be the parent chain matching h_{-1}: sign $h := H(\mathsf{chain}||\mathsf{B})$ to obtain the signature σ, and multicast (h, σ).

Finalize. At any time, a node outputs the freshest chain that is considered final where finality is defined below: if a node observes a notarized chain that ends at 13 consecutive epochs, chop off the trailing 8 blocks and the prefix is considered final.

Figure 1: The PiLi* protocol.

In Section 4, we shall prove that the PiLi* protocol (Figure 1) satisfies consistency under 0.5-weak-synchrony and a weaker notion of liveness called *chain growth* during periods of synchrony. Then, in Section 5.2, we propose simple transformation that allows us to achieve the stronger notion of liveness defined in Section 5.2. Specifically, as mentioned earlier in Section 2.3, chain growth guarantees that during periods of synchrony, honest nodes' output logs grow at a steady pace, but does not guarantee that the blocks confirmed have good quality (e.g., do not selectively censor transactions). To achieve the stronger liveness notion defined in Section 5.2, in Section 5.2 we need a simple technique for ensuring block quality: basically, honest nodes will simply refuse to vote if they believe that the current proposer is proposing bad blocks (e.g., censoring certain transactions); this will effectively choke the current proposer and the proposer re-election mechanism will then be triggered to re-elect the proposer.

Theorem 3.2 (PiLi* blockchain)**.** *Suppose that the digital signature scheme employed is secure and that the hash family satisfies collision resistance. Let λ be any super-logarithmic function in κ. The* PiLi* *protocol in Figure 1 satisfies consistency and $\lambda\Delta$-growth w.r.t. any non-uniform $(\mathcal{A}, \mathcal{Z})$ that respects* 0.5*-weak-synchrony.*

Our PiLi* protocol also satisfies *optimistic responsiveness*: roughly speaking as long as $3/4$ fraction of nodes are honest, then transaction confirmation time depends only on the actual network delay δ (but not on the a-priori upper-bound Δ) as soon as an honest proposer takes over after the GST. We will formally state this property and prove it in Section 4.

4 Proofs

In all of our theorem and lemma statements below, we by default assume that $(\mathcal{A}, \mathcal{Z})$ is non-uniform p.p.t. and moreover respects 0.5-weak-synchrony.

4.1 Additional Definitions and Useful Facts

Good executions. Henceforth we ignore the negligible fraction of bad executions where honest nodes' signatures are forged or where a hash collision is found. Most of our theorems and lemmas below hold only for "good executions" where honest nodes' signatures are not forged and hash collisions do not exist in the union of honest nodes' views.

Additional terminology. We introduce some additional useful terminology:

- We say that a message m is in honest view in some execution if some honest node observes m in some round during the execution. We say that a notarized chain chain is in honest view in some execution if there exists some round in which some honest node observes not only chain but also a notarization for chain.
- We say that in some execution, a message m is in "honest and online view" in some round r iff everyone in $\mathcal{O}_t$ where $t \geq r$ must have observed the message m by the end of round t.
- $\mathsf{chain} \preceq \mathsf{chain}'$ means chain is a prefix of chain'; by convention, $\mathsf{chain} \preceq \mathsf{chain}$.
- When the execution trace we are referring to is clear from the context, we shall use the notation $R^{(<e)}$ or equivalently $R^{(\leq e-1)}$ to denote the last round in which every honest and online node is in an epoch strictly smaller than e; and we use the notation $R^{(>e)}$ or equivalently $R^{(\geq e+1)}$ to denote the first round in which every honest and online node is in an epoch greater than

e. Here we measure the which epoch the node is in some round after the node processes all the incoming possibly clock messages at the beginning of the round — recall that we assume that a node always processes incoming clock messages first before taking any other action in a round.

- Henceforth, for simplicity, a collection of signed $\texttt{clock}(e)$ messages from more than $n/2$ distinct nodes is also said to be *notarized* $\texttt{clock}(e)$ message.

Fact 4.1. *If in some round r, an honest and online node is in epoch e, then in round $r + \Delta$, all nodes honest and online must have entered epoch e or greater. A direct corollary is the following: for any e, it must be that $R^{(\geq e)} - R^{(<e)} \leq \Delta + 1$.*

Proof. Follows directly from Assumption 2. □

Fact 4.2. *Consider any good execution, it must be that for any e, in round $R^{(<e)}$, no honest (online or offline) node is in epoch $e + 1$ or greater.*

Proof. For an honest node to be in epoch $e + 1$ or greater, it must have heard either an epoch e'-notarization for $e' \geq e$, or a notarized $\texttt{clock}(e'')$ message for $e'' \geq e + 1$. Note that the honest and online nodes in round $R^{(<e)}$ cannot have sent an epoch-e' vote or $\texttt{clock}(e'')$ message. Thus such an epoch e'-notarization or a $\texttt{clock}(e'')$ message cannot have gained notarization in honest view. □

Fact 4.3. *Consider some good execution: if* chain *is a notarized chain in honest view at the beginning of round r, then every prefix of* chain *must be a notarized chain in honest view too at the beginning of round r.*

Proof. For a block to obtain notarization in honest view, some honest node must have voted for it, and this node will only vote for it if it has seen the parent block's notarization. The proof then follows by inductively applying this argument to every block in the prefix. □

Fact 4.4. *In a good execution, only chains that satisfy our blockchain validity rules can gain notarization in honest view.*

Proof. Straightforward by observing that no honest node will vote for a chain that breaks blockchain validity rules. □

4.2 Consistency

Lemma 4.5 (Uniqueness of strong notarization)**.** *Consider some good execution and let* chain *and* chain' *be two strongly notarized chains ending at epoch e in honest view . Then it must be* $\mathsf{chain} = \mathsf{chain}'$.

Proof. By honest protocol definition, each honest node will only vote for at most one epoch-e block; and each corrupt node can vote for both chain and chain'. Let $f < n/2$ denote the number of corrupt nodes; thus the total number of distinct votes for chain and chain' is less than $n - f + 2f = n + f < 3n/2$ (note that if the same node signs two signatures for the same block it is counted only once). Suppose for contradiction that $\mathsf{chain} \neq \mathsf{chain}'$ but both blocks gained a strong notarization (i.e., at least $3n/4$ votes from distinct nodes) in honest view. This means that the number of distinct votes for chain and chain' is at least $3n/2$ and thus we have reached a contradiction. □

Lemma 4.6. *Consider a good execution: suppose that* chain *is a notarized chain in honest view containing three blocks at epochs* e*,* $e+1$*, and* $e+2$ *respectively; then a notarization for* $\mathsf{chain}\langle e\rangle$ *must be in honest and online view by the beginning of round* $R^{(>e+2)}+\Delta$*.*

Proof. By Fact 4.3 every block in chain must have notarization in honest view. We consider only good executions in our argument below. Let r be the first round in which a notarization for $\mathsf{chain}\langle e+1\rangle$ appeared in honest view. By the end of round r, some honest node who has voted for $\mathsf{chain}\langle e+1\rangle$ must have been online in or after the round in which it voted for $\mathsf{chain}\langle e+1\rangle$ — since if every honest node that voted for $\mathsf{chain}\langle e+1\rangle$ voted for the block when it is offline and moreover has remained offline since, there cannot be a notarization for $\mathsf{chain}\langle e+1\rangle$ in honest view. Recall that an honest node only votes for a block after observing a valid notarization for the parent block, therefore a notarization for $\mathsf{chain}\langle e\rangle$ must be in honest view in round $r+\Delta$ by our strong message delivery assumption (Assumption 2).

It suffices to prove that $r \le R^{(>e+2)}$. Suppose not, we now reach a contradiction below. Recall that in any round $t \ge R^{(>e+2)}$, everyone in $\mathcal{O}_t$ must be in epoch $e+3$ or greater. Thus no node in $\mathcal{O}_{R^{(>e+2)}}$ will vote for any block at epoch $e+2$ or smaller in or after round $R^{(>e+2)}$. Note also that $\mathsf{chain}\langle e+2\rangle$ cannot have gained a notarization in honest view before $\mathsf{chain}\langle e+1\rangle$ gained a notarization in honest view. Thus if $r > R^{(>e+2)}$ we reach the conclusion that $\mathsf{chain}\langle e+2\rangle$ cannot gain notarization in honest view which contradicts our assumption. □

Lemma 4.7 (Non-skipping condition)**.** *Consider a good execution: suppose that* chain *is a notarized chain in honest view containing three blocks at epochs* e*,* $e+1$*, and* $e+2$ *respectively, suppose also that* chain' *is also a notarized chain in honest view ending at an epoch* $e' \ge e$*. Then,* chain' *cannot skip all of the epochs* $e, e+1, e+2, e+3$*,* $e+4$*, and* $e+5$*.*

Proof. It suffices to prove that if chain' ends at an epoch $e' \ge e+6$, chain' cannot skip all of e to $e+5$. Suppose for the sake of contradiction that indeed chain' skips all epochs from e to $e+5$. Let e_1 be the smallest epoch contained in chain' that is $e+6$ or greater. For $\mathsf{chain}'\langle e_1\rangle$ to ever gain a notarization in honest view, someone $i^* \in \mathcal{O}_{R^{(<e+5)}}$ must vote for $\mathsf{chain}'\langle e_1\rangle$ after round $R^{(<e+5)}$.

By Lemma 4.6, it suffices to prove that $R^{(<e+5)} \ge R^{(>e+2)}+\Delta$ since in this case when i^* enters epoch $e+5$ it must have observed a notarization for $\mathsf{chain}\langle e\rangle$ and therefore i^* will not vote for $\mathsf{chain}'\langle e_1\rangle$ and this leads to a contradiction. Below we focus on proving the following claim — this claim, combined with the following simple fact would complete the proof: some honest node in $\mathcal{O}_{R^{(\ge e+5)}}$ must vote for $\mathsf{chain}'\langle e_1\rangle$ at some point, and this honest node cannot have entered epoch $e+5$ through fast-forwarding.

Claim 4.8. *Consider a good execution: suppose that some honest node in* $\mathcal{O}_{R^{(\ge e+5)}}$ *did not enter epoch* $e+5$ *through fast-forwarding, then it must hold that* $R^{(<e+5)} \ge R^{(>e+2)}+\Delta$*.*

Proof. On one hand, due to Fact 4.1, we have that

$$R^{(>e+2)}+\Delta = R^{(\ge e+3)}+\Delta \le \Delta+1+R^{(<e+3)}+\Delta = 2\Delta+1+R^{(<e+3)}$$

On the other hand, due to Fact 4.2, we have that in round $R^{(<e+3)}$, no honest node is in epoch $e+4$ or greater. Moreover, since some honest node in $\mathcal{O}_{R^{(\ge e+5)}}$ did not enter epoch $e+5$ through fast-forwarding, at least 5Δ rounds must have elapsed between round $R^{(<e+3)}$ and $R^{(\ge e+5)}$. We now have that

$$R^{(>e+2)}+\Delta \le 2\Delta+1+R^{(<e+3)} \le 2\Delta+1+R^{(\ge e+5)}-5\Delta \le -3\Delta+1+R^{(<e+5)}+\Delta+1 \le R^{(<e+5)}$$

where the last but second inequality is again by Fact 4.1. □

□

Theorem 4.9 (Consistency). *Consider a good execution: suppose that* chain *is a notarized chain in honest view ending at* 13 *consecutive epochs denoted* $e-5, e-4, e-3, \ldots, e+7$*, and suppose that* chain' *is also a notarized chain in honest view ending at an epoch* $e' \geq e+7$*. Then, it must be that* $\mathsf{chain}\langle e \rangle \preceq \mathsf{chain}'$*.*

Proof. Due to Lemma 4.7, chain' must contain one block denoted $\mathsf{chain}'\langle e_0 \rangle$ whose epoch $e_0 \in \{e-5, e-4, \ldots, e\}$ and one block denoted $\mathsf{chain}'\langle e_1 \rangle$ whose epoch $e_1 \in \{e+5, e+6, \ldots, e+10\}$.

By blockchain validity rule, in chain', there must be consecutive normal blocks at every epoch number $e_0+1, e_0+2, \ldots, e_1$. In particular, this means that epochs $e, e+1, \ldots, e+5$ must belong to chain'.

For the sake of contradiction, suppose that $\mathsf{chain}\langle e \rangle$ is not a prefix of chain'. This means that all the blocks in chain' whose epochs are between $e, e+1, \ldots, e+5$ are not part of chain. By Lemma 4.5, for each of $\mathfrak{e} \in \{e, e+1, \ldots, e+5\}$, it must be that that either $\mathsf{chain}\langle \mathfrak{e} \rangle$ or $\mathsf{chain}'\langle \mathfrak{e} \rangle$ does not have a strong notarization in honest view.

By Lemma 4.6, a notarization for $\mathsf{chain}\langle e \rangle$ and a notarization for $\mathsf{chain}'\langle e \rangle$ must be in honest view by the beginning of round $R^{(>e+2)} + \Delta$. Since either $\mathsf{chain}\langle e+4 \rangle$ or $\mathsf{chain}'\langle e+4 \rangle$ does not have a strong notarization in honest view, without loss of generality, we assume that $\mathsf{chain}\langle e+4 \rangle$ does not have a strong notarization in honest view (otherwise a symmetric argument could be made). For $\mathsf{chain}\langle e+5 \rangle$ to gain a notarization in honest view, it must be that some node i^* in $\mathcal{O}_{R^{(<e+5)}}$ must vote for $\mathsf{chain}\langle e+5 \rangle$ after round $R^{(<e+5)}$.

Now by Claim 4.8, we can prove that $R^{(<e+5)} \geq R^{(>e+2)} + \Delta$, since here we also have that some honest node in $\mathcal{O}_{R^{(\geq e+5)}}$ must vote for $\mathsf{chain}\langle e{+}5 \rangle$ (whose parent does not have a strong notarization) at some point, and this honest node cannot have entered epoch $e+5$ through fast-forwarding.

Therefore, we may conclude that i^* cannot have voted on $\mathsf{chain}\langle e{+}5 \rangle$ since it must have observed conflicting notarizations for epoch e in round $R^{(<e+5)}$. Thus we have reached a contradiction. □

Corollary 4.10. *The* PiLi* *protocol (Figure 1) satisfies consistency w.r.t. any non-uniform p.p.t.* $(\mathcal{A}, \mathcal{Z})$ *that respects* 0.5*-weak-synchrony.*

Proof. Straightforward from the finalization rule and Theorem C.6. □

4.3 Chain Growth

Fact 4.11 (Epoch advancement does not stop). *Let* t_0 *be at least* 3Δ *rounds after GST in a good execution, and let* e_0 *be the largest epoch that any honest node is in during round* t_0*. Let* $e \geq e_0$*, we have that* $R^{(\geq e)} \leq t_0 + \Delta + 6\Delta(e - e_0)$*.*

Proof. By the beginning of round $t_0 + \Delta$, every honest node will be in epoch e_0 or greater; then by the beginning of round $t_0 + \Delta + 5\Delta = t_0 + 6\Delta$ every honest node will either be in round $e_0 + 1$ or greater or will have sent $\texttt{clock}(e_0+1)$; thus by the beginning of round $t_0 + 6\Delta + \Delta$, every honest node will be in round $e_0 + 1$ or greater; and the rest of the proof can be completed inductively in this manner. □

Lemma 4.12. *Let* t_0 *be at least* 3Δ *rounds after the GST in a good execution, and suppose that* $R^{(<16j-1)} \geq t_0$*. Moreover, suppose that there is no epoch-*$(16j-1)$ *notarized block ever in honest view, and that an honest node proposed an epoch-*$16j$ *block. Then, it must be that in round* $R^{(\leq 16j+13)} + 2\Delta$*, some honest node will have output a final chain that includes some epoch that is* $16j$ *or greater.*

Proof. We first prove the following simple fact:

Fact 4.13. *Let t_0 be at least 3Δ rounds after the GST in a good execution, and suppose that $R^{(<16j-1)} \geq t_0$. For any $e \geq 16j-1$, if there is no epoch-e notarization in honest view, it must be that $R^{(\leq e)} - R^{(\geq e)} \geq 2\Delta$.*

Proof. Recall that $R^{(\geq e)}$ is the first round in which all honest nodes enter epoch e or greater, therefore before round $R^{(\geq e)} - \Delta$, no honest node must have entered epoch e or greater yet. In round $R^{(\leq e)}+1$, some honest node would have entered epoch $e+1$ or greater. However, since there is no epoch-e notarization in honest view, no honest node can enter epoch $e+1$ until 5Δ rounds after the first honest node enters epoch e. □

If there is no epoch-$(16j-1)$ notarized block ever in honest view, by our blockchain validity rule, only a skip block can be proposed for epoch $16j$. Thus there can only be a single honest node (denoted i^*) and no corrupt node eligible for proposing an epoch-$16j$ block. Node i^* must propose this epoch-$16j$ block in the round it first enters epoch $16j$ — and let t^* be this round. We will now argue that all honest nodes will vote on this proposal when the receive it:

- *Parent not too stale.* First, due to Fact 4.13, $t^* \geq R^{(\leq 16j-1)} \geq R^{(\geq 16j-1)} + 2\Delta$. Thus if an honest node sees a message at the beginning of epoch $16j-1$, the honest proposer i^* must have seen it in round t^*. Thus the proposal will not be rejected due to the parent being too stale.

- *No recent conflicts.* This trivially holds because the proposed block is a skip block itself.

Observe also that at the beginning of round $t^* + \Delta$, all honest nodes will have seen the proposal as well as the parent chain and its notarization; further, all honest nodes will have entered epoch e (or greater). Note also that no node can have sent any $\texttt{clock}(e')$ message for any $e' \geq 16j+1$ by round $t^* + 2\Delta$ — this is because that in round t^*, no honest node must have been in epoch $16j$ for more than 3Δ rounds. Thus, all honest nodes will have cast a vote for the epoch-$(16j)$ proposal by the end of the round $t^* + \Delta$. Moreover, at the beginning of the round $t^* + 2\Delta$, all honest nodes would have received a notarization for the epoch-$(16j)$ proposal made by i^*.

Now, when node i^* enters epoch $e = 16j+1$, say, in round t_1^*:

- either i^* entered this epoch by fast-forwarding in which case it proposes an epoch-e block extending from a strongly notarized epoch-$(e-1)$ parent in round t_1^* or

- it entered epoch e by collecting a notarized $\texttt{clock}(e)$ message — in this case it is not hard to see that $t_1^* \geq t^* + 2\Delta$ and thus i^* will have seen an epoch-$(e-1)$ notarized block in round t_1^*; in this case i^* will propose an epoch-e block extending from an epoch-$(e-1)$ parent in round t_1^*.

Repeating the same argument as above, and additionally observing that there cannot be any conflicts since the last skip block which is the block at epoch $16j$ since there is a unique honest proposer, we may conclude that in round $t_1^* + 2\Delta$, every honest node must have observed a notarization for the proposed block at epoch e.

At this moment, we can inductively show that for every $k > 0$, for epoch $e = 16j + k$, let t_k^* be the round in which node i^* first enters epoch e. It must be that 1) node i^* will propose an epoch-e block extending from an epoch-$(e-1)$ parent in round t_k^*, and 2) every honest node has observed a notarization for the proposed block at epoch e in round $t_k^* + 2\Delta$.

Thus by the round $R^{(\leq 16j+13)} + 2\Delta$, there is a notarized chain in honest view containing every epoch $e \in \{16j, \ldots, 16j+13\}$, this is sufficient to conclude our proof due to the protocol's finalization rule. □

Theorem 4.14 (Chain growth during periods of synchrony)**.** *Let $\lambda = \omega(\log \kappa)$ be a super-logarithmic function in the security parameter κ. Except for a $\mathsf{negl}(\kappa)$ fraction of the executions, the following must hold: suppose that t_0 is at least 3Δ round after the GST and $t_1 - t_0 \geq \lambda\Delta$. Then, by the end of round t_1, some honest node must have output a new block B which did not gain notarization in honest view yet by the end of round t_0.*

Proof. Let e_0 be the largest epoch that any honest node is in during round t_0. It is also not hard to see that in round t_0 there is no epoch-e notarization in honest view for any $e > e_0$. Let $16j$ be the first epoch that is greater than $e_0 + 16$. First, suppose that there is a notarization for an epoch-$(16j - 1)$ block in honest view by the end of round t_1, then due to our blockchain validity rules and the finalization rule, the conclusion stated in the theorem holds as long as this is a good execution. Below we focus on the case where there is no notarization for an epoch-$(16j - 1)$ block in honest view by the end of round t_1. In this case, we claim that if the proposer denoted i_0 for proposing an epoch-$(16j)$ skip block is honest and the proposer denoted i_1 for proposing an epoch-$(16(j+1))$ skip block is also honest, then one of i_0 and i_1 must successfully propose a block of epoch $16j$ or $16(j+1)$ respectively. To see this, suppose that the proposer i_0 did not propose a block of epoch $16j$. Since there is no notarization for epoch-$(16j-1)$ in honest view, there cannot be a valid normal block proposed at epoch $16j$ either. Thus no honest node can observe any valid proposal at epoch $16j$ and no block of epoch $16j$ can gain notarization in honest view. This implies that no block of epoch $16j+1, \ldots, 16(j+1)-1$ can gain notarization in honest view. Now when i_1 first enters epoch $16(j+1)$, it will obviously succeed in proposing a skip block at epoch $16(j+1)$. Thus if the proposer for proposing an epoch-$(16j)$ skip block and the proposer for proposing an epoch-$(16(j+1))$ skip block are both honest, by Lemma 4.12, it must be that either the execution is bad or by round $R^{(\leq 16(j+1)+13)}$, some honest node must have output a new block that had not gained notarization yet by the end of round t_0.

Repeating the above argument, we can prove that for every $j' \geq j$ such that $R^{(\leq 16(j'+1)+13)} \leq t_1$, if the proposer for proposing an epoch-$(16j')$ skip block and the proposer for proposing an epoch-$(16(j'+1))$ skip block are both honest, then either the execution is bad or some honest node must have output a new block that had not gained notarization yet by the end of round t_0.

The proof now concludes by observing that due to Fact 4.11 and the fact that $t_1 - t_0 \geq \omega(\log\kappa)\Delta$, such a j' must exist except with negligible in κ probability. □

Corollary 4.15 (Chain growth during periods of synchrony)**.** *Let $\lambda = \omega(\log\kappa)$ be any super-logarithmic function in κ. Our* PiLi⋆ *protocol satisfies $\lambda\Delta$-growth w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects 0.5-weak-synchrony.*

Proof. Straightforward from Theorem 4.14 and the fact that if some honest node observes B (see the statement of Theorem 4.14) after GST, then all honest nodes will see it in another Δ rounds. □

4.4 Optimistic Responsiveness

We say that an honest proposer i *takes over* in round t in some execution iff i proposes a skip block in round t. Our PiLi⋆ protocol achieves optimistic responsiveness as soon as 1) an honest proposer takes over after the GST; and 2) at least $3n/4$ of the nodes are honest.

More specifically, we can show the following:

Theorem 4.16 (Optimistic responsiveness under good conditions)**.** *Suppose that in some good execution, an honest proposer i takes over in some round t that is at least 3Δ rounds after the GST; and moreover, at least $3n/4$ number of nodes are honest. Let $\delta \leq \Delta$ be the actual maximum* *honest message delay after the GST in this execution. Then, suppose that i proposes an epoch-e block in round $t' \geq t$, we have the following:*

- *the proposed block obtains notarization in every honest node's view by the beginning of round $t' + 2\delta$;*
- *by the beginning of round $t' + 2\delta$, every honest node will have fast-forwarded to epoch $e + 1$ or greater;*
- *i will successfully propose a block at epoch $e + 1$ extending from a strongly notarized parent of epoch e.*

Proof. Deferred to the Supplemental Materials (Section A). □

5 Extensions: Reconfigurability and Block Quality

5.1 Reconfigurable PiLi*

We consider how to live-reconfigure of the set of consensus nodes and propose a reconfigurable version of PiLi*. During the operational life-cycle of the consensus protocol, reconfigurability is typically needed due to maintenance reasons as well as key updates. Reconfigurability is also important in a decentralized, proof-of-stake setting where the consensus committee must be reelected periodically as stake switches hands. Later in Section 6.2, we will discuss more how to use our "reconfigurable PiLi*" in such a decentralized proof-of-stake environment.

It is quite easy to support reconfiguration in our PiLi* protocol. Imagine that we use the blockchain itself to encode the decision to reconfigure. In general, let $\mathsf{Comm}(\mathsf{chain})$ be a function that outputs a set of public keys given the prefix of some blockchain denoted chain. Thus the function $\mathsf{Comm}(\mathsf{chain})$ can be used to signal that a reconfiguration is needed and to establish common knowledge of the new consensus committee. If $\mathsf{Comm}(\mathsf{chain}[:\ell]) \neq \mathsf{Comm}(\mathsf{chain}[:\ell - 1])$, we say that the block $\mathsf{chain}[\ell]$ is a *reconfiguration-signaling* block.

In the following, we focus on describing the consensus mechanism that enables such reconfiguration for general choices of Comm. How to design the function Comm is an orthogonal problem, and has been discussed extensively in prior work [9,10]. Our idea is simply to terminate the current consensus instance and spawn a new one whenever a reconfiguration-signaling block becomes part of the finalized log. Since now there can be many consensus instances, nodes would output the sequential concatenation of the logs output by all instances spawned so far. Below we describe this idea more formally.

Notations. We use a session identifier $sid \in \{1, 2, 3, \ldots, \}$ to identify instances of the PiLi* protocol a node spawns sequentially over time. When a specific instance identified by sid terminates, the finalized chain it outputs is called the instance's *concluding chain* henceforth denoted CC^{sid}. We use the shorthand $\mathsf{CC}^{1..sid}$ to denote the the concatenation:

$$\mathsf{CC}^{1..sid} := \mathsf{CC}^1 || \mathsf{CC}^2 || \ldots || \mathsf{CC}^{sid}$$

Supporting reconfiguration. More formally, we use the following mechanism to support reconfiguration.

- At any time, let sid denote a node's present instance; the node outputs the following finalized chain where chain^{sid} is the output thus-far of the present instance sid:

$$\mathsf{CC}^{1..sid-1}||\mathsf{chain}^{sid}$$

- Let sid be a node's present instance and let chain^{sid} denote a blockchain in the present instance sid. We say that $\mathsf{chain}^{sid}[\ell]$ is a reconfiguration-signaling block iff

$$\mathsf{Comm}\left(\mathsf{CC}^{1..sid-1}||\mathsf{chain}^{sid}[:\ell]\right) \neq \mathsf{Comm}\left(\mathsf{CC}^{1..sid-1}\right)$$

- If the present instance sid outputs a blockchain chain^{sid} that contains a reconfiguration-signaling block and let $\mathsf{chain}^{sid}[\ell]$ be the first such block, then the concluding chain of the present instance CC^{sid} is defined as $\mathsf{CC}^{sid} := \mathsf{chain}^{sid}[:\ell]$.
- Whenever a node observes a reconfiguration-signaling block in the output chain of the present instance sid, then terminate instance sid and spawn a new instance $sid+1$. The set of consensus node for instance $sid+1$ is defined as $\mathsf{Comm}\left(\mathsf{CC}^{1..sid}\right)$.
- Finally, recall that all protocol messages are signed by the sender. For standard compositional reasons [17], we assume that whenever a node signs a message in instance sid, it prefaces the message with sid before signing; and correspondingly the verification algorithm would also preface the message with sid. We may also assume that if a node receives a message pertaining to an instance that has not yet been spawned, the message will be queued in a buffer and delivered when that instance is spawned.

5.2 Quality of the Confirmed Blockchain

In Section 4, we showed that our basic protocol achieves a weaker notion of liveness called chain growth. Chain growth requires that during periods of synchrony, honest nodes' output logs grow at a steady pace; but it does not impose any block quality requirement. To attain the liveness notion defined in Section 2.3, we would like to make sure that only blocks with good quality can become notarized during periods of synchrony. Roughly speaking, a block is of good quality iff it includes all *unconfirmed* transactions that have been pending in honest view for sufficiently long.

To achieve this goal, we now describe a very simple modification to our basic PiLi* protocol: we add an additional voting rule as follows (the voting rules described in Figure 2 still apply):

A node will vote for a proposal B only if either B or its parent chain contains all transactions the node has observed 2Δ or more rounds ago.

Intuitively, this additional voting rule has the following effect: if the current proposer is corrupt and censoring pending transactions, honest nodes will refuse to vote and choke the current proposer. This will trigger proposer re-election until some well-behaved proposer is found. We now prove that our PiLi* protocol with the above modification satisfies the liveness notion defined in Section 2.3.

Corollary 5.1. *Let $\lambda = \omega(\log \kappa)$ be any super-logarithmic function in κ. Then, our* PiLi* *protocol with the above modification satisfies consistency and $\lambda\Delta$-liveness w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects 0.5-weak-synchrony.*

Proof. It is not difficult to verify that consistency (i.e., Corollary 4.10) still holds and its proof is unaffected by this modification.

Below we focus on proving liveness. First, it is not difficult to verify that Lemma 4.12 and Theorem 4.14 still hold — specifically in the proof of Lemma 4.12, we need to observe that honest nodes will not refuse to vote on i^*'s proposal due to detection of censoring.

We now continue with the proof assuming that Lemma 4.12 and Theorem 4.14 hold. Suppose that $2\lambda'+6 = \lambda$ — note that λ' is super-logarithmic too. Henceforth we ignore the negligible fraction of bad executions in which signature forgeries or hash collisions occur, or in which Theorem 4.14 fails for the parameter λ' (in lieu of λ in Theorem 4.14).

Let r be any round that is at least $(2\lambda' + 5)\Delta$ rounds after the GST. By Theorem 4.14, at the end of round r, some honest node has output a block B that has not gained notarization in honest view yet by the end of round $r_1 := r - \lambda'\Delta$. Applying Theorem 4.14 again, by the end of round r_1, some honest node has output a block B_1 that has not gained notarization in honest view yet by the end of round $r_2 := r - 2\lambda'\Delta$. By consistency, B_1 must be an ancestor to the block B. Thus no honest vote for the block B can be cast in round r_2 or earlier, since by honest protocol definition and Fact 4.3, an honest vote may be cast for B only when there is a notarization for B_1 in honest view.

Since an honest node (henceforth denoted i^*) casts a vote for B in some round after r_2, by honest protocol definition, this block B or its parent chain must include all transactions i^* has observed at the beginning of round $r_2 - 2\Delta$.

Thus, the block B or its parent chain must include all transactions some honest node (not necessarily i^*) has observed at the beginning of round $r_2 - 3\Delta$. Note that in round $r + \Delta$, all honest nodes will have output B.

In summary, we have shown that for any round r^* that is at least $(2\lambda' + 6)\Delta$ rounds after the GST, all honest nodes must have output a block B that includes all transactions some honest node has observed at the beginning of round $r^* - (2\lambda' + 4)\Delta$. The above corollary then follows in a straightforward manner. $\square$

6 Practical Considerations

PiLi* is suitable for both permissioned blockchains or in decentralized proof-of-stake applications. In this section, we discuss how to use PiLi* in these settings.

6.1 Performance Considerations in a Permissioned Environment

To achieve high performance in a permissioned deployment environment, one approach would be to separate the proposers and the voters. This way, the proposer nodes can be better provisioned than the voters and can serve as dedicated infrastructure to accelerate the consensus protocol (but need not be trusted for consistency). Specifically, in a typical implementation, the proposer nodes can provide an optimistic network layer on the fast path: each proposer acts as a relay to forward the blocks proposed to the voters over a direct IP link; they then collect the voters' votes and send the aggregated notarization back to the nodes. Underneath we can employ a (slower) network diffusion mechanism (e.g., the diffusion mechanism employed by Bitcoin or Ethereum) where nodes keep telling each other the freshest notarized chain they have seen, and help each other get caught up on notarized blocks that they do not have.

6.2 Considerations for Decentralized Proof-of-Stake

Earlier in the paper, we primarily focused on a permissioned setting for concreteness. In a permissioned setting especially if high performance is a primary concern, a stability-favoring proposer rotation policy might be more desirable.

Our reconfigurable PiLi* is also a great consensus candidate for a decentralized proof-of-stake setting. In decentraized proof-of-stake, anyone can bid to become part of the consensus committee by freezing stake on the blockchain. In this way, the consensus committee can be re-elected periodically and the our reconfigurable PiLi* (Section 5.1) readily supports such re-election.

Democracy-favoring proposer eligibility. We describe a *democracy*-favoring proposer eligibility policy that is desirable for such a decentralized environment. Suppose that each elected committee will serve for one "committee-term". Recall that in reconfigurable PiLi*, we would spawn a new consensus instance for each committee-term. Let λ be a suitable super-logarithmic function in κ that denotes the number of epochs in each committee-term; henceforth, we assume that each committee-term's epoch numbers are renumbered as $1, 2, \ldots, \lambda$. Let S_k denote a public random seed that can be determined before the k-th committee-term starts — we will describe how to select such a seed shortly after. Now, a node $i \in [n_k]$ is said to be an eligible propser to propose a block of epoch $e' \in [1, \lambda]$, iff

$$i = (H^*(S_k||e) \mod n_k) + 1$$

where n_k is the size of the k-th committee. Note that once the k-th committee is determined, we can easily assign each node a unique index, e.g., by lexicographical ordering of their public keys.

To understand this scheme, first pretend that S_k is chosen by a trusted party after the k-th committee is determined (we will later describe how to remove this assumption). If so, then for any super-logarithmic function $\lambda' = \omega(\kappa)$ and any constant $C > 1$, the following good event must happen except with negligible (in κ) probability: over the course of any λ' consecutive epochs, there must exist C consecutive epochs whose proposers are all honest. Using almost the same proof as Theorem C.7, whenever C consecutive epochs all have honest proposers for some appropriate $C \approx 30$, honest nodes' final output must grow by at least one honestly-proposed-block. Summarizing the above, in every window of super-logarithmically many consecutive epochs, honest nodes' output logs must have grown by at least one honestly-proposed-block.

We now discuss how to resolve two concerns: 1) how to select the random seed for seeding the proposer eligibility random oracle H^*; and 2) how to improve the concrete parameters of the scheme.

A two-phase approach for electing committee and seed. One possible attack we need to defend against is the following: after the adversary observes the random seek S_k, it can choose the indices of the seats corrupt nodes want to hold in the k-th committee such that corrupt nodes can elected as proposer for many consecutive epochs — such an attack can deny liveness to the blockchain for up to $\Omega(n_k)$ blocks. Therefore, it is important that the random seed S_k be selected after the k-th committee is determined. To this end, we can employ an idea proposed in Snow White [10] and subsequently adopted by later versions of Algorand [9] too:

- When the i-th committee is serving and we are running the i-th consensus instance, we will use two phases, called `bid` and `seed` respectively, to determine the committee and the random seed of the next committee-term. Each phase encompasses super-logarithmically many epochs.

- During the `bid` phase, all interested players freeze stake on the blockchain to become elected into the next committee. We assume that the proof-of-stake system may specify some election policy, e.g., a minimal amount of stake that must be frozen.

- After the bid phase ends, the next consensus committee will have been determined. At this moment, there is a seed phase that consists of λ' blocks where λ' is a suitable super-logarithmic function in κ. We also assume that a proposer would include a random string of κ bits into the payload field of any block it proposes. Suppose that the seed phase encompasses blocks of epochs between $[e, e+\lambda']$, and let $\rho_e, \ldots, \rho_{e+\lambda'}$ denote the random strings of all blocks contained in the seed phase — if an epoch is skipped we simply let the corresponding ρ be $\perp$.

  We can simply let the concatenation $\rho_e || \ldots || \rho_{e+\lambda'}$ be the random seed for the next epoch.

As earlier works [9,10] have shown, although a corrupt proposer can arbitrarily manipulate the random string inserted into any block it proposes, as long as one of the random strings among $\rho_e, \ldots, \rho_{e+\lambda'}$ is created by an honest propser, then the concatenated string $\rho_e || \ldots || \rho_{e+\lambda'}$ has high entropy. Such a high-entropy is safe for seeding the proposer-eligibility random oracle H^*, as long as H^* is used to elect super-logarithmically many proposers [10] — intuitively, although the adversary can get corrupt nodes elected for a few number of epochs, it cannot consistently win for λ' number of epochs except with $\exp(-\Omega(\lambda')) \cdot \mathsf{poly}(\kappa)$ probability. More specifically, when the random seed is elected using the above mechanism rather than by a trusted party, using the same analysis as Snow White [10], we lose only by a polynomial factor in the failure probability and thus we retain the same asymptotical guarantee: except with negligible probability, in every window of super-logarithmically many consecutive epochs, honest nodes' final logs must have grown by at least one honestly-proposed block.

Concrete parameters for the decentralized setting. As mentioned, our scheme requires $O(1)$ number of consecutively honest proposers for liveness. In practice, we may consider the following tweak for better concrete parameters: every multiple of 32 is a proposer-relection opportunity and once elected, every proposer proposes blocks for the next 32 epochs.

Finally, it is not hard to see from our proofs that some of our constants can be easily tightened by making minor tweaks to our scheme — in preparing this manuscript, we made an explicit choice not to include all these minor tweaks in our presentation to maximize conceptual simplicity, to help illustrate this "streamlined" paradigm not only in terms of construction but also in terms of proof technique, and to help the reader understand the essence of this new paradigm. We leave it as exciting future work to have the most concretely efficient scheme of our paradigm implemented and open sourced.

References

[1] Gmail and google drive are experiencing issues, and naturally people are complaining about it on twitter. `https://www.huffingtonpost.com/entry/gmail-issue_n_3099988`.

[2] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. Efficient synchronous byzantine consensus. In *Financial Crypto*, 2019.

[3] I. Abraham, G. Gueta, and D. Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

[4] S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. Secure mpc: Laziness leads to god. Cryptology ePrint Archive, Report 2018/580, 2018.

[5] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC*, pages 27–30, 1983.

[6] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[8] T.-H. H. Chan, R. Pass, and E. Shi. Pala: A simple partially synchronous blockchain. Cryptology ePrint Archive, Report 2018/981, 2018.

[9] J. Chen and S. Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

[10] P. Daian, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. In *Financial Cryptography and Data Security (FC)*, 2019.

[11] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

[12] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[13] Y. Guo, R. Pass, and E. Shi. Synchronous, with a chance of partition tolerance. Cryptology ePrint Archive, 2019.

[14] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Crypto*, 2017.

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

[16] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[17] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. In *STOC*, 2002.

[18] J. Loss and T. Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018.

[19] S. Micali and V. 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.

[20] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *CCS*, 2016.

[21] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[22] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.

[23] R. Pass and E. Shi. Rethinking large-scale consensus. In *CSF*, 2017.

[24] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.

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

[26] V. G. Vitalik Buterin. Casper the friendly finality gadget. `https://arxiv.org/abs/1710.09437`.

Supplementary Materials

A Proofs for Optimistic Responsiveness

Theorem A.1 (Optimistic responsiveness: restatement of Theorem 4.16)**.** *Suppose that in some good execution, an honest proposer i takes over in some round t that is at least 3Δ rounds after the GST; and moreover, at least $3n/4$ number of nodes are honest. Let $\delta \leq \Delta$ be the actual maximum honest message delay after the GST in this execution. Then, suppose that i proposes an epoch-e block in round $t' \geq t$, we have the following:*

- *the proposed block obtains notarization in every honest node's view by the beginning of round $t' + 2\delta$;*
- *by the beginning of round $t' + 2\delta$, every honest node will have fast-forwarded to epoch $e + 1$ or greater;*
- *i will successfully propose a block at epoch $e + 1$ extending from a strongly notarized parent of epoch e.*

Proof. We first show it for the skip block proposed in round t. First, using a simlar argument as the proof of Lemma 4.12, we can conclude the following: 1) every honest node will have voted for this proposal by the end of $t + \delta$, and all honest node will have observed a notarization for the proposal at the beginning of round $t + 2\delta$; 2) every honest node will enter epoch $e + 1$ by fast-forwarding; and 3) i will propose an epoch-$(e + 1)$ block extending from a strongly notarized parent of epoch-e upon entering epoch $e + 1$ (i.e., no later than round $t + 2\delta$).

Now, suppose that above claims hold for every block proposed by i with epoch numbers $e, e + 1, \ldots, e' - 1$ where e is the first skip block proposed in round t. By induction hypothesis, i will propose an epoch-e' block in some round $t' \geq t$ extending from a strongly notarized parent of epoch-$(e' - 1)$. Note that all honest nodes will have received the proposal, its parent chain and the parent's strong notarization at the beginning of round $t' + \delta$. Using a similar argument as in the proof of Lemma 4.12, between rounds $[t' + \delta, t' + 2\delta]$, every honest node must be in epoch e'. Clearly, no honest node will reject the proposal due to the parent being too stale or due to having seen conflicting notarizations since the last skip block (i.e., the block proposed by i in round t). Thus, every honest node will have voted for the epoch-e' proposal by round $t + \delta$ and every honest node will have seen its notariation at the beginning of round $t + 2\delta$. Thus all honest nodes will have fast-forwarded to epoch $e' + 1$ by the beginning of round $t + 2\delta$; and by round $t + 2\delta$ i will have proposed a block of epoch $e' + 1$ extending from a strongly notarized parent of epoch e'. □

B Echo Mechanism

As mentioned earlier, a brute-force mechanism to realize Assumption 2 from Assumption 1 is to have nodes echo (i.e., multicast) all messages they have seen in every round. This approach, however, would be very expensive.

We can rely on a more efficient mechanism to realize Assumption 2 from Assumption 1 — this approach was in fact described in the work by Guo et al. [13] and we include it here for completeness. Effectively, online nodes only need to retry a few times whereas offline nodes might need to keep retrying until shortly after they come online — however, since nodes are unaware whether they are online or offline, the actual protocol needs to look for a different indicator as to when to stop retrying. The idea is to stop retrying when more than $n/2$ nodes have echoed the message. More concretely, nodes rely on the following echo mechanism to realize Assumption 2:

- An echo from a node i for a message m is of the format $(\mathsf{echo}, \mathsf{m})$ tagged with node i's signature[3].
- Upon observing a fresh message m (including messages contained in an echo, messages input from $\mathcal{Z}$, messages received over the network, or messages the node tried to send itself), multicast an echo for m in every round until more than $n/2$ valid echos for m have been heard from distinct nodes. Note that nodes need not recursively send echos for an echo.

Theorem B.1. *Consider a good execution: the above echo mechanism satisfies Assumption 2.*

Proof. Suppose that some honest node stops echoing m in round r, we prove that everyone in $\mathcal{O}_{r'}$ where $r' \geq r + \Delta$ will have seen m.

Let $r'' \leq r$ be the first round in which some honest node hears more than $n/2$ echos for m. Some node $i^* \in \mathcal{O}_{r''-1}$ must have sent an echo for m; and since r'' is the first round in which some honest node hears more than $n/2$ echos for m, it must be that i^* sent an echo for m in round $r''-1$ too. Thus, by Assumption 1, every node honest and online in round $r' \geq r'' + \Delta$ will have seen m. □

C Warmup: Basic PiLi for Classical Synchrony

In this section, we describe an extremely simple warmup protocol and we prove it secure in the classical synchronous model assuming honest majority.

C.1 A Fully Synchronous Execution Model

We consider a set of n consensus nodes numbered $1, 2, \ldots, n$. The nodes have access to a public-key infrastructure (PKI), and we use $(\mathsf{pk}_i, \mathsf{sk}_i)$ to denote node i's public- and secret-key pair where $i \in [n]$. We adopt a standard *synchronous* model of protocol execution. Within a synchronous round, the following takes place. First, nodes receive messages at the beginning of each round; they then perform computation and process these messages; finally, they send messages. Any honest message sent in round r will be received by honest recipients at the beginning of round $r+1$. Henceforth in the paper, "the end of round r" and "the beginning of round $r+1$" are used interchangably.

We assume that the adversary $\mathcal{A}$ *statically* corrupts a minority set of f nodes before the PKI is chosen and moreover $n \geq 2f+1$. Corrupt nodes are entirely under the control of the adversary: the messages they receive are forwarded to $\mathcal{A}$, and the corrupt nodes send whatever messages $\mathcal{A}$ instructs them to send. We assume a rushing adversary, i.e., corrupt nodes can send messages in a round after looking at honest nodes' messages sent in the same round.

[3]We assume that all protocol instances share the same echo-PKI. For composition, we assume that the message signed is tagged with the session identifier *sid* and verification is aware of the *sid* too.

C.2 The Basic PiLi Protocol

C.2.1 Definitions

We formalize some useful definitions; all of these notions are defined in the most natural manner.

Block. Each block is of the format $(e, \mathsf{TXs}, h_{-1})$ where $e \in \mathbb{N}$ denotes an epoch number, $\mathsf{TXs} \in \{0,1\}^*$ denotes the block's payload (e.g., a batch of transactions to confirm), and $h_{-1} \in \{0,1\}^\kappa$ denotes the parent hash, i.e., hash of the blockchain the block extends from.

Valid blockchain. A valid blockchain denoted chain is a sequence of blocks where for $1 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i]$ denotes the i-th block in chain. We often use the following useful blockchain notation:

- $\mathsf{chain}[-1]$ denotes the last block in chain and $\mathsf{chain}[: i]$ denotes the first i blocks of chain;
- assume that a block of epoch e exists in chain, we would then use $\mathsf{chain}\langle e\rangle$ to denote this block at epoch e in chain; and use $\mathsf{chain}\langle : e\rangle$ to denote the prefix of chain ending at the block $\mathsf{chain}\langle e\rangle$.

For a blockchain chain to be valid, the following must be respected:

1. For all $1 \leq i < j \leq |\mathsf{chain}|$, $\mathsf{chain}[i].e < \mathsf{chain}[j].e$; and
2. It must be that $\mathsf{chain}[1].h_{-1} = \perp$, and for every $2 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i].h_{-1} = H(\mathsf{chain}[: i-1])$ where H is randomly sampled from a collision-resistant hash family[4].

In other words, in a valid blockchain, all blocks' sequence numbers must strictly increase (but sequences numbers may jump); and moreover, every block must correctly contain the parent chain's hash.

Epoch-e blockchain and "fresher than" relation. We say that chain is an epoch-e chain iff $\mathsf{chain}[-1].e = e$. A chain is said to be *fresher* than another chain' iff $\mathsf{chain}[-1].e > \mathsf{chain}'[-1].e$.

Vote and notarization. A pair (h, σ) from a purported sender $i \in [n]$ is said to be a valid vote for some chain, iff $h = H(\mathsf{chain})$ and moreover $\Sigma.\mathsf{Verify}_{\mathsf{pk}_i}(h, \sigma) = 1$.

A collection of $f + 1$ valid votes for chain from distinct senders is said to be a *notarization* for chain.

Remark C.1. *Note that since our blockchain validity definition requires that each block specify the parent hash, assuming no hash collision, a block may be considered an alias for a chain. Thus, we often use the term "a vote/notarization for* chain*" and the term "a vote/notarization for the block* $\mathsf{B} := \mathsf{chain}[-1]$*" interchangeably.*

[4]In practice, H should be incrementally evaluated using a base hash function H_0: let $H(\mathsf{chain}) := H_0(H(\mathsf{chain}[: -1])||\mathsf{chain}[-1])$ if $|\mathsf{chain}| > 1$; and let $H(\mathsf{chain}) := H_0(\mathsf{chain}[1])$ if $|\mathsf{chain}| = 1$.

Basic PiLi for classical synchrony

In every epoch $e = 1, 2, \ldots$, every node performs the following. Henceforth, we assume that every message is tagged with the (purported) sender's identity.

1. **Propose.** Let chain be the freshest notarized chain observed so far. Let TXs be a set of outstanding transactions. and let $\mathsf{B} := (e, \mathsf{TXs}, H(\mathsf{chain}))$. If the node is an *eligible* proposer for proposing B, multicast (B, σ) where σ is the node's signature on B.

2. **Vote.** If a node has received a proposal of the form $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$ with a valid signature from some node i that is an eligible proposer for proposer B, vote on B iff the following conditions hold:

   - the node has observed some chain such that $h_{-1} = H(\mathsf{chain})$ as well as a notarization for chain; and
   - unless $e = 1$, the following condition must hold: let chain' be the freshest notarized chain the node had observed at the beginning of the *previous* epoch $e - 1$, it must be that chain (whose hash matches h_{-1}) is at least as fresh as chain'.

   To vote on $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$, let chain be the blockchain matching h_{-1}: sign $h := H(\mathsf{chain}||\mathsf{B})$ to obtain the signature σ, and multicast (h, σ).

   A node is allowed to vote on multiple proposals if they all satisfy the above conditions.

Finalize. At any time, let chain be the freshest notarized chain ending with six blocks at consecutive epochs — and let $e, e+1, \ldots, e+5$ be these six epochs. If for every $e' \in \{e, e+1, \ldots, e+5\}$ the only epoch-e' notarized block observed so far belongs to chain, then output $\mathsf{chain}\langle : e\rangle$.

Figure 2: The basic PiLi blockchain for classical synchrony.

C.2.2 Protocol Description

Conventions. In our protocol all messages are always *multicast* to everyone. We assume that a node always signs any message it wants to send, and the message is tagged with the purported sender. If a message with an invalid signature is received, it is discarded immediately without being processed.

We assume that every node always echos (i.e., multicasts) every fresh message it has seen. With such echoing, we may assume the following stronger network delivery assumption: if an honest node ever sends or receives a message m at the beginning of round r, then all honest nodes will have observed m by the beginning of round $r + 1$.

Epochs and proposer eligibility. The protocol proceeds in *epochs.* We assume that there is a publicly known and efficient function that can check whether a node i is *eligible* for proposing a block denoted $\mathsf{B} := (e, \mathsf{TXs}, h_{-1})$ extending from a parent chain (whose hash matches h_{-1}). For the time being, it is easiest to think of a round-robin policy, i.e., node $(e \mod n) + 1$ is the only eligible proposer for proposing any epoch-e block. We discuss other proposer-eligibility policies in Sections 3 and 6.2.

Protocol. Each epoch contains exactly two synchronous rounds called **Propose** and **Vote** respectively.

Our basic PiLi protocol proceeds in a natural manner: every epoch, an eligible proposer proposes a next block extending its current freshest notarized chain. Consensus nodes vote on all valid proposals from eligible proposers, as long as the proposed block extends from a chain that is sufficiently fresh. When a block collects $f + 1$ votes, it is considered *notarized.* Thus if an epoch has a single honest proposer, only a unique block of this epoch can be notarized. Not all notarized blocks are considered final. If, however, at the beginning of epoch e the freshest notarized chain (denoted chain) ends with 6 blocks at epochs $e-6, e-5, \ldots, e-1$ respectively, and moreover there are no other notarized blocks at these epochs have been observed, then $\mathsf{chain}\langle : e-6\rangle$ is considered *final* (i.e., chop off the trailing 5 blocks from chain).

A formal description of the protocol is presented in Figure 2. In Section C.3, we prove the following:

- *Consistency.* The protocol satisfies consistency as long as the majority of the nodes are honest (and even when proposers are corrupt); and
- *Chain growth under good proposers.* Suppose that for each of $C \geq 6$ consecutive epochs $e + 1, e + 2, \ldots, e + C$, there is a single honest proposer per epoch and no corrupt node is elected proposer, then by the end each of the epochs $e+6, e+7, \ldots, e+C$, any honest node's finalized chain would grow one honest block per epoch. More intuitively, whenever sufficiently many consecutive epochs each has a single honest proposer and no corrupt proposer, then after 6 epochs of wait-time, afterwards, every epoch will have a new block finalized for every honest node.

C.3 Proofs

Good executions. Henceforth we ignore the negligible fraction of bad executions where honest nodes' signatures are forged or where a hash collision is found. For simplicity, in all theorems and lemmas below, we omit stating "except with negligible probability" — however, the reader should keep in mind that all theorems and lemmas below hold only for "good executions" where honest nodes' signatures are not forged and hash collisions do not exist in the union of honest nodes' views.

Additional terminology. We introduce some additional useful terminology:

- We say that chain is *a notarized chain in honest view* in some execution if there exists some round in which some so-far honest node has observed chain and a notarization for it.
- For convenience, we pretend that there is an imaginary genesis block denoted $\mathsf{chain}[0] := (0, \perp, \perp)$ at index 0 in every valid blockchain chain.
- $\mathsf{chain} \preceq \mathsf{chain}'$ means chain is a prefix of chain'; by convention, $\mathsf{chain} \preceq \mathsf{chain}$.

A simple observation is the following.

Fact C.2. *Consider some good execution: if* chain *is a notarized chain in honest view at the beginning of round r, then every prefix of* chain *is a notarized chain in honest view at the beginning of round r too.*

Proof. All statements below are with respect to the honest view at the beginning of some round r. For contradiction, suppose that $\mathsf{chain}[:\ell]$ is a notarized chain in honest view but $\mathsf{chain}[:\ell-1]$ is not. Since no honest node has observed a notarization for $\mathsf{chain}[:\ell-1]$, no honest node has voted for $\mathsf{chain}[:\ell]$ by our protocol definition. This means that $\mathsf{chain}[:\ell]$ cannot gain notarization in honest view in a good execution. □

C.4 Consistency

Lemma C.3. *Consider some good execution: suppose* chain *is some valid notarized chain in honest view and there are two consecutive blocks in* chain *at epochs e and $e' > e$ respectively. Let* $\mathsf{chain}\langle: e\rangle$ *denote the prefix of* chain *ending at the epoch-e block. It must be that* all *honest nodes have observed a notarization for* $\mathsf{chain}\langle: e\rangle$ *by the beginning of epoch $e' + 1$.*

Proof. Suppose for the sake of contradiction that some honest node has not observed a notarization for $\mathsf{chain}\langle: e\rangle$ by the beginning of epoch $e' + 1$. This means that no honest node has observed a notarization for $\mathsf{chain}\langle: e\rangle$ by the beginning of the **Vote** round of epoch e' (since otherwise all honest nodes would have observed it by the beginning of epoch $e' + 1$). This means that no honest node will vote on $\mathsf{chain}\langle: e\rangle$ in epoch e'; combining Fact C.2, we have that $\mathsf{chain}\langle: e\rangle$ cannot be followed by a notarized block at epoch e' in any notarized chain in honest view — this contradicts the assumption of this lemma. □

Good event. We define the good event $\mathcal{G}(e)$ to be the following where $e > 1$: some honest node has observed a notarized chain that contains both an epoch-$(e-1)$ block and an epoch-e block.

Lemma C.4. *Consider some good execution such that for some $e > 1$, $\mathcal{G}(e)$ is true. Then, for every notarized* chain *in honest view, if for some i* $\mathsf{chain}[i].e \geq e + 2$, *it holds that* $\mathsf{chain}[i-1].e \geq e - 1$.

Proof. By Lemma C.3, if $\mathcal{G}(e)$ is true, then all honest nodes will have observed an epoch-$(e-1)$ notarized block by the beginning of epoch $e + 1$. Thus, by our honest protocol definition, in epochs $e + 2$ or higher, no honest node will vote to extend any chain that ends at epoch $e - 2$ or smaller. Thus, any notarized block at epoch $e + 2$ or higher cannot extend from a block at epoch $e - 2$ or smaller. □

Lemma C.5. *Consider some good execution in which $\mathcal{G}(e+1)$ is true for $e > 0$: then for any notarized* chain *ever in honest view such that* $\mathsf{chain}[-1].e > e+2$*, it must be that* chain *contains at least one block at epoch e, $e+1$, or $e+2$, i.e., it cannot be that* chain *skips all three epochs e, $e+1$, and $e+2$.*

Proof. For the sake of contradiction, suppose that there is some notarized chain in honest view such that $\mathsf{chain}[-1].e > e+2$ and moreover chain skips epochs e, $e+1$, and $e+2$. Consider the earliest (i.e., least fresh) block in chain whose epoch is greater than $e+2$ and let $\mathsf{chain}[i]$ be this block. It must be that $\mathsf{chain}[i-1].e < e$ but this violates Lemma C.4. □

Theorem C.6 (Consistency). *Consider some good execution: suppose that* ch *is output by honest node i at some time t and* ch' *is output by honest node j at some time t' (where the nodes i and j and the times t and t' need not be distinct). Then, it holds that* $\mathsf{ch} \preceq \mathsf{ch}'$ *or* $\mathsf{ch}' \preceq \mathsf{ch}$.

Proof. Let chain be the freshest notarized chain of node i at time t, and let chain' be the freshest notarized chain of node j at time t'. Further, suppose that chain and chain' ends at epochs e and e' respectively. Without loss of generality, assume $e' \geq e$. By definition, $\mathsf{ch} := \mathsf{chain}\langle: e-5\rangle$ and $\mathsf{ch}' := \mathsf{chain}'\langle: e'-5\rangle$. It suffices to show that $\mathsf{ch} \preceq \mathsf{chain}'$. Suppose this is not true for the sake of reaching a contradiction.

We next show that chain' has some block with epoch in $\{e-2, e-1, e\}$. If $e' = e$, then this is satisfied. Otherwise, we apply Lemma C.5. Since the truth of $\mathcal{G}(e-1)$ is witnessed by chain, and the last block of chain' is of epoch $e' \geq e+1$, it follows that chain' must contain some block with epoch in $\{e-2, e-1, e\}$, as required.

Let $\mathsf{chain}'\langle e^*\rangle$ be the earliest block in chain' where $e^* \in \{e-2, e-1, e\}$. We apply Lemma C.4. Since the truth of $\mathcal{G}(e-4)$ is witnessed by chain, and chain' contains a block with epoch $e^* \geq e-2$, the parent block B of $\mathsf{chain}'\langle e^*\rangle$ has epoch at least $e-5$.

By our assumption, B is not in ch. Now, by Lemma C.3, by the end of epoch $e^* \leq e$ (which is the beginning of epoch e^*+1), all honest nodes would have observed a notarization for B. Note that no honest node can have seen any epoch-e block notarized before the end of epoch e. Thus, t must be at least the end of epoch e or larger. However, since node i output $\mathsf{chain}\langle: e-5\rangle$ at time t, by the definition of the honest finalization procedure (see Figure 2), it must be that node i has not seen a notorization for B at time t. We have thus reached a contradiction.

□

C.5 Liveness

Theorem C.7 (Liveness). *Consider some good execution: suppose that in each of 6 consecutive epochs $e, e+1, e+2, \ldots, e+5$, a single honest node is elected proposer and and no corrupt node is elected proposer. Then, at the beginning of epoch $e+6$, all honest nodes must output a finalized chain ending at epoch e.*

Note that if the above theorem holds, at the end of epoch $e+6$, honest nodes' finalized chain ends at an epoch-e block proposed by an honest node, and this honest node must include all outstanding transactions into this block.

Proof. If some epoch e, a single honest is elected proposer and no corrupt node is proposer, then a *unique* epoch-e block will be proposed in the **Propose** round of this epoch. To prove the above theorem, it suffices to argue the following:

1. *Progress:* all honest nodes will have oberved a notarization for the proposed epoch-e block by the end of epoch e;

2. *Uniqueness:* there cannot be two different epoch-e blocks notarized in honest view.

Uniqueness is obvious since the single honest proposer of epoch e makes a unique proposal and thus honest nodes will not vote for any other epoch-e block other than the proposed. We now argue about progress. If $e > 1$ then for any notarized chain ch seen by any honest node at the beginning of the **Vote** round of epoch $e - 1$, all honest nodes must have seen ch by the beginning of epoch e; thus an honest proposer of epoch e must propose to extend some chain that is at least as fresh as ch. Thus all honest nodes will vote on the proposal by this honest proposer in epoch e, and all honest nodes will have observed a notarization for the proposal by the end of epoch e.

Applying the same argument to epoch $e + i$, for $i \in \{1, \ldots, 5\}$, we can conclude that by the beginning of epoch $e + 6$, all honest nodes must have observed the notarized chain containing the 6 blocks proposed by the unique honest node from epochs e to $e + 5$, and observed no other notarized blocks in these epochs.

Therefore, according to the finalization procedure specified in Figure 2, by the beginning of epoch $e + 6$, all honest nodes must output the same finalized chain ending at epoch e. $\square$