PaLa: A Simple Partially Synchronous Blockchain*

T-H. Hubert Chan Rafael Pass Elaine Shi

October 14, 2018

Abstract

Classical-style BFT protocols use two or more rounds of voting to confirm each block, e.g., in PBFT, they are called the "prepare" round and the "commit" round respectively. Recently, an elegant pipelining idea came out of the cryptocurrency community, i.e., *if each block required two rounds of voting, why not piggyback the second round on the next block's voting?* We refer to this idea as the pipelined-BFT paradigm.

We describe a simple partially synchronous blockchain protocol called PaLa that is inspired by the pipelined-BFT paradigm. In PaLa, a proposer proposes a block extending the freshest *notarized* chain seen so far. Consensus nodes vote on the proposal if certain conditions are met. When a block gains at least $\frac{2n}{3}$ votes it becomes *notarized*. A block becomes *finalized* if the next immediate block becomes notarized too.

We propose a conceptually simple and provably secure committee rotation algorithm for PaLa. We also describe a generalization called "doubly-pipelined PaLa" that is geared towards settings that require high throughput.

1 Introduction

We describe PaLa, a conceptually simple, partially-synchronous blockchain protocol that tolerates fewer than 1/3 corruptions. In PaLa, a proposer proposes a block extending the freshest *notarized* chain seen so far. Consensus nodes (a.k.a. committee members) vote on the proposal if certain conditions are met. When a block gains at least $\frac{2n}{3}$ votes it becomes *notarized*. There are two types of blocks in a valid blockchain, *normal* blocks and *timeout* blocks, and a different set of validity constraints apply to each type. At any point of time, every block in a notarized chain up till the last normal block (exclusive) are considered *finalized*.

PaLa's design is streamlined. There are only two types of messages, block proposal and vote. Under good conditions, each block requires only a single round of voting to confirm — but for finalization, one must wait for the next normal block to gain notarization too.

In general two philosophies exist regarding proposer election: the *democracy*-favoring approach (e.g., Dfinity [11], Algorand [8]) switches proposer for every confirmed block; and the *stability*-favoring approach (e.g., PBFT [7] and Thunderella [14]). Our paradigm enables using either of the policies.

We additionally describe a simple method for changing the committee without waiting for any synchronization event and while keeping the streamlined execution of the protocol.

*"Pili-pala" is the sound of thunder in Chinese; it also means fast, furious, and streamlined. PaLa is also short for "PArtition-LAsting", and a subtle reference to the part-time PArLiAment [12].

1.1 Background on the Pipelined-BFT Paradigm

Classical-style BFT protocols use two or more rounds of voting to confirm each block, e.g., in PBFT, they are called the "prepare" round and the "commit" round respectively. Recently, an elegant pipelining idea came out of the cryptocurrency community, i.e., *if each block required two rounds of voting, why not piggyback the second round on the next block's voting?* This idea was implicitly described in the elegant Casper-FFG work by Buterin and Griffith [15] in the form of a "finality gadget" for a proof-of-work blockchain. Forums and blog-posts online [2, 3]indicate that DPoS (i.e., the consensus protocol adopted by EOS, a top-5 cryptocurrency by market cap [1]) might be considering a similar approach, although no formal description of the DPoS protocol and proofs have been released at the time of the writing. The instantiations of this idea by Casper-FFG [15] and DPoS [2, 3]both achieve only *synchronous* performance, i.e., even when the network is capable of delivering packets much faster, these protocols can only confirm blocks at slower intervals. Specifically, DPoS proposes blocks only at fixed intervals [2,3]whereas in Casper [15], the block proposal is process is the underlying PoW blockchain, and thus block proposal happens only sporadically at random intervals.

The recent elegant work HotStuff by Abraham et al. [4] extended this "pipelining" approach to present a provably-secure BFT protocol tolerating 2/3 corruptions in a *partially synchronous* network. On a high-level, PaLa relies on intuitions very similar to HotStuff (but we give a somewhat simpler presentation of the idea which may be of value in implementations). In HotStuff, liveness relies on having a rotating proposer (i.e., the "democracy-favoring approach"), whereas in PaLa, we additionally show how to instantiate PaLa using a stability-favoring proposer rotation policy geared towards performance-demanding scenarios (see Section 9 for more discusssions on a concrete, real-world application scenario). Moreover, we propose a doubly-pipelined variant which is a further generalization of the pipelined-BFT paradigm necessary for performance-demanding scenarios. Additionally, as mentioned above, we present a simple approach for enabling committee switch (which again is needed for the applications we consider).

1.2 Our Protocol in a Nutshell

We here provide a description of the PaLa consensus protocol focusing on a "random" proposer election policy and a fixed committee. Let Δ be a bound on the network delay in a period of synchrony. As we are in the partially-synchronous model, the protocol only needs to make progress during periods of synchrony, but consistency/safety should always hold even under arbitrary network partitions. To simplify notation, we will use the alias *second* refer to $c\Delta$, and the alias *minute* to mean $c^2\Delta$ (i.e., c seconds) for some appropriate constant c (in the actual protocol $c = 6$.)

Each node maintains a local epoch counter e which starts off at 0 and a current chain chain of blocks $(b_1, b_2, \ldots, b_n)$; each block is of the form $(e, \mathsf{TXs}, h_{-1})$ such that the epoch numbers e are strictly increasing, and h_{-1} is a hash of the chain before the current block. We say π is a notarization for a chain chain if π contains valid signatures from $\geq 2/3$ fraction of the nodes for every block in the chain. The epoch number of a chain is defined as the epoch number of the last block. The protocol proceeds as follows:

Chain update: Every node stores the freshest notarized chain seen so far denoted chain. Whenever a node sees a valid chain chain' (of the above form) with a notarization, with a higher epoch number than the epoch number of its current chain chain — we refer to chain' as being "fresher" than chain — replace the current chain with chain'.

Block proposal: In each epoch e, a random node, determined by applying a hash function (a random oracle) to the epoch number, is elected "proposer". The proposer proceeds as follows:

- If the proposer's chain ends with a block of epoch $e-1$, the proposer can immediately propose a new block $(e, \mathsf{TXs}, H(\mathsf{chain}))$ and multicast it where TXs denotes the outstanding transactions to confirm.
- If the proposer's chain has an epoch number earlier than than $e-1$, (i.e., the previous proposer failed to get its block incorporated in the chain), *wait* for 1 second (to potentially receive a "fresher" chain), and next propose a new block (as in the previous case).

Notarization: In epoch e, whenever a node i receives a block $(e, \mathsf{TXs}, h_{-1})$ from the proposer of epoch e such that (1) it has seen a notarized chain chain_{-1} consistent with h_{-1}, and (2) chain_{-1} is at least as fresh as i's chain *at the beginning of epoch* e, and (3) it has not previously signed a block for epoch e, i signs the block and multicasts the signature.

Epoch advance: A node i advances to epoch e whenever either of the following events happen and it is currently in an epoch smaller than e: (1) it sees a notarized chain for epoch $e-1$, or (2) it receives signed $\texttt{clock}(e)$ messages from $\geq 2/3$ fraction of the nodes. If a minute has elapsed since a node entered epoch $e-1$, it will multicast $\texttt{clock}(e)$.

Roadmap. The above protocol is called basic PaLa, and will be formally described in Section 3 and proven secure in Section 4.

Then, we focus on describing a generalization of PaLa, called doubly-pipelined PaLa, geared towards a high-performance scenario. This application is motivated by ThunderCore, a blockchain company that aims to move Ethereum to the "fast-path" accelerated by dedicated infrastructure nodes (i.e., proposers) called "accelerators". We also describe simple but non-trivial techniques for performing committee reconfiguration with PaLa.

2 Model

2.1 Modeling a Partial Synchronous Network

Background: modeling partial synchrony with GST. We adopt a *partially synchronous* communication network, and our formal model is an extension of the "global stabilization time (GST)" model proposed by Dwork, Lynch, and Stockmyer [10]. Recall that the GST model assumes the following:

- eventually, at some point called the GST, the network will stabilize and honest messages will be delivered within Δ time.
- the partially synchronous protocol is provided with the parameter Δ but GST is unknown to the protocol.

Modeling partial synchrony for long-running protocols. The GST model phrased by Dwork et al. [10] is more suitable for single-shot concensus protocols. By contrast, a blockchain protocol is long-running, confirming an ever-growing, linearly ordered log (e.g., of transactions). We thus extend the GST model to be suited for this long-running scenario.

We formally define a notion called *periods of synchrony*. Intuitively, a period of synchrony is when the network is well-connected and honest nodes are more or less synchronized so that they can communicate with one another experiencing maximum Δ delay. Sometimes, however, the network can lose synchrony, e.g., during a major outage or partition. When such network partitioning

happens, a partially synchronous protocol should retain consistency regardless. On the other hand, liveness is only required during periods of synchrony. A partially synchronous protocol knows the parameter Δ (i.e., the protocol decription has Δ hardwired in it) but is unaware when the periods of synchrony will happen.

Definition 1 (Period of synchrony)**.** Fix some protocol execution. We say that the period $[t_0, t_1]$ is period of synchrony (w.r.t. this execution) iff every message sent by an honest node at its local time $t \leq t_1$ will also be received by any other honest node by its local time $\max(t_0, t + \Delta)$.

We note that the term "period of synchrony" has been adopted (sometimes slightly informally) in several existing works on partially synchronous consensus [4, 5]. The above Definition 1 is an explicit formulation of this notion which we adopt for our paper.

For convenience, throughout the paper, unless otherwise noted, we assume that a node would always echo every new message it sees by multicasting it to everyone[1] — note that all messages will be signed in our protocol. In this way, we may make a stronger period of synchrony assumption:

Definition 2 (Stronger notion for period of synchrony)**.** We say that the period $[t_0, t_1]$ is period of synchrony (w.r.t. this execution) iff every message **observed** by an honest node at its local time $t \leq t_1$ will also be observed by any other honest node by its local time $\max(t_0, t + \Delta)$.

Remark 1. Throughout the paper, we may assume that nodes are allowed bounded clock skew and in this case, the maximum clock skew among honest nodes is absorbed by the notation Δ.

2.2 Execution Model and Assumptions

There are a total of n nodes numbered $1, 2, \ldots, n$ where n is a publicly known parameter. We assume a public key infrastructure (PKI) and the public key of the i-th node is denoted pk_i, and the corresponding secret key denoted sk_i.

Corruption model. We assume that a polynomially-bounded adversary who can corrupt less than $\frac{n}{3}$ nodes. For simplicity, we will first prove security under a *static* corruption model, where the adversary makes the corruption choices prior to the start of execution. Later in Section A.1, we will explain how to extend our security guarantees to adaptive corruptions, i.e., when the adversary can corrupt nodes adaptively during the middle of the protocol execution.

Throughout the paper, we assume that *all honest nodes start executing the protocol at time* 0.

3 Warmup: Basic PaLa

3.1 Definitions and Notations

Aliases for units of time. For convenience, we use the terms a *second* (or sec for short) and a *minute* (or min for short) not to measure time by the wallclock, but as aliases as specific units of time (that are parameters input to the protocol). Throughout the paper, we assume the following:

$$1\text{sec} \geq 5\Delta \qquad \text{and} \qquad 1\text{min} \geq 6\text{sec}$$

[1] If the underlying network adopts a peer-to-peer diffusion mechanism like in Bitcoin and Ethereum, such echoing comes for free. In Section 8.2, we describe practical optimizations to our protocol that avoid such potentially wasteful echoing.

Valid block. A block is now of the form $(e, \mathsf{TXs}, h_{-1})$, i.e., we no longer need the timestamp field in the block; moreover, now each block carries an epoch number $e \in \mathbb{N}$. The definitions of TXs and h_{-1} are as before.

Blockchain notation. A valid blockchain, denoted chain, is an ordered sequence of blocks. Henceforth we use the following notations:

- $\mathsf{chain}[i]$ denotes the i-th block in chain where $1 \leq i \leq |\mathsf{chain}|$; for convenience we shall assume every blockchain chain is prefaced with an imaginary *genesis* block $\mathsf{chain}[0] := ((0,1), \bot, \bot, 0)$;
- $\mathsf{chain}[:i]$ denotes all blocks till the i-th block (inclusive);
- $\mathsf{chain}[-i]$ is an alias for $\mathsf{chain}[L - i + 1]$ where $L = |\mathsf{chain}|$; and

Blockchain validity. A blockchain chain is said to be valid iff

1. for every $1 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i].h_{-1} = H(\mathsf{chain}[:i-1])$.
2. for any $1 \leq i < j \leq |\mathsf{chain}|$, it must be that $\mathsf{chain}[i].e < \mathsf{chain}[j].e$, i.e., the epoch numbers contained in a valid blockchain must be strictly increasing (but can possibly skip).

We will later use the following terminology in our proofs of the basic PaLa scheme: if $\mathsf{chain}[i].e = \mathsf{chain}[i-1].e + 1$, then $\mathsf{chain}[i]$ is said to be a *normal* block; else it is said to be a *timeout* block.

"Fresher than" relation. We say that chain is fresher than chain' iff $\mathsf{chain}[-1].e > \mathsf{chain}'[-1].e$. If $\mathsf{chain}[-1].e = e$ we also say that chain is an epoch-e chain.

As before, we sometimes use the term block as an alias for the (prefix of the) chain ending at that block.

Votes and notarization. A signature σ is said to be a valid vote from a node $i \in [n]$ on chain if $\Sigma.\mathsf{verify}(\mathsf{pk}_i, H(\mathsf{chain}), \sigma) = 1$ where pk_i denotes node i's public key and Σ denotes the signature scheme in use.

A collection of valid votes from at least $\frac{2n}{3}$ distinct nodes on chain is said to be a *notarization* for chain.

Remark 2 (Block as an alias for chain)**.** Note that since each block refers to the hash of the parent chain it extends from, we often use the term "block" as an alias for the chain ending at that block (assuming no hash collisions happen). For example, when we say a vote or a notarization on a block, it is the same as a vote or a notarization on the chain ending at that block.

Proposer eligibility. We assume that node $i \in [n]$ is an eligible proposer for epoch e iff $i = (H^*(e) \mod n) + 1$ where H^* is a random oracle chosen after the adversary decides which nodes to corrupt[2].

[2]As pointed out in the earlier work [13], we can remove the random oracle with a common reference string (CRS) that is chosen after the adversary makes corruption choices, and instead use a $(\mathsf{PRF}_{\mathrm{crs}}(e) \mod n) + 1$ to determine the eligible proposer for epoch e.

3.2 Protocol

Internal clock synchronization. Each consensus node maintains a local clock that indicates the current epoch (henceforth called the node's *local epoch*). Initially, the local epoch is set to 1. Nodes use the following mechanism to advance their local epochs:

- If a node has been in local epoch e for more than 1min, sign and multicast $\texttt{clock}(e+1)$, indicating that it wants to advance to epoch $e+1$ (but do not advance to epoch $e+1$ yet at this point).
- Upon seeing a notarization for an epoch-$(e-1)$ block, or upon seeing signed $\texttt{clock}(e)$ messages from at least $\frac{2n}{3}$ distinct nodes, advance local epoch to e (if the local epoch is smaller than e).

Proposal. When a node is in local epoch e, choose a block B to propose (if possible) based on which of the following happens first:

- It sees an epoch-$(e-1)$ notarized chain denoted chain, then if eligible, propose the block $\mathsf{B} := (e, \mathsf{TXs}, H(\mathsf{chain}))$ where TXs is the set of outstanding transactions.
- Else, if 1sec has passed since it entered epoch e, choose the freshest notarized chain denoted chain observed so far, if eligible and B is a valid block extending from chain, then propose the block $\mathsf{B} := (e, \mathsf{TXs}, H(\mathsf{chain}))$.

Note that an honest node proposes no more than 1 block in the same epoch. Further, if a proposer entered epoch e due to receiving an epoch-$(e-1)$ notorization, it will immediately propose an epoch-e block at the beginning of the epoch.

Vote. When a node is in local epoch e, for the *first* epoch-e valid proposal of the form $(e, \mathsf{TXs}, h_{-1})$ ever heard (potentially earlier than local epoch e), vote on the proposal iff the following hold:

- the node has observed chain such that $H(\mathsf{chain}) = h_{-1}$, and observed a notarization for chain;
- the parent chain chain defined above is at least as fresh as the freshest notarized chain the node has observed by the beginning of its local epoch e.

Finalization. Let $\mathsf{Finalize}(\cdot)$ be defined as follows:

$$\mathsf{Finalize}(\mathsf{chain}) := \mathsf{chain}[:\ell-1], \text{ where } \mathsf{chain}[\ell] \text{ is the last } \textit{normal} \text{ block in } \mathsf{chain}.$$

In other words, $\mathsf{chain}[:\ell] \preceq \mathsf{chain}$ is the longest prefix of chain ending at two blocks with consecutive epoch numbers.

At any time, let chain be the freshest notarized chain a node has observed so far, if $\mathsf{Finalize}(\mathsf{chain})$ is longer than the current output log, replace the output log with $\mathsf{Finalize}(\mathsf{chain})$.

4 Proofs for the Basic PaLa

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 define a few useful terms.

- We say that chain is *a notarized chain in honest view* in some execution if there exists some round in which some honest node has observed chain and a notarization for it. Note that if chain is a notarized chain in honest view in some execution, then every prefix of chain is a notarized chain in honest view too.
- $\mathsf{chain} \preceq \mathsf{chain}'$ means chain is a prefix of chain'; by convention, $\mathsf{chain} \preceq \mathsf{chain}$.

4.1 Consistency

Lemma 1 (Uniqueness for each epoch number)**.** *Let* chain *and* chain' *be two notarized chains in the honest view of a good execution such that* $\mathsf{chain}[-1].e = \mathsf{chain}'[-1].e$*, it must be that* $\mathsf{chain} = \mathsf{chain}'$*.*

Proof. Suppose two different chain and chain' have the same epoch e in their ending blocks, and both chains obtain notarization in honest view. Assuming that there is no hash collision between the two chains, the total number of distinct signatures for $H(\mathsf{chain})$ and those for $H(\mathsf{chain}')$ must be at least $\frac{4n}{3}$. However, recall that any honest node casts no more than one vote for an epoch-e block, whereas corrupt nodes can vote on both $H(\mathsf{chain})$ and $H(\mathsf{chain}')$. Therefore, total number of distinct signatures for $H(\mathsf{chain})$ and those for $H(\mathsf{chain}')$ cannot exceed $n_H + 2n_C = n + n_C < \frac{4n}{3}$, where n_H and n_C denote the number of honest and corrupt nodes respectively. Thus, we reach a contradiction. □

Theorem 1. *Let* chain *and* chain' *be two notarized chains in honest view in a good execution, it must hold that either* $\mathsf{Finalize}(\mathsf{chain}') \preceq \mathsf{Finalize}(\mathsf{chain})$ *or* $\mathsf{Finalize}(\mathsf{chain}) \preceq \mathsf{Finalize}(\mathsf{chain}')$*.*

Proof. For contradiction's sake, suppose $\mathsf{Finalize}(\mathsf{chain})$ and $\mathsf{Finalize}(\mathsf{chain}')$ are not prefix of each other; let B_0 be their last common block and i_0 be the index such that $\mathsf{B}_0 = \mathsf{chain}[i_0] = \mathsf{chain}'[i_0]$.

For each of chain and chain', we define the notions of *stabilized* and *stabilized* blocks. We illustrate these notions with chain, and the blocks associated with chain' are defined in the same manner.

- *Stabilizing Block.* If the block $\mathsf{chain}[i_0 + 1]$ immediately following B_0 is a timeout block, then the stabilizing block B_2 in chain is the first normal block after B_0. Otherwise, the stabilizing block B_2 in chain is the first normal block after $\mathsf{chain}[i_0 + 1]$; this is well-defined because $\mathsf{Finalize}$ removes the last normal block in chain.
- *Stablized Block.* The stabilized block B_1 in chain is the one that immediately precedes the stabilizing block B_2.

Observe that both B_1 and B_2 are after B_0 in chain. We define the blocks B'_1 (stabilized block) and B'_2 (stabilizing block) in the same manner for chain'.

We next derive a contradiction by case analysis. First, observe that if both $\mathsf{chain}[i_0 + 1]$ and $\mathsf{chain}'[i_0 + 1]$ are normal blocks, then they have the same epoch number; but this contradicts Lemma 1. Hence, for the rest of the proof, we can assume that at least one $\mathsf{chain}[i_0 + 1]$ and $\mathsf{chain}'[i_0 + 1]$ is a timeout block.

Next, define $e_0 := \mathsf{B}_0.e$, $e := \mathsf{B}_2.e$ and $e' := \mathsf{B}'_2.e$. By Lemma 1, $e \neq e'$; without loss of generality, assume $e < e'$. We divide into two cases: (i) $e = e_0 + 2$, and (ii) $e > e_0 + 2$.

(i) Consider the case $e = e_0 + 2$. It must be the case that in chain, the normal blocks $\mathsf{B}_1 = \mathsf{chain}[i_0 + 1]$ and $\mathsf{B}_2 = \mathsf{chain}[i_0 + 2]$ have epoch numbers $e_0 + 1$ and $e_0 + 2$, respectively.

Furthermore, since B_2 is a normal block, there must have been a set S of more than $\frac{n}{3}$ honest nodes, each of which must have notarized B_2 during its local epoch $e_0 + 2$. In particular, each honest node in S has seen seen the notarization of the prefix of chain up to B_1 during its local epoch $e_0 + 2$.

On the other hand, we can deduce that the block $\mathsf{B}'_* = \mathsf{chain}'[i_0 + 1]$ immediately following B_0 in chain' must be a timeout block with some epoch number e'', where $e'' \geq e_0 + 3$, because of Lemma 1.

Because local epoch numbers cannot decrease, for each honest node in S, at the beginning of epoch $e'' > e_0 + 2$, it must have seen the prefix of chain up to B_1, which is strictly fresher than the prefix up to B_0. Hence, it follows that less than $\frac{2n}{3}$ nodes could have voted for the block B'_*.

(ii) Consider the case $e \geq e_0 + 3$. In this case, the stabilized block B_1 must be a timeout block with epoch number $e - 1 \geq e_0 + 2$.

Since B_2 is a normal block, there must be a subset S of more than $\frac{n}{3}$ honest nodes, each of which has notarized the prefix of chain up to B_1 during its local epoch e.

Observe that $e < e'$ implies that chain' must have some block from $\mathsf{chain}[i_0 + 1]$ to B'_2 whose epoch number is strictly larger than e. Hence, consider the block B'_* in chain' with epoch number e'' such that e'' is the minimum epoch larger than e. By the choice of B'_*, its preceding block has epoch number at most e; however, by Lemma 1, this preceding block must have epoch number strictly smaller than e.

Hence, we can conclude that at the beginning of its local epoch $e'' > e$, each honest node in S must have seen the prefix of chain up to B_1, which is strictly fresher than the prefix of chain' excluding blocks from B'_* onwards. Hence, it follows that less than $\frac{2n}{3}$ nodes could have voted for the block B'_* in chain'.

This completes the proof of the lemma.

□

4.2 Liveness

Fact 1. *Consider a good execution and let $[t_0, t_1]$ denote a period of synchrony. It must be that if some honest node is in local epoch e at some time $r \in [t_0 - \Delta, t_1]$ or earlier, then by time $r + \Delta$, all honest nodes are in epoch e or higher.*

Proof. Straightforward by the "strong period of synchrony" assumption. □

Lemma 2. *Consider a good execution and let $[t_0, t_1]$ denote a period of synchrony. Suppose that at some time $t \in (t_0 + 2\text{sec}, t_1]$, at most 2sec has passed since some honest node enters its local epoch e. Then, at time t, no honest node is in epoch $e' > e$ unless by t there exists in honest view notarizations for epochs e, $e + 1$, ..., $e' - 1$.*

Proof. By Fact 1, it cannot be that at time t some honest node is 1min or more into local epoch e. Thus no honest node would have sent $\texttt{clock}(e')$ by time $t+1$sec for $e' > e$. Now, by honest protocol definition, the only way for an honest node to enter epoch $e' > e$ is by receiving an epoch-$(e' - 1)$ notarization. If $e' - 1 > e$, to receive an epoch-$(e' - 1)$ notarization, at least one honest node must must first enter epoch $e' - 1$ and vote in the epoch, and thus there must exist in honest view a notarization for epoch $e' - 2$, and so on. □

Lemma 3 (Honest proposer can successfully propose)**.** *Consider a period of synchrony denoted $[t_0, t_1]$ in a good execution and let t be some time that belongs to the period of synchrony. If an honest proposer of epoch e first enters epoch e at $t \in (t_0 + 1\text{sec}, t_1 - 1\text{sec}]$, then it must successfully propose*

an epoch-e block; moreover, before it proposes an epoch-e block, there cannot be any notarization for any epoch $e' > e$ in honest view.

Proof. By Lemma 2, when the honest epoch-e proposer (denoted u) is 1sec deep into its local epoch e, no honest node (including u itself) can be in epochs $e' > e$ unless there is an epoch-e notarization in honest view — but this cannot happen until u proposes an epoch-e block. Therefore, if the epoch-e proposer has seen an epoch-$(e-1)$ notarized chain before it is 1sec deep into its local epoch e, then it must successfully propose by definition of the algorithm. Otherwise, when it is 1sec deep into its local epoch e, it will attempt to propose a block extending from the freshest notarized chain in its view. It suffices to show that this freshest notarized chain does not contain any block from epochs $e' \geq e$. Obviously no epoch-e block can have notarization in honest view before u proposes an epoch-e block; further, by Lemma 2, no honest node can be in local epoch $e' > e$ before u makes an epoch-e proposal (and thus no honest node would have voted for any epoch-e' block). Therefore, no epoch-e' block can have notarization in honest view before u makes an epoch-e proposal for $e' > e$. □

Lemma 4 (Honest proposal gains notarization)**.** *Consider a period of synchrony denoted $[t_0, t_1]$ in a good execution. Suppose that an honest node proposes an epoch-e block* B *extending from* chain *at time $t \in (t_0 + 1\text{sec}, t_1 - 1\text{sec}]$. Then, by time $t + 1\text{sec}$, every honest node will have seen a notarization for* $\mathsf{chain}||\mathsf{B}$.

Proof. When an honest proposer u proposes a block at $t \in (t_0 + 1\text{sec}, t_1 - 1\text{sec}]$, by time $t + \Delta$, every honest node will have received the parent chain the proposed block extends from and its notarization due to the "strong period of synchrony" assumption. By Fact 1, by $t + \Delta$, all honest nodes will have entered epoch e or higher. Note that an honest node can only propose an epoch-e block by the time it is 1sec into its local epoch e. By Lemma 2, by time $t + \Delta$ no honest node is in epoch $e' > e$ unless there is an epoch-e notarization in honest view by then — and in the latter case the lemma obviously holds due to the "strong period of synchrony" assumption and the fact that honest nodes will vote only for the epoch-e block proposed by u.

Therefore henceforth we may assume that for every node node i, at some point of time in $[t, t + \Delta]$, i has received the parent chain the proposed block extends from and its notarization, moreover i is in local epoch e. It would suffice to show that i will vote for the proposal when this happens (and this holds for every honest node i). To show this, it suffices to prove that the parent chain denoted chain the honest proposal extends from is at least as fresh as any honest node's freshest notarized chain at the beginning of its local epoch e. Now, if the proposed parent chain ends at epoch $e - 1$, then the claim obviously holds due to Lemma 3. Else, suppose the parent chain ends at $e' < e - 1$. This means that the honest proposer u must have proposed this block when it is 1sec into its local epoch e. This means that honest nodes cannot have entered epoch e or higher after $t - 1\text{sec} + \Delta$. Thus if some honest node has seen an epoch-e'' notarized chain where $e'' > e'$ at the beginning of its local epoch e, it must be that the honest proposer u has seen it when it is 1sec into its local epoch e (i.e., at time t) by the "strong period of synchrony" assumption. Therefore u cannot propose to extend a parent chain that is epoch $e' < e''$. □

Theorem 2 (Liveness)**.** *Consider a good execution and let $[t_0, t_1]$ denote a period of synchrony during this execution. Suppose that epoch-e and epoch-$(e+1)$ both have honest proposers henceforth denoted u and v respectively. Moreover, assume that u entered epoch e at time $t \in (t_0 + 1\text{sec}, t_1 - 2\text{sec}]$. It must be that for every honest node i, its output log at time $t + 2\text{sec}$ is longer than its log at time t and must contain the block proposed by u.*

Proof. Due to Lemmas 3 and 4, u must successfully propose an epoch-e block and by $t + 1$sec, all honest nodes will see an epoch-e notarized block. Therefore, v enters epoch-$(e + 1)$ or higher by $t + 1$sec. By Lemma 2, v cannot have entered epoch greater than $e + 1$ by time $t + 1$sec. Therefore, at the moment v enters epoch-$(e + 1)$ during $[t, t + 1\text{sec}]$, by Lemma 3, v must successfully propose an epoch-$(e + 1)$ block. It is not difficult to see that this block must extend from an epoch-e block no matter whether v proposes before 1sec elapses into local epoch $e + 1$ or when 1sec has elapsed into local epoch $e + 1$. The remainder of the proof follows from Lemma 4. □

Corollary 1 (Liveness during sufficiently long periods of synchrony)**.** *Except with negligible in κ probability over the choice of execution, it must be that during any period of synchrony denoted $[t_0, t_1]$ that is at least $\omega(\log \kappa)$*min *long, it must be that for any honest node, its output log at time t_1 contains one or more block(s) proposed by honest nodes in comparison with its output log at time t_0.*

Proof. Due to Fact 1 and the honest protocol definition, if an honest node has entered some epoch e at time $t \in [t_0, t_1 - 3\text{sec}]$, then by time $t + 1\text{min} + 2\text{sec}$, all honest nodes must have entered epoch $e + 1$ or higher. Thus if $[t_0, t_1]$ that is at least $\omega(\log \kappa)$min long, there must exist at least $\omega(\log \kappa)$ consecutive epochs (denoted the set E) such that all honest nodes have been in these epochs during $(t_0 + 3\text{sec}, t_1 - 3\text{sec})$. Since for each epoch a random node is chosen as proposer, except with negligible probability, there must exist consecutive epochs e and $e + 1$ in E both having honest proposers. The corollary now holds from Theorem 2 and the fact that there are only negligible fraction of bad executions where exist either signature forgery or hash collision in honest view. □

5 Doubly-Pipelined PaLa

As mentioned, there are two common philosophies for proposer rotation, the *democracy-favoring* approach and the *stability-favoring* approach. In the earlier warmup, we have focused on the former where we rotate proposers for every block. We now describe a novel variant of PaLa that is suitable for the latter scenario, where we wish to stick with the same proposer as long as it is performing well. Specifically, the stability-favoring policy is often chosen for high performance applications — in this scenario, proposers can be dedicated infrastructure nodes with more abundant provisioning (e.g., in terms of bandwidth, CPU, and other resources) than normal consensus nodes.

In the basic PaLa described earlier, consensus nodes must wait to collect notarizations for the entire ancestor chain before voting on the next block. Thus the system is limited by the inverse of network latency due to the sequential nature of the voting algorithm. This is undesirable for performance particularly when the network latency is large but bandwidth is abundant.

Our "doubly-pipelined" PaLa avoids this drawback, and thus is geared towards a stability-favoring proposer-rotation policy. In this doubly-pipelined variant, nodes are allowed to opportunistically vote for blocks even if not all notarizations have been collected for ancestor blocks, as long as the number of unnotarized blocks at the end is bounded by an opportunistic parameter k, where $k = 1$ is the basic scheme in Section 3. For finalization, nodes must now wait for roughly k consecutively notarized blocks.

5.1 Definitions and Notations

Proposers and committee. We consider a generalization where the logical roles of proposers and committee are separate. In this section, we assume that there are n committee members and n' proposers. We can achieve consistency and liveness as long as the adversary controls less than $\frac{1}{3}$ fraction of the committee, and at least 1 proposer is honest.

Aliases for units of time. Defined in the same way as before.

Valid block. A block is now of the form $(\mathsf{seq}, \mathsf{TXs}, h_{-1})$ where $\mathsf{seq} := (e, s)$ is called the sequence number and $e \in \mathbb{N}$ is called the epoch number. The definitions of TXs and h_{-1} are as before.

Blockchain validity. A blockchain chain is said to be valid iff

1. for every $1 \leq i \leq |\mathsf{chain}|$, $\mathsf{chain}[i].h_{-1} = H(\mathsf{chain}[:i-1])$.
2. for every $1 \leq i \leq |\mathsf{chain}|$, each block is either a *normal* or *timeout* block:
   - *Normal block*: it must be that $\mathsf{chain}[i].e = \mathsf{chain}[i-1].e$ and $\mathsf{chain}[i].s = \mathsf{chain}[i-1].s + 1$.
   - *Timeout block*: it must be that $\mathsf{chain}[i].e > \mathsf{chain}[i-1].e$ and $\mathsf{chain}[i].s = 1$.

"Fresher than" relation. We say that chain is fresher than chain' iff $\mathsf{chain}[-1].\mathsf{seq} > \mathsf{chain}'[-1].\mathsf{seq}$. If $\mathsf{chain}[-1].\mathsf{seq} = (e, \_)$ we also say that chain is an epoch-e chain.

Votes and notarization. Defined in the same way as before.

Fully notarized. To avoid ambiguity (that possibly stems from using block as an alias of chain), we say that a chain is *fully notarized* in some node's view if every block in this chain has gained notarization in the node's view.

Proposer eligibility. We consider a simple round-robin policy where in epoch e, the $(e \mod n')$-th proposer is considered eligible.

5.2 Doubly-Pipelined PaLa

We now describe our doubly-pipelined PaLa scheme.

Internal clock synchronization. Each consensus node maintains a local clock that indicates the current epoch (henceforth called the node's *local epoch*). Initially, the local epoch is set to 1. Nodes use the following mechanism to advance their local epochs:

- If a node's freshest fully notarized chain has not included a new epoch-e block since 1min ago and moreover the node has been in epoch e for the past 1min, now sign and multicast $\texttt{clock}(e+1)$ (if it has not already done so).
- Upon observing signed $\texttt{clock}(e)$ messages from at least $\frac{2n}{3}$ distinct nodes, advance local clock to e (if the local epoch is smaller than e).

Proposal. Whenever a node enters a new epoch e, for the first 1sec, do nothing and wait for the network. If the node is an eligible proposer for epoch e, from this point on, the proposer performs the following:

1. first, let chain be the freshest fully notarized chain that the node has observed so far; propose the timeout block $\mathsf{B} := ((e, 1), \mathsf{TXs}, H(\mathsf{chain}))$ where TXs is a set of outstanding transactions;

2. now repeat the following: if the node has proposed blocks at sequence numbers $(e, 1), \ldots, (e, s)$ so far, and if either $s < k$ or all of $(e, 1), \ldots, (e, s - k + 1)$ have been notarized in the node's current view, then propose the next normal block containing the outstanding transactions (extending from the chain ending at (e, s) that it has proposed so far).

Intuitively, the proposer can keep on proposing as long as there are fewer than k proposed but un-notarized blocks at the end belonging to the current epoch.

Voting. Upon hearing a proposal of the form $\mathsf{B} := ((e, s), \mathsf{TXs}, h_{-1})$ with a valid signature from an *eligible* proposer for B, a node votes on the proposed extended chain including B iff the following hold:

1. it is currently in local epoch e and it has not voted for any block with the sequence number (e, s);
2. it must have observed some chain such that B is a valid block extending chain, $H(\mathsf{chain}) = h_{-1}$;
3. all blocks in chain belonging to a different epoch than B must be notarized, and moreover the prefix $\mathsf{chain}[: -k]$ must be fully notarized in the node's view (i.e., including the proposed block itself, at most k blocks at the end can be un-notarized).
4. chain must be at least as fresh as the node's freshest fully notarized chain observed at the beginning of this epoch e.

Finalization. Given chain, let $\mathsf{chain}[: i]$ denote the longest prefix of chain that ends with at least k consecutive normal blocks, $\mathsf{Finalize}(\mathsf{chain})$ would then output $\mathsf{chain}[: i-k]$, i.e., chop off k trailing blocks from $\mathsf{chain}[: i]$. If no such prefix is found in chain, then $\mathsf{Finalize}(\mathsf{chain})$ outputs $\emptyset$.

At any time, a node takes its freshest fully notarized chain denoted chain and outputs $\mathsf{Finalize}(\mathsf{chain})$ if it is longer than what is previously output.

Formal proofs. We defer the formal proofs to Section 7. In fact, there we shall directly prove the variant that supports committee reconfiguration which is a strict generalization of the scheme described in this section.

6 Committee Reconfiguration for Doubly-Pipelined PaLa

We consider an application scenario like ThunderCore, a blockchain company that provides dedicated infrastructure nodes as "accelerators" (i.e., proposers) and committee participation is open to anyone who stakes in on the blockchain. We describe simple but non-trivial techniques to support committee reconfiguration. Throughout this section, we may assume that there is a fixed and publicly-known set of infrastructure nodes that act as proposers; on the other hand, the set of committee members responsible for voting are chosen from the blockchain (e.g., computed from the set of stake-in messages on the blockchain).

6.1 Intuition for $k = 1$

We describe the intuition assuming $k = 1$, i.e., nodes must have the parent's notarization before voting for the next block, and moreover, we chop off 1 normal block for finalization.

Committee switch is tricky due to the following reason. Recall that our consistency proof critically relies on the following intuition: when two consecutive sequence numbers e.g., (e, s) and

$(e, s+1)$ both obtain notarization in honest view, we conclude that many honest nodes must have seen a notarization for (e, s) very quickly, and thus all these honest nodes will insist on the block at (e, s) be included in any future chain. In this way, we can be sure that the notarized block at (e, s) is final. This argument will break if the committee voting for the future chain changes. In this case we want that the future committee will also insist on the notarized block at (e, s) be included.

We propose a surprisingly simple idea to fix the aforementioned issue and support committee reconfiguration. At a very high level, every time a normal block is encountered, the immediate next block's committee can switch by half. As a special case, to fully switch from an old committee C_{old} to a new committee C_{new}, we can complete the switch over two normal blocks, having a hybrid transitional committee consisting of both C_{old} and C_{new} in between.

More generally, henceforth we may assume that every committee consists of two sub-committees denoted (C_0, C_1), and each sub-committee is of size n. We shall assume that fewer than $\frac{1}{3}$ fraction of nodes are corrupt in each of the sub-committees. For a block to be notarized, there must be at least $\frac{2n}{3}$ votes from each of the sub-committees. Now, whenever a normal block is encountered in the blockchain, the next block's committee may switch from (C_0, C_1) to (C_1, C_2) — notice that only one of the two sub-committees may switch. Note that the two sub-committees in a committee may be the same. Therefore, as a special case, if we wish to switch from C_{old} to C_{new}, using this new notation, we can complete the switch from $(C_{\text{old}}, C_{\text{old}})$ to $(C_{\text{old}}, C_{\text{new}})$ to $(C_{\text{new}}, C_{\text{new}})$ over two normal blocks.

Below we generalize this idea to the doubly-pipelined PaLa where k may in general be larger than 1.

6.2 Committee Reconfiguration Scheme

In the earlier Section 6.1, we described the intuition for $k = 1$. There, the committee's composition can change by half after every normal block. This idea can be generalized to $k > 1$ as follows: the committee's composition can change by half after every k consecutive normal blocks. This means that a full switch from an old committee C_{old} to a new committee C_{new} must take place over two groups of consecutive k normal blocks (belonging either to the same epoch or different). We now give a formal description: after the first group (of k-consecutive normal blocks), a hybrid committee containing both C_{old} and C_{new} is in charge; after the second group (of k-consecutive normal blocks), the committee can completely switch to C_{new}.

Committee reconfiguration function. We assume that there is an efficiently computable and deterministic function $\mathsf{Comm}(\cdot)$, and $\mathsf{Comm}(\mathsf{chain}[:i-1])$ outputs the committee for the *next* block $\mathsf{chain}[i]$. Each such committee is the form (C_0, C_1) where each of C_0 and C_1 is a set of n public keys. Henceforth, we assume that *the adversary controls less than $\frac{1}{3}$ fraction of each of C_0 and C_1*. Note that each of C_0 and C_1 must contain n distinct public keys, but they need not be disjoint.

Further, we require that $\mathsf{Comm}(\cdot)$ satisfies the following constraints:

If $\mathsf{Comm}(\mathsf{chain}[:i]) \neq \mathsf{Comm}(\mathsf{chain}[:i-1])$, i.e., the committees for $\mathsf{chain}[i+1]$ and $\mathsf{chain}[i]$ are different, then the following must be true:

1. parse $\mathsf{Comm}(\mathsf{chain}[:i-1])$ as (C_0, C_1), then $\mathsf{Comm}(\mathsf{chain}[:i])$ must be of the form (C_1, C_2) for some C_2 containing n public keys; i.e., the committee's composition can switch by only a half;
2. the previous k blocks $\mathsf{chain}[i-k+1:i]$ must all be normal blocks having the same epoch number; moreover, all these k blocks must have the same committee.

Corruption assumption. Henceforth we assume the following: for any notarized chain denoted chain in honest view, let $(C_0, C_1) := \mathsf{Comm}(\mathsf{chain})$, it must be that the adversary controls less than $\frac{1}{3}$ fraction of each of C_0 and C_1.

Doubly-pipelined PaLa with committee reconfiguration. To support committee reconfiguration, the following modifications are necessary on top of our doubly-pipelined PaLa described earlier in Section 5.

We say that (C_0, C_1) is the committee for chain iff it is the committee for $\mathsf{chain}[-1]$ as computed by the $\mathsf{Comm}(\cdot)$ function.

- *Notarization.* For chain to be notarized, let (C_0, C_1) denote the committee for chain, it must be that at least $\frac{2n}{3}$ nodes from C_0 have voted on chain and at least $\frac{2n}{3}$ nodes from C_1 have voted on chain too. More formally, a collection of signatures is said to be a *valid notarization* for chain if it contains at least $\frac{2n}{3}$ signatures on $H(\mathsf{chain})$ from distinct nodes in C_0 and at least $\frac{2n}{3}$ signatures on $H(\mathsf{chain})$ from distinct nodes in C_1.
- *Clock synchronization.* Clock synchronization is performed as follows where sufficiently many clock messages from any committee defined by any notarized chain would count.
  - if a node has heard some notarized chain chain and let $\mathsf{Comm}(\mathsf{chain}) = (C_0, C_1)$ be its committee; if the node has also heard at least $\frac{2n}{3}$ signed $\texttt{clock}(e)$ messages from distinct nodes of either C_0 or C_1, then advance to epoch e if its current local epoch is smaller;
  - if for the past 1min, a node has been in epoch e but its freshest fully notarized chain has not included any new epoch-e block since 1min ago, sign and multicast $\texttt{clock}(e+1)$.

In the next section, we shall prove our doubly-pipelined PaLa scheme secure including the committee reconfiguration technique.

7 Proofs for Doubly-Pipelined PaLa and Committee Reconfiguration

We directly prove the doubly-pipelined PaLa with committee reconfiguration since it is a strict generalization of the static-committee scheme. Consistency is guaranteed as long as the adversary controls less than $\frac{1}{3}$ fraction of the committee; liveness is guaranteed during sufficiently long periods of synchrony if additionally, at least one proposers is honest.

7.1 Consistency

Lemma 5 (Uniqueness for each sequence number)**.** *Let* chain *and* chain' *be two chains notarized by the same sub-comittee* C *in the honest view of a good execution such that* $\mathsf{chain.seq} = \mathsf{chain}'\mathsf{.seq}$*, it must be that* $\mathsf{chain} = \mathsf{chain}'$*.*

Proof. Suppose two different chain and chain' have the same seq, and both chains obtain notarization in honest view. Assuming that there is no hash collision between the two chains, the total number of distinct signatures for $H(\mathsf{chain})$ and those for $H(\mathsf{chain}')$ must be at least $\frac{4n}{3}$. However, recall that any honest node casts no more than one vote for each sequence number seq whereas corrupt nodes can vote on both $H(\mathsf{chain})$ and $H(\mathsf{chain}')$. Therefore, total number of distinct signatures for $H(\mathsf{chain})$ and those for $H(\mathsf{chain}')$ cannot exceed $n_h + 2n_c = n + n_c < \frac{4n}{3}$, where n_h and n_c denote the number of honest and corrupt nodes respectively. Thus, we reach a contradiction. □

Theorem 3. *Consider the doubly-pipelined variant of the protocol with committee reconfiguration. Let* chain *and* chain' *be two fully notarized chains in honest view in a good execution, it must hold that either* $\mathsf{Finalize}(\mathsf{chain}') \preceq \mathsf{Finalize}(\mathsf{chain})$ *or* $\mathsf{Finalize}(\mathsf{chain}) \preceq \mathsf{Finalize}(\mathsf{chain}')$.

Proof. For contradiction's sake, suppose $\mathsf{Finalize}(\mathsf{chain})$ and $\mathsf{Finalize}(\mathsf{chain}')$ are not prefix of each other; let B_0 be their last common block and i_0 be the index such that $\mathsf{B}_0 = \mathsf{chain}[i_0] = \mathsf{chain}'[i_0]$, which has epoch number e_0.

For each of chain and chain', we define the notions of *stabilized* and *stabilizing* blocks. We illustrate these notions with chain, and the blocks associated with chain' are defined in the same manner.

- *Stabilizing Block.* If all the $k+1$ blocks in $\mathsf{chain}[i_0+1 : i_0+k+1]$ have the same epoch number, then the stabilizing block B_2 in chain is $\mathsf{chain}[i_0+k+1]$. Otherwise, suppose e is the smallest epoch strictly larger than e_0 such that there is a block in chain with sequence number $(e, k+1)$; then, in this case, the stabilizing block B_2 in chain is the block with sequence number $(e, k+1)$. One can check that this is well-defined due to the definition of $\mathsf{Finalize}$.
- *Stabilized Block.* If the stabilizing block $\mathsf{B}_2 = \mathsf{chain}[i_2]$ for some index i_2, then the stabilized block in chain is defined as $\mathsf{B}_1 := \mathsf{chain}[i_2 - k]$.

Observe that B_1 and B_2 have the same epoch number, and they are both after B_0 in chain. We define the blocks B'_1 and B'_2 in the same manner for chain'.

By the definition of $\mathsf{Comm}(\cdot)$, both $\mathsf{chain}[i_0+1]$ and $\mathsf{chain}'[i_0+1]$ have the same committee pair (C_0, C_1). For the case that $\mathsf{chain}[i_0+1]$ is a normal block, each block from $\mathsf{chain}[i_0+1]$ to the stabilizing block B_2 shares at least one sub-committee in (C_0, C_1); for the case that $\mathsf{chain}[i_0+1]$ is a timeout block, each block from $\mathsf{chain}[i_0+1]$ to B_2 has exactly the committee pair (C_0, C_1). This also holds analogously for chain'.

We next derive a contradiction by case analysis. First, observe that if both $\mathsf{chain}[i_0+1]$ and $\mathsf{chain}'[i_0+1]$ are normal blocks, then they have the same sequence number; but this contradicts Lemma 5, because both blocks have the same (C_0, C_1). Hence, for the rest of the proof, we can assume that at least one $\mathsf{chain}[i_0+1]$ and $\mathsf{chain}'[i_0+1]$ is a timeout block, which implies that there is some common sub-committee C shared by all blocks from $\mathsf{chain}[i_0+1]$ to the stabilizing block B_2 and all blocks from $\mathsf{chain}'[i_0+1]$ to B'_2.

Next, define $(e_0, s_0) := \mathsf{B}_0.\mathsf{seq}$, $(e, s) := \mathsf{B}_2.\mathsf{seq}$ and $(e', s') := \mathsf{B}'_2.\mathsf{seq}$; without loss of generality, assume $e \leq e'$.

Case 1: $e = e'$. The sub-case $e_0 = e = e'$ implies that both $\mathsf{chain}[i_0+1]$ and $\mathsf{chain}'[i_0+1]$ are normal blocks, which we have already shown impossible. The case $e_0 < e = e'$ means that both chains have timeout blocks B and B' after B_0 with sequence number $(e, 1)$. Since these blocks share the sub-committee C, this contradicts Lemma 5.

Case 2: $e < e'$. We further divide into two cases: (i) $e_0 = e$, and (ii) $e_0 < e$.

(i) Consider the case $e_0 = e$. It must be the case that in chain, the normal blocks $\mathsf{B}_1 = \mathsf{chain}[i_0+1]$ and $\mathsf{B}_2 = \mathsf{chain}[i_0+k+1]$ have sequence numbers (e_0, s_0+1) and (e_0, s_0+k+1), respectively.

Furthermore, since B_2 is notarized by the sub-committee C, there must have been a set $S \subseteq C$ of more than $\frac{n}{3}$ honest nodes, each of which must have notarized B_2 in its local epoch $e = e_0$. This means that during its local epoch e_0, such an honest node in S must have seen the full notarization of the prefix of chain up to B_1.

On the other hand, we can deduce that the block $\mathsf{B}'_* = \mathsf{chain}'[i_0 + 1]$ immediately following B_0 in chain' must be a timeout block with sequence number $(e'', 1)$, for some $e'' > e_0$.

Since the local epoch numbers of a node does not decrease as time passes, at the beginning of its local epoch e'', every honest node in S must have seen the fully notarized prefix of chain up to B_1, which is strictly fresher than the prefix up to B_0. Hence, it follows that less than $\frac{2n}{3}$ nodes in the sub-committee C could have voted for the block B'_*.

(ii) Consider the case $e_0 < e$. In this case, the stabilized block B_1 must be a timeout block with sequence number $(e, 1)$ and B_2 has sequence number $(e, k + 1)$. Since B_2 is notarized by the sub-committee C, there must be a subset $S \subseteq C$ of more than $\frac{n}{3}$ honest nodes, each of which has seen the fully notarized prefix of chain up to B_1 during its local epoch e.

Observe that $e < e'$ implies that chain' must have some timeout block from $\mathsf{chain}[i_0 + 1]$ to B'_1 whose epoch number is strictly larger than e. Hence, consider the timeout block B'_* in chain' with sequence number $(e'', 1)$ such that e'' is the minimum epoch larger than e.

Again, we can conclude that at the beginning of its local epoch e'', each honest node in S must have seen the fully notarized prefix of chain up to B_1, which is strictly fresher than the prefix of chain' excluding blocks from B'_* onwards; this is because the block preceding B'_* cannot have epoch number e (and must have an epoch number strictly smaller than e), because of Lemma 1. Hence, it follows that less than $\frac{2n}{3}$ nodes in the sub-committee C could have voted for the block B'_* in chain'.

This completes the proof of the lemma.

□

7.2 Liveness

First, it is not hard to see that Fact 1 of Section 4.2 still holds.

Lemma 6. *Consider a good execution and let $[t_0, t_1]$ denote a period of synchrony. Suppose that at some time $t \in (t_0 + 2\text{sec}, t_1]$, some honest node is at most 2sec into its local epoch e. Then, at time t, then no honest node is in local epoch e' where $e' > e$.*

Proof. Suppose that at some time $t \in (t_0 + 2\text{sec}, t_1]$, some honest node is at most 2sec into its local epoch e. Then, every honest node must have entered epoch e by t but cannot be more than 3sec into its local epoch e due to Fact 1. Thus no honest node will have sent $\texttt{clock}(e+1)$ or higher. □

Lemma 7 (Honest proposer can successfully propose)**.** *Consider a period of synchrony denoted $[t_0, t_1]$ in a good execution and let t be some time that belongs to the period of synchrony. If an honest proposer of epoch e first enters local epoch $(e, 1)$ at $t \in (t_0 + 1\text{sec}, t_1 - 1\text{sec}]$, then the timeout block it proposes must be a valid block extending from the parent.*

Proof. By Lemma 6, by the time the honest proposer of epoch e (henceforth denoted u) proposes, no honest node can be in epoch $e' > e$ or higher, and thus there cannot be any epoch-e' block notarized in honest view yet for $e' > e$; and obviously there cannot be any epoch-e block notarized in honest view yet either before u proposes any epoch-e block. Thus, the freshest fully notarized parent chain the proposed timeout block extends from must have a sequence number that is less fresh than $(e, 1)$. □

Lemma 8 (Progress in periods of synchrony)**.** *Consider a period of synchrony denoted $[t_0, t_1]$ in a good execution. Suppose that an honest node u is the proposer of epoch e and it first entered epoch e at some time $r \in (t_0 + 2\text{sec}, t_1 - 2\text{sec})$. Suppose u proposes an epoch-e block at some time t where $r \leq t < t_1 - 1\text{sec}$. Then, by time $t + 1\text{sec}$,*

(a) every honest node will have seen a notarization for the said proposed block;

(b) every honest node must still be in epoch e;

(c) u must have proposed more epoch-e blocks.

Proof. We prove by induction. Let (e, s) denote the sequence number of the block B proposed. For both the base case and induction step, Claim (c) is directly implied by Claims (a) and (b). We thus focus on proving these two claims for the base case and induction step.

Base case: $s \leq k$. As soon as u first enters epoch e (let r be this time), it waits for 1sec and then proposes blocks at $(e, 1), \ldots, (e, k)$. Claim (b) holds by Lemma 6.

We now prove Claim (a). Due to Fact 1 and Lemma 6, at time $r + \Delta$, all honest nodes must be in epoch e and they must have received all these proposals at $(e, 1), \ldots, (e, k)$.

To show that every honest node will receive a notarization for each of these blocks by $r + 1$sec, it suffices to show that the parent chain the block at $(e, 1)$ extends from is at least as fresh as any honest node's freshest notarized chain when it first enters local epoch e. This can be argued in a similar manner as that of Lemma 4.

Induction step. Suppose that the lemma holds for all $s \leq s'$ for some $s' \geq k$. We now show that the lemma holds for $s^* = s' + 1$ too. Let t be the time the block at sequence number (e, s^*) is proposed by u where $r \leq t < t_1 - 1$sec. All honest nodes must have received the proposal by $t + \Delta$.

We first prove Claim (a). We need to argue that all honest nodes will vote for it upon receiving this proposal. To show this, it suffices to argue that for any honest node i, when i receives this proposal, it cannot be in epoch $e' > e$ yet.

Fact 2. *For any $t^* \leq t + 1$sec, for any honest node j, it must be that either node j's freshest fully notarized sequence at time t^* is fresher than its freshest fully notarized sequence at time $t^* - 2$sec, or node u has not been in epoch-e for more than 2sec at t^*.*

Proof. Henceforth assume that u has been in epoch-e for more than 2sec at t^*; otherwise the claim trivially holds.

Suppose for some j, its maximal fully notarized sequence ends at (e', s') at time $t^* - 2$sec. By induction hypothesis note that t cannot be more than 1sec since the last time u proposed blocks. Thus it must be that $e' \leq e$.

Note that by time $t^* - 2\text{sec} + \Delta$, u must have seen a fully notarized sequence ending at (e', s'). Thus if $e' = e$, u must have proposed blocks at $(e, s'+1), \ldots, (e, s'+k)$ by time $t^* - 2\text{sec} + \Delta$; if $e' < e$, u must have proposed $(e, 1), \ldots, (e, k)$ by time $t^* - 2\text{sec} + \Delta$. Due to the induction hypothesis, in another 1sec amount of time every honest node (including j) will have observed notarizations for these proposed blocks (and by induction hypothesis it must have observed notarization for earlier blocks of this epoch too). □

Due to the above fact, and Lemma 6, by $t + 1$sec, no honest node will be in epoch $e' > e$. Thus by $t + \Delta$ all honest nodes must still be in epoch e; and therefore they will vote on the proposal at (e, s^*). Note that the above proof directly implies Claim (b) too.

□

Recall that we use n' to denote the number of proposers.

Theorem 4 (Liveness for doubly-pipelined PaLa)**.** *Except with negligible in κ probability over the choice of the execution, the following must hold: for any period of synchrony denoted $[t_0, t_1]$ that is at least $3kn'$min long, it must be that for any honest node, its output log at time t_1 must be longer than its output log at time t_0.*

Proof. We first prove the following claim.

Claim 1. *Let $\alpha = 3k$. If an honest node has entered some epoch e at time $t \in [t_0, t_1 - \alpha\text{min}]$, then by time $t + \alpha\text{min} + 2\text{sec}$, either every honest node has seen $2k$ new notarized epoch-e blocks in its freshest fully notarized chain (in comparison with at time t), or every honest node has advanced to epoch $e + 1$ or higher.*

Proof. Straightforward from the following aspect of the honest protocol and Fact 1: if an honest node ever remains in epoch e for more than 1min without seeing a new epoch-e block included in its freshest fully notarized chain, then the node must send a `clock`$(e + 1)$ message. □

Thus, for every αmin window $[r, r'] \subseteq (t_0 + 3\text{sec}, t_1 - 3\text{sec})$, either all honest nodes have advanced to a later epoch compared to the slowest honest node (in terms of epoch) at r, or each of them has seen $2k$ new notarized blocks epoch-e blocks in its freshest notarized chain at time r' in comparison with at time r. In the latter case, the honest node's finalized log must have grown in between.

In the former case, if epoch $e + 1$ has an honest proposer, then all honest nodes' logs will have grown in 3sec by Lemma 8. Otherwise, the adversary can prevent an honest node's output log from growing for at most αmin more.

The remainder of the proof follows from the fact that at least one among the n' proposer is honest and that the proposer rotation is round-robin. □

Remark 3. In the above liveness theorem, we proved that honest nodes' output logs must grow in sufficiently long periods of synchrony (except with negligible probability), but we did not show that they must grow by honest blocks (i.e., blocks proposed by honest nodes). This is because in our current scheme, a corrupt proposer can stay as the proposer as long as it continues to propose (possibly bad-quality) blocks. Later in Section 8, we describe how to avoid this drawback with a simple technique. Note also that this is not an issue with the basic PaLa in Section 3 that implements a democracy-favoring proposer election policy.

8 Practical Considerations for Doubly-Pipelined PaLa

8.1 Ensuring Block Quality

Our earlier liveness theorem (Theorem 4) for the doubly-pipelined PaLa guarantees the growth of finalized chains for honest nodes during sufficiently long periods of synchrony, but states no guarantee about the contents of the newly finalized blocks (see also Remark 3). In particular, if the newly finalized blocks are proposed by corrupt proposers, they can censor certain transactions. This issue can be remedied by introducing an additional check during voting: a consensus node only votes for a proposal if the proposal does not leave out transactions that have been outstanding for a long time. In this way, malicious proposals that try to censor long-standing transactions will not gain notarization.

8.2 Practical Optimizations for Dedicated-Infrastructure Proposers

As mentioned, in high-performance deployments, it makes sense to separate the proposer role from the voter role and incentivize proposer nodes to be better provisioned in terms of compute and bandwidth resources. In this case we may refer to proposers as *accelerators* and refer to voters as *committee nodes*.

In particular, due to practical considerations, it makes sense to provision much more bandwidth for accelerators and have committee nodes route messages through accelerators — in this way, committee nodes need not implement a peer-to-peer diffusion network.

The idea is the following. Imagine that there are m accelerators running in parallel. At any time, the *primary* accelerator is in charge of proposing blocks, gathering notarization, and sending notarizations to all other nodes (including committee nodes and other accelerators). All accelerators, including the *primary* and *standby* ones, relay `clock` messages among all nodes — we assume committee nodes inform every accelerator of every `clock` message they have heard. Note that as long as one of the accelerators is honest, then during a period of synchrony, if some honest node observes some `clock` message at time t, all honest nodes will have observed it by $t + 2\Delta$.

Note that the "strong period of synchrony" assumption now holds only for `clock` messages (since they are relayed through all accelerators), but not for other messages; however the standard "period of synchrony" assumption still holds (Definition 1). Therefore, the following change to the protocol is necessary: when an epoch-e accelerator enters local epoch e, instead of skipping the first 1sec, it actually uses the first 1sec to perform reconciliation with all nodes, to make sure that it obtains from everyone the freshest fully notarized chain they saw at the beginning of their respective local epoch e.

Finally, we can also use an aggregate signature scheme such as BLS [6] such that the accelerator can compress the multiple signatures collected before sending them out to nodes. This approach has been used in several existing works [4, 11].

9 Conclusion

We have described PaLa, a simple blockchain protocol that is arbitrarily partition tolerant and resists any adversary that controls less than $\frac{1}{3}$ fraction of corrupt nodes. We described a novel technique for performing committee rotation with PaLa, and a doubly-pipelined variant suitable for a performance demanding scenario with a stability-favoring proposer rotation policy.

A variant of PaLa is being considered as a main-net candidate for ThunderCore. ThunderCore is building a fast, EVM-compatible cryptocurrency using a practical variant of Thunderella paradigm [14]. In this paradigm, a committee of nodes perform voting on a fast path to confirm transactions faciliated by an accelerator. When the fast path stops functioning, a slowchain is used to diagnose and resolve the problem. ThunderCore is considering adopting a variant of PaLa to make it possible to switch accelerators on the fast path in most circumstances (such that falling back to the slowchain is only necessary in extraordinary circumstances).

Acknowledgments

We acknowledge helpful discussions with Bobby Bhattacharjee. We are grateful to the brilliant engineering team at ThunderCore who inspired us to work on this problem. We are grateful to Lorenzo Alvisi and Robbert van Renesse for being supportive.

References

[1] http://coinmarketcap.com/.

[2] Dpos consensus algorithm - the missing white paper. https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper.

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

[4] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

[5] Ittai Abraham, Guy Gueta, Dahlia Malkhi, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance: Thelma, velma, and zelma. *CoRR*, abs/1801.10022, 2018.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004.

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

[8] Jing Chen and Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

[9] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. http://eprint.iacr.org/2017/573.

[10] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

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

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

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

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

[15] Virgil Griffith Vitalik Buterin. Casper the friendly finality gadget. https://arxiv.org/abs/1710.09437.

A Other Extensions and Practical Considerations

In this section, we describe additional extensions and practical considerations, mostly on top of the scheme in Section 3 that implement a democracy-favoring proposer rotation policy.

A.1 Security under Adaptive Corruptions

So far in the paper, we considered the static corruption model where the adversary announces the corrupt nodes upfront. We now discuss how to extend our security guarantees to an adaptive adversary.

For a stability-favoring policy (e.g., Section 5) where we stick to the same proposer until liveness is hampered, since an honest proposer is responsible for proposing multiple blocks in a row, an adaptive adversary can always corrupt the proposer, and it can continue doing so for $\Theta(n)$ proposers. Therefore, in the worse case, liveness may require $\Theta(n)$ minutes even during periods of synchrony if consecutive $\Theta(n)$ proposers are adaptively corrupt.

For the democracy-favoring approach (e.g., Section 3) where we rotate proposer every block, we suggest the following variation that allows us to asymptotically preserve the same liveness parameter as in Theorem 2 even in the presence of an adaptive adversary. The idea is inspired by several recent works [8, 9, 11, 13]. Instead of adopting a hash function H to determine eligibility as a proposer, we rely on a Verifiable Random Function (VRF) instead. Further, a proposer would use a *forward-secure* signing scheme to sign the proposal and evolve the signing key (erasing the old key in the process) immediately after the proposal is sent out.

More concretely, a VRF ensures that the node itself can use a secret key to evaluate a pseudo-random outcome, which determines its eligibility as a proposer. No one else other than the node itself can perform this eligibility determinination. However, if elected, the node can demonstrate a proof such that anyone with the proof can verify that the node is indeed elected as a proposer for some block. At a high level, the VRF ensures that the adversary cannot predict in advance which node is the proposer for the next block. Of course, the adversary can adaptively corrupt the proposer immediately after it proposes a next block, and at this time it can make the proposer immediately propose a different block (at the same sequence number). To defend against this attack, the idea is to use a forward-secure signing scheme for signing the proposal. Once a proposal is sent out, the node immediately erases this signing key such that if the node is corruptted at this point, it will no longer be able to sign an equivocating proposal at the same sequence number.

A concrete instantiation of a VRF is to hash a unique signature on the sequence number. Let Σ_{unique} denote a unique signature scheme, let H be a random oracle, and let D_p denote an appropriate difficulty parameter. For node i to determine whether it is eligible for proposing a block of epoch e, it checks if the following is true:

$$H(\sigma) < D_p \quad \text{where} \quad \sigma := \Sigma_{\text{unique}}.\mathsf{Sign}_{\mathsf{sk}_i}(e)$$

Everyone knowing σ can verify if node i is indeed eligible for proposing an epoch-e block by checking[3]:

$$H(\sigma) < D_p \quad \text{and} \quad \Sigma_{\text{unique}}.\mathsf{Verify}_{\mathsf{pk}_i}(e, \sigma) = 1$$

Finally, the difficulty parameter D_p should be set such that for every e, in expectation 2 out of n nodes are elected proposer — in this case, for every e, with $O(1)$ probability, there is a unique honest proposer. Note that unlike the liveness proofs in Section 4.2 which assumed that the definition of proposer is unique per epoch, here there may be multiple eligible proposers each epoch. However, if so, we can just treat this epoch as having a corrupt proposer. It is not difficult to adjust the liveness proofs in Section 4.2 to this case, as long as every now and then, we have two consecutive epochs, and in each of these epochs, a unique honest node is an eligible proposer and no corrupt node is eligible.

[3] Although not explicitly expressed earlier in our scheme description, in this case, the proposer must additionally attach the unique signature σ to a proposed block for others to verify its eligibility.