Thunderella: Blockchains with Optimistic Instant Confirmation

Rafael Pass[1(✉)] and Elaine Shi[2]

[1] CornellTech, New York, USA
rafael@cs.cornell.edu
[2] Cornell, Ithaca, USA

Abstract. State machine replication, or "consensus", is a central abstraction for distributed systems where a set of nodes seek to agree on an ever-growing, linearly-ordered log. In this paper, we propose a practical new paradigm called Thunderella for achieving state machine replication by combining a fast, asynchronous path with a (slow) synchronous "fall-back" path (which only gets executed if something goes wrong); as a consequence, we get *simple* state machine replications that essentially are as robust as the best synchronous protocols, yet "optimistically" (if a super majority of the players are honest), the protocol "instantly" confirms transactions.

We provide instantiations of this paradigm in both permissionless (using proof-of-work) and permissioned settings. Most notably, this yields a new blockchain protocol (for the permissionless setting) that remains resilient assuming only that a majority of the computing power is controlled by honest players, yet *optimistically*—if 3/4 of the computing power is controlled by honest players, and a special player called the "accelerator", is honest—transactions are confirmed as fast as the actual message delay in the network. We additionally show the 3/4 optimistic bound is tight for protocols that are resilient assuming only an honest majority.

1 Introduction

State machine replication, also referred to as atomic broadcast, is a core distributed systems abstraction that has been investigated for three decades. In a state machine replication protocol, a set of servers seek to agree on an ever-growing, *linearly-ordered log*, such that two important properties are satisfied: (1) *consistency*, i.e., all servers must have the same view of the log; and (2) *liveness*, i.e., whenever a client submits a transaction, the transaction is incorporated quickly into the log. In this paper, we will also refer to state machine replication as *consensus* for short[1].

The full version of this paper is available at https://eprint.iacr.org/2017/913 [36].

[1] Although the term "consensus" has been used in the distributed systems literature to mean other related abstractions such as single-shot consensus; in this paper, we use "consensus" to specifically refer to "state machine replication".

© International Association for Cryptologic Research 2018
J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10821, pp. 3–33, 2018.
https://doi.org/10.1007/978-3-319-78375-8_1

State machine replication is a fundamental building block for replicated databases. For more than a decade, companies such as Google and Facebook have deployed Paxos-style protocols [5,24,30] to replicate a significant part of their computing infrastructure. These classical deployment scenarios are typically relatively small scale, with fast local-area networking, where crash (rather than byzantine) faults are usually of concern.

Fuelled by decentralized cryptocurrencies, recently the community has been excited about large-scale applications of distributed consensus. Two deployment scenarios are of interest: (1) the *permissionless* setting where anyone can join freely (e.g., decentralized cryptocurrencies); and (2) the *permissioned* setting where only approved participants may join (e.g., a consortium blockchain where multiple banks collaborate to build a distributed ledger). Regardless of which setting, the typical deployment would involve a large number of nodes (e.g., thousands or more) controlled by mutually distrustful individuals and organizations.

Roughly speaking, two broad classes of protocols have been considered for the large-scale setting, each with their own set of deficiencies:

- First, *classical-style protocols* such as PBFT [9] and Byzantine-Paxos [30] confirm transactions quickly in the normal case; but these protocols are notoriously complicated, making implementation, reconfiguration, and maintenance relatively difficult especially in a large-scale setting. Further, these protocols achieve "fast confirmation" by adopting the asynchronous (or partially synchronous) model, and thus inherently they can tolerate at most $\frac{1}{3}$ corruptions [15,38].
- Second, *blockchain-style* protocols, represented by Nakamoto's original blockchain [19,34,35], are a new breakthrough in distributed consensus: these protocols are conceptually simple and tolerate *minority corruptions*. Moreover, it is has been shown how to remove the expensive proof-of-work from blockchain-style consensus [11,26,40] thus solving the energy waste problem. Further, not only has blockchains' robustness been empirically proven, earlier works [11,40] have also shown mathematically that blockchain-style consensus indeed achieves certain robustness properties in the presence of sporadic participation and node churn that none of the classical-style protocols can attain! Unfortunately known blockchain-style protocols suffer from slow transaction confirmation, e.g., Bitcoin's Nakamoto consensus has a 10-minute block interval and it takes several blocks to confirm a transaction with sufficient confidence. Earlier works that mathematically analyze blockchain-style consensus [35,40] have pointed out that such slowness is inherent for blockchain-style protocols since the expected block interval must be set to be sufficiently large for the protocol to retain security.

A natural question that arises is whether there is some way to *simultaneously* reap the benefit of both of these "worlds". Unfortunately, a negative answer was presented by earlier works [38–40] which showed that a natural notion of fast transaction confirmation called "responsiveness" is unattainable against $\frac{1}{3}$

(even static) corruptions in classical or permissionless models. In this paper we consider a new notion called *optimistic responsiveness* that allows us "circumvent" this lower bound such that we can achieve responsiveness most of the time in practice and yet tolerate up to minority corruptions in the worst-case. In our approach, in the *optimistic case* (when e.g., a super majority is honest), we enjoy the fast nature of asynchronous protocols; and yet we retain the resilience of synchronous (e.g., blockchain) protocols as well as their robustness properties (e.g., support for sporadic participation). More precisely, we show how to combine a fast and simple "asynchronous path"—which guarantees consistency but not liveness—with a (slow) synchronous "fall-back" path which only gets executed if something goes wrong.

1.1 The Thunderella Paradigm

To characterize what we mean by "fast" or "instant confirmation", we adopt the same notion of *responsiveness* as proposed in the work by Attiya et al. [1] and later adopted by others [23,38]. A consensus protocol is said to be responsive iff any transaction input to an honest node is confirmed in time that depends only on the *actual network delay*, but not on any a-priori known *upper bound on the network delay*. Henceforth in this paper, we use δ to denote the actual network delay and use Δ to denote an a-priori known upper bound of the network's delay where Δ is possibly provided as input to the protocol.

As shown in [38], achieving responsiveness requires us to assume that 2/3 of the players are honest. (Additionally, all known protocols that are responsive are very complicated, and thus hard to implement.)

Towards overcoming this issue, we here instead consider a notion of **optimistic responsiveness**—where responsiveness is only required to hold whenever some "goodness conditions" are satisfies. More precisely, we consider two sets of conditions:

- worst-case conditions (denoted W) under which the protocol provides worst-case guarantees including consistency and "slow" confirmation (e.g., $W =$ majority honest).
- optimistic-case conditions (denoted $O \subseteq W$) under which the protocol additionally provides responsive confirmation (e.g., $O =$ "more than $\frac{3}{4}$ are honest and online, and some designated player (the "leader") is honest").

Our main result is a paradigm for taking any blockchain protocol (permissioned or permissionless) that satisfies consistency and liveness under conditions W, and transform it into a new protocol that satisfies consistency and liveness under "essentially" the same conditions W (and in many cases, actually the same conditions W), and additionally satisfies optimistic responsiveness under condition O.

The idea in a nutshell. To explain our approach, consider first the following simple protocol:

- We have a designated entity: the leader, or "accelerator".
- Transactions are sent to the leader; the leader signs the transaction (with an increasing sequence number), and sends out the signed transaction to a "committee" of players.
- The committee members "ack" all leader-signed transactions, but at most one per sequence number.
- If a transaction has received more than 3/4 of the committees signatures—we refer to such a transaction as being notarized. Participants, can *directly output* their longest sequence of consecutive (in terms of their sequence numbers) notarized transactions—all those transactions are confirmed.

It is not hard to see that this protocol is consistent under condition W' = "1/2 the committee is honest"); additionally, it satisfies liveness with optimistic responsiveness under condition O = "leader is honest, and 3/4 of the committee is honest". In fact, under these optimistic condition, we only need 2 communication rounds to confirm a transaction! This approach is extremely practical and indeed this protocol is often used in practice—for instance `chain.com` use something very similar as their permissioned blockchain (and manage to handle a very high volume of transactions with fast confirmations).

The problem with this approach, however, is that the protocol does not satisfy liveness (even "slow" liveness) under condition W'. If the leader is cheating (or is simply taken down from the network), the protocol halts. (Indeed, in this case `chain.com` resorts to manually fixing the issue.)

To overcome this problem, we leverage the underlying (slow) blockchain protocol, which satisfies both consistency and liveness under W = "honest majority of players". Roughly speaking, if players notice that transactions are not getting confirmed by the leader/committee, some "evidence" of this is sent to the underlying blockchain. We then enter a "cool-down" period, where committee members stop signing messages from the leader, yet we allow players to broadcast any notarized transactions they have seen so far. The length of the cool-down period is counted in blocks on the underlying blockchain (say κ blocks where κ is a security parameter). Finally, after the cool-down period ends, we can safely enter a "slow period" where transactions only get confirmed in the underlying blockchain. We can next use the blockchain to switch out the leader (if needed) and begin a new epoch of the optimistic protocol.

Let us point out the reason for having a cool-down period: without it, players may disagree on the set of transactions that have been confirmed before entering the "slow mode", and thus may end up with inconsistent views. The cool-down period enables honest players to post all notarized transactions they have seen to the (slow) underlying blockchain, and thus (slowly) reach consistency of this set of transactions; once we have reached this consistent view (at the end of the cool-down), we can finally fully switch over to confirming new transactions on the blockchain.

Collecting evidence of cheating. It only remains to explain how to collect evidence that the leader (and/or committee) is cheating or is simply "unavailable". This turns out to also be simple: if a player notices that his transaction is not getting confirmed by the leader or committee, he can send the transaction to the underlying blockchain. The leader is additionally instructed to confirm all transactions it sees on the blockchain.

Now, if players see some transaction on the blockchain, that has not gotten notarized within a sufficiently long amount of time—counted in blocks in the underlying blockchains (say within n blocks)—they know that the leader/committee must be cheating/unavailable, and thus should enter the cool-down period. (Note that as long as the leader can confirm transactions before n blocks are created on the underlying blockchain, he cannot be "falsely accused"; and, by the security of the underlying blockchains those blocks cannot be created too fast.)

Selecting the committee. So far we have constructed a protocol that satisfies consistency and liveness under conditions $W \cap W'$ (i.e., assuming an honest majority of players, and an honest majority in the committee), and additionally satisfies liveness with optimistic responsiveness under condition O. The question now is how to select the committee. We consider two different approaches:

- **Using all players as the committee:** In a permissioned setting, the simplest approach is to simply use all players as the committee. In this case, $W' = W$ and thus, we trivially have resilience under W. A variant of this approach with improved communication complexity is to subsample a committee among the set of players (for instance, using the approach in [11] which additionally requires a random oracle), and change committees on a regular basis (to ensure adaptive security)—the resulting protocol, however, will only be secure if corruptions are "slow" (to ensure the attacker does not have time to corrupt the whole committee before it gets switched out). If sub-sampling is instead done "secretly" using a VRF and a random oracle (as in [32]), we can also ensure that the resulting protocol is adaptively secure *in a model with erasures*, even with "instantaneous corruption".

  We mention that these approaches may also be used in the permissionless setting if Thunderella is used to construct a crypto currency: then we can use (potentially a sub-sample of) recent "stakeholders" to form a committee.
- **Using "recent miners" as the committee:** A different approach that works in both the permissioned and permissionless setting is to select the committee as the miners of recent blocks (as was done in [38]). We note, however, that to rely on this approach, we need to ensure that the underlying is blockchain is "fair" [37] in the sense that the fraction of honestly mined blocks is close to the fraction of honest players. This is not the case for Nakamoto's original blockchain (see e.g., [17]), but as shown in [37], any blockchain can be turned into a fair one. If we use this approach, the resulting protocol will now be consistent and live under simply the condition W (i.e., honest majority), yet also satisfy optimistic liveness under condition O. (Again, this only gives security under adaptive corruption where corruption is "slow", so the set of recent miners changes sufficiently fast before they can all be corrupted.)

Permissionless Thunderella. For instance, if we apply the second approach (of selecting the committee as the recent miners) to Nakamoto's proof-of-work based blockchain, we get the following theorem:

Theorem 1 (Thunderellafor permissionless environments, informal). *Assume a proof-of-work random oracle. Then, there exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a permissionless environment as long as the adversary wields no more than* $\frac{1}{2} - \epsilon$ *the total online computation power in every round where* ϵ *is an arbitrarily small constant, and moreover it takes a short while for the adversary to adaptively corrupt nodes. Moreover, if more than* $\frac{3}{4}$ *of the online computation power is honest and online, then the protocol achieves responsiveness (after a short non-responsive warmup period) in any "epoch" in which the leader is honest and online.*

Permissioned Thunderella. Similar theorems can be shown for permissioned environments (in e.g., the "sleepy model" of [40], or even just in the "classic" model of Dolev-Strong [14]).

The classical mode is essentially the standard synchronous model adopted by the existing distributed systems and cryptography literature. In this model, all nodes are spawned upfront, and their identities and public keys are provided to the protocol as input; further, crashed nodes are treated as faulty and count towards the corruption budget. In a classical, synchronous network, we show that the classical Dolev-Strong byzantine agreement protocol [14] can be extended to implement Thunderella's underlying "blockchain". In this case, our Thunderella paradigm (where use the first approach to instantiate the committee) gives rise to the following informal theorem:

Theorem 2 (Thunderellafor permissioned, classical environments (informal)). *Assume the existence of a PKI and one-way functions. There exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a classical environment under any* $f < n$ *number of fully adaptive, byzantine corruptions where* n *denotes the total number of nodes; moreover, the protocol achieves responsiveness as long as the leader is honest and moreover* $\lfloor \frac{n+f}{2} + 1 \rfloor$ *nodes are honest.*

The "sleepy" model was recently proposed by Pass and Shi [40] to capture the requirements arising from "sporadic participation" in large-scale, permissioned consensus. Specifically, the sleepy model was in fact inspired by permissionless decentralized cryptocurrencies such as Bitcoin, where nodes may come and go frequently during the protocol, and the protocol should nonetheless guarantee consistency and liveness even for players that join late, and for players who might have had a short outage and woke up later to rejoin the protocol.

The sleepy model is "permissioned" in nature in that the set of approved protocol participants and their public keys are a-priori known and provided to the protocol as input. However, unlike the classical setting, (1) nodes are allowed to be non-participating (i.e., *sleeping*); (2) sleeping nodes are not treated as faulty; and (3) the protocol may not know in advance how many players will

actually show up. In comparison, in a classical setting, non-participating nodes are regarded as having crashed and count towards the corruption budget; and moreover a classical protocol need not guarantee consistency and liveness for nodes that have crashed but wake up later to rejoin.

In such a sleepy model, Pass and Shi [40] show that roughly speaking, we can achieve consensus when the majority of *online* (i.e., non-sleeping) nodes are honest (interestingly, unlike the classical synchronous model, [40] also prove that no state machine replication protocol can tolerate more than $\frac{1}{2}$ corruption (among online nodes).

Our Thunderella paradigm (again using the first approach for selecting the committee) can be instantiated in the sleepy model using the sleepy consensus protocol as the underlying blockchain. This gives rise to the following informal theorem in a sleepy environment (where we assume that the adversary can adaptively put honest nodes to sleep).

Theorem 3 (Thunderella for permissioned, sleepy environments (informal)). *Assume the existence of a PKI, enhanced trapdoor permutations, and a common reference string (CRS). There exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a sleepy environment with static corruptions, as long as $\frac{1}{2} - \epsilon$ of the online nodes are honest in every round for any arbitrarily small constant ϵ; moreover, if more than $\frac{3}{4}$ fraction of nodes are honest and online, the protocol achieves responsiveness (after a short non-responsive warmup period) in any epoch in which leader is honest and online.*

In fact, the above theorem also extends to adaptive corruptions *with erasures* using the adaptively secure version of sleepy consensus [40][2] as Thunderella's underlying blockchain, assuming the existence of a VRF and a random oracle (using the approach from [32]).

Lower bounds on the optimistic honest threshold. We additionally prove that our optimistic bound of 3/4 is tight: no protocol that is (worst-case) resilient for simply an honest majority, can be optimistically responsive when more than 1/4 of the player can be corrupted.

Practical Considerations: Instant Confirmation and Scalability. The *low latency* and *poor scalability* of Nakamoto's blockchain protocol are typically viewed as the main bottlenecks for Bitcoin as well as other cryptocurrencies.

Our paradigm provides a very practical and simple approach for overcoming these issue. The Thunderella paradigm shows how to build on top of currently running blockchains, to enable "optimistic instant confirmation" of transactions. Additionally, note that in our protocol, players only need to send transactions

[2] The paper has multiple adaptively secure versions; here we rely on the one that achieves adaptive security with erasures in the random oracle model (as this protocol has better parameters than the one which satisfies adaptive security without a random oracle).

to the leader, who in turn lead the committee to confirm the transaction. Most notably, the underlying blockchain is essentially only used when something goes wrong, and blocks need not be distributed to the whole network before getting confirmed; thus, Thunderella also solves the scalability issue with Nakamoto's blockchain protocol. Of course, both of these guarantees are only "optimistic"—but arguably, under normal circumstances one would expect 3/4 of the players to act honestly, and the leader could be incentivized (paid) to perform its job (and if it doesn't, will be kicked out). Thus, we believe our approach is a practically viable approach for circumventing the main bottlenecks of today's cryptocurrencies.

Comparison. At the surface, our idea is reminiscient of classical-style protocols such as PBFT and Byzantine-Paxos. In particular, protocols like PBFT also have a very simple normal path that consists of $O(1)$ rounds of voting. However, when the normal path gets stuck, PBFT-style protocols fall back to a "view change" mechanism that is also responsive—and thus these protocols tolerate only $\frac{1}{3}$ corruptions in the worst-case, and are invariably complex due to the need to handle asynchrony. (Furthermore, this approach is not amenable for protocols in the permissionless setting). Our key insight is to instead fall back to a synchronous path in the worst case, thus allowing us to circumvent the $\frac{1}{3}$ lower bound for partial synchrony and yet still be responsive in practice most of the time. Moreover, since our protocol is fundamentally synchronous, we benefit from the simplicity and robustness enjoyed by synchronous protocols (e.g., blockchains).

Interestingly, Thunderella is also a constant factor faster in the fast path than most PBFT- or Paxos-style protocols. PBFT-style protocols typically require multiple rounds of voting even in the normal path (*c.f.* Thunderella has exactly one)—and the latter rounds are necessary to prepare for the possibility of a view change. Although it is possible to compress the normal path to a single round of voting, this is typically achieved either by sacrificing resilience (e.g., tolerating only $\frac{1}{5}$ corruptions) [42] or by adding yet another optimistic layer on top of the normal path—thus further complicating the already complex protocol [29].

Roadmap. In this extended abstract, we simply provide a description and proof of the general Thunderella paradigm (informally described above) assuming the existence of a fixed committee, a majority of which is honest. We defer the formal treatment of how to select the committee to the online full version [36] (although we described it informally above).

2 Definitions

We present informal definitions in this section while deferring the detailed formal definitions to the online full version [36].

We adopt the standard Interactive Turing Machines (ITM) approach to model protocol execution. A protocol refers to an algorithm for a set of interactive Turing Machines (also called nodes) to interact with each other.

The execution of a protocol Π that is directed by an environment $\mathcal{Z}(1^\kappa)$ (where κ is a security parameter), which activates a number of nodes as either *honest* or *corrupt* nodes. Honest nodes would faithfully follow the protocol's prescription, whereas corrupt nodes are controlled by an adversary $\mathcal{A}$ which reads all their inputs/message and sets their outputs/messages to be sent.

A protocol's execution proceeds in *rounds* that model atomic time steps. Henceforth, we use the terms *round* and *time* interchangeably. At the beginning of every round, honest and online nodes receive inputs from an environment $\mathcal{Z}$; at the end of every round, honest and online nodes send outputs to the environment $\mathcal{Z}$.

Corruption model. In the standard distributed systems or cryptography literature, crashed nodes are often treated as faulty and count towards the corruption budget. In this paper, we describe a more general model in which we distinguish *crashed* nodes (also referred to as *sleeping* nodes) and *corrupt* nodes. An honest node may have a short-term or long-term outage during which it is not able to participate in the protocol. However, such a crashed node is not in the control of the adversary—in this case we do not attribute this node as corrupt. Informally, we often refer to the set of honest nodes that have not crashed as being *online*. We also consider all corrupt nodes as being online (since this gives the adversary more advantage).

We stress that the motivation for not treating crashed nodes as corrupt is to allow us to prove a more powerful theorem: our Thunderella paradigm ensures consistency and worst-case liveness when α fraction of the committee are honest but not necessarily online (and assuming that the underlying blockchain is secure). In particular, as we noted, α can be as small as a single member of the committee—but in this case the conditions necessary for instant confirmation are somewhat more stringent (i.e., all committee members must be honest and online for instant confirmation). In a more traditional model where crash is treated as corrupt, all of our theorems still apply—except that "honest" would equate to "honest and online".

More formally, in our model, the environment $\mathcal{Z}$ controls when nodes are spawned, corrupted, put to sleep, or waken up:

- At any time during the protocol execution, the environment $\mathcal{Z}$ can spawn new nodes, and newly spawned nodes can either be *honest* or *corrupt*. The adversary $\mathcal{A}$ has full control of all corrupt nodes.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `corrupt` instruction to an honest (and possibly sleeping) node. When this happens, its internal states are exposed to $\mathcal{A}$ and $\mathcal{A}$ henceforth controls the node.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `sleep` instruction to an honest node. When this happens, the node immediately becomes asleep (or sleeping), and it stops sending and receiving protocol messages and performing any computation. Sleeping is similar to the notion of a crash fault in the classical distributed systems terminology. In our paper, though, we treat sleeping nodes as being honest rather than attributing them towards the faulty budget.

- At any time during the protocol execution, $\mathcal{Z}$ can issue a `wake` instruction to an honest, sleeping node. At this point, this node immediately wakes up and continues to participate in the protocol. When an honest, sleeping node wakes up, pending messages that the node should have received while sleeping and additionally some adversarialy inserted messages may be delivered to the waking node.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `kill` instruction to a corrupt node. At this point, the corrupt node is removed from the protocol execution and is no longer considered as an online node—but note that the adversary $\mathcal{A}$ still knows the internal states of a killed node prior to its being killed.

Formally, we use the terminology *online* nodes to refer to the set of nodes that are *(i)* either honest and not sleeping; or *(ii)* corrupt but not having been killed.

Communication model. We assume that honest and online nodes can send messages to all other honest and online nodes. The adversary $\mathcal{A}$ is in charge of scheduling message delivery. $\mathcal{A}$ cannot modify the contents of messages broadcast by honest nodes, but it can reorder and delay messages sent by honest and online nodes, possibly subject to constraints on the maximum delays to be defined later. The adversary $\mathcal{A}$ is allowed to send messages to a subset of honest and online nodes but not all of them. The identity of the sender is not known to the recipient[3].

Formally, we say that $(\mathcal{A}, \mathcal{Z})$ *respects* Δ*-bounded delay* iff $\mathcal{Z}$ inputs Δ to all honest nodes when they are spawned, and moreover the following holds:

> Δ*-bounded delay.* Suppose an honest (and online) node sends a message at time t, then in any round $r \geq t + \Delta$, any honest node that is online in round r will have received the message, including nodes that may possibly have been sleeping but just woke up in round r, as well as nodes which may have just been spawned at the beginning of round r.

Throughout this paper, we assume that $\mathcal{Z}$ inputs the maximum delay parameter Δ to all honest nodes upon spawning (as noted in the above definition of Δ-bounded delay)—this means that the protocol has a-priori knowledge of an upper bound on the network's maximum delay. This is akin to the synchronous communication model in the classical distributed systems literature.

2.1 Classical, Sleepy, and Permissionless Models

The above generic model does not impose any constraints on when nodes are spawned, how many nodes are spawned, and which nodes are allowed to join the protocol. Thus the generic model can capture *permissionless* executions.

[3] Later in the paper, for instantiations in the permissioned model under a PKI, authenticated channels are implied by the PKI.

In this generic model, we can also model *classical* and *sleepy* executions by imposing *constraints* on $(\mathcal{A}, \mathcal{Z})$. The classical setting is what the vast majority of distributed systems literature focuses on. In a classical execution, $(\mathcal{A}, \mathcal{Z})$ is required to spawn all nodes, numbered $1..n$, upfront; further, honest nodes are assumed to be always online (i.e., $(\mathcal{A}, \mathcal{Z})$) are not allowed to issue `sleep` or `wake` instructions. The *sleepy* model was first proposed by Pass and Shi [40], which is meant to be "in-between" a fully permissionless and a classical permissioned model. In a sleepy execution, the set of allowed players are determined upfront and number $1..n$; however, nodes can join late, they can also fall asleep and later wake up again. Nodes that fall asleep are not treated as corrupt and when they wake up, the security properties we define such as consistency and liveness must ensue for them.

We use the terms (n, ρ, Δ)-permissionless environment, (n, ρ, Δ)-sleepy environment, or (n, f, Δ)-classical environment to capture the execution environment we care about and the parameters respected by $(\mathcal{A}, \mathcal{Z})$ (where ρ is a corruption fraction but f is the absolute number of corrupt nodes). We defer the formal definition of these terms to the online full version [36].

2.2 State Machine Replication

State machine replication has been a central abstraction in the 30 years of distributed systems literature. In a state machine replication protocol, a set of nodes seek to agree on an ever-growing log over time. We require two critical security properties: (1) *consistency*, i.e., all honest nodes' logs agree with each other although some nodes may progress faster than others; (2) *liveness*, i.e., transactions received by honest nodes as input get confirmed in all honest nodes' logs quickly. We now define what it formally means for a protocol to realize a "state machine replication" abstraction.

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

Security definitions. Let $T_{\text{confirm}}(\kappa, n, \rho, \Delta, \delta)$ and $T_{\text{warmup}}(\kappa, n, \rho, \Delta, \delta)$ be polynomial functions in the security parameter κ and possibly other parameters of the view such as the number of nodes n, the corrupt fraction ρ, the actual maximum network delay δ, the network delay upper bound Δ that is provided by $\mathcal{Z}$ to the protocol as input, etc.

Definition 1 (Security of a state machine replication protocol). *We say that a state machine replication protocol Π satisfies consistency (or $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness resp.) w.r.t. some $(\mathcal{A}, \mathcal{Z})$, 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 $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, consistency (or $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness resp.) is satisfied:*

- *Consistency*: A view satisfies consistency iff the following holds:
  - *Common prefix.* Suppose that in view, an honest node i outputs LOG to $\mathcal{Z}$ at time t, and an honest node j outputs LOG' to $\mathcal{Z}$ at time t' (i and j may be the same or different), it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that in view, a node i is honest and online at time t and $t' \geq t$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.
- *Liveness*: A view satisfies $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness iff the following holds: if in some round $T_{\text{warmup}} < t \leq |\mathsf{view}| - T_{\text{confirm}}$, some node honest and online in round t either received from $\mathcal{Z}$ an input set txs that contains some transaction m or has m in its output log to $\mathcal{Z}$ in round t, then, for any node i honest and online at any time $t' \geq t + T_{\text{confirm}}$, let LOG be the output of node i at time t', it holds that $\mathsf{m} \in \mathsf{LOG}$.
  Intuitively, liveness says that transactions input to an honest node get included in honest nodes' LOGs within T_{confirm} time; and further, if a transaction appears in some honest node's LOG, it will appear in every honest node's LOG within T_{confirm} time.

2.3 Abstract Blockchain Protocols

A blockchain protocol can be regarded as a way to realize state machine replication. We now formally define what it means for a protocol to realize to a blockchain abstraction. In our paper, our end goal is to realize state machine replication and we leverage an abstract blockchain as an underlying building block. We note that while the blockchain abstraction may superficially resemble that of state machine replication, the blockchain abstraction in fact allows us to additionally express (1) a rough notion of time through chain growth; and (2) fairness properties [37] through chain quality.

Syntax and Security Definitions

Syntax. An abstract blockchain protocol satisfies the following syntax. In each round, every node that is honest and online in this round receives from $\mathcal{Z}$ a set of transactions txs at the beginning of the round; and outputs to $\mathcal{Z}$ an abstract blockchain chain at the end of the round. An abstract blockchain denoted chain is an ordered sequence of blocks of the following format:

$$\mathsf{chain} := \{\mathsf{txs}_i\}_{i \in [|\mathsf{chain}|]}$$

where each txs_i is an application-specific payload such as a set of transactions.

Blockchain notations. We use the notation chain to denote an abstract blockchain. The notation $\mathsf{chain}[:-\ell]$ denotes the entire chain except the trailing ℓ blocks; $\mathsf{chain}[:\ell]$ denotes the entire chain upto the block at length ℓ; $\mathsf{chain}[-\ell:]$ denotes the trailing ℓ blocks of chain; and $\mathsf{chain}[\ell:]$ denotes all blocks at length ℓ or greater.

Henceforth we say that chain is "*an honest chain in* view", iff chain is some honest (and online) node's output to the environment $\mathcal{Z}$ in some round in view. We use the notation $\mathsf{chain}_i^t(\mathsf{view})$ to denote node i's chain in round t in view—since the context is clear, we often omit writing the view explicitly in the above notation.

Security definitions. A blockchain protocol should satisfy chain growth, chain quality, and consistency. Intuitively, chain growth requires that honest nodes' blockchains grow steadily, neither too fast nor too slow. Chain quality requires that in any honest node's chain, any sufficiently long window of consecutive blocks contains a certain fraction of blocks that are mined by honest nodes. Consistency requires that all honest nodes' chains agree with each other except for the trailing few blocks. We will formally define these security properties below.

Definition 2 (Security of an abstract blockchain protocol). *We say that a blockchain protocol $\Pi_{\text{blockchain}}$ satisfies (T, g_0, g_1)-chain growth, (T, μ)-chain quality, and T-consistency w.r.t. some $(\mathcal{A}, \mathcal{Z})$, iff there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathit{EXEC}^{\Pi_{\text{blockchain}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold for view:*

- *(T, g_0, g_1)-chain growth.* A view satisfies (T, g_0, g_1)-chain growth iff the following hold:
  - *Consistent length:* If in round r some honest chain is of length ℓ, then in round $r + \Delta$, all honest chains must be of length at least ℓ.
  - *Growth lower bound:* For any r and t such that $g_0(t - r) \geq T$, let chain^r and chain^t denote two honest chains in round r and t respectively, it holds that
$$|\mathsf{chain}^t| - |\mathsf{chain}^r| \geq \lfloor g_0(t - r) \rfloor$$
  - *Growth upper bound:* For any r and t such that $g_1(t - r) \geq T$, let chain^r and chain^t denote two honest chains in round r and t respectively, it holds that
$$|\mathsf{chain}^t| - |\mathsf{chain}^r| \leq \lceil g_1(t - r) \rceil$$
- *(T, L, μ)-chain quality.* A view satisfies (T, L, μ)-chain quality iff the following holds: for any honest chain denoted chain in view, for any T consecutive blocks $\mathsf{chain}[j + 1..j + T]$, more than μ fraction of the blocks in $\mathsf{chain}[j + 1..j + T]$ are mined by honest nodes at most L blocks ago—here we say that a block $\mathsf{chain}[i]$ is "mined by an honest node at most L blocks ago" iff there is a set txs such that $\mathsf{txs} \subseteq \mathsf{chain}[i]$ and moreover $\mathcal{Z}$ input txs to some honest node when its last output to $\mathcal{Z}$ contains the prefix $\mathsf{chain}[: i - L]$ (here if $i - L < 0$, we round it to 0).

– *T-consistency.* A view satisfies T-consistency iff the following hold: for any two honest chains chain^r and chain^t in round r and $t \geq r$ respectively, it holds that

$$\mathsf{chain}^r[:-T] \prec \mathsf{chain}^t$$

We stress that since chain^r and chain^t can possibly belong to the same node, the above definition also implies "future self consistency" except for the trailing T blocks.

Liveness as a derived property. Intuitively, liveness requires that if honest nodes receive a transaction m as input, then m appear in honest chains very soon. More formally, we say that a blockchain protocol $\Pi_{\text{blockchain}}$ satisfies (K, T)-liveness w.r.t. some $(\mathcal{A}, \mathcal{Z})$ iff there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{blockchain}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds:

– Suppose that in any round $r \geq t$, $\mathcal{Z}$ always inputs a set that contains some m to every honest and online node i unless $\mathsf{m} \in \mathsf{chain}_i^r[:-T]$. Then, for any honest chain denoted chain in view whose length is at least $\ell + K + T$, it holds that $\mathsf{chain}[:\ell+K]$ contains m where ℓ denotes the shortest honest chain length at time t.

The liveness of a blockchain protocol is directly implied by chain growth and chain quality as stated in the following lemma.

Lemma 1 (Liveness). *Suppose that a blockchain protocol $\Pi_{\text{blockchain}}$ satisfies (K, g_0, g_1)-chain growth, (K', L, μ) chain quality and T-consistency w.r.t. some $(\mathcal{A}, \mathcal{Z})$ for positive parameters K, g_0, g_1, K', L, μ and T, then it holds that $\Pi_{\text{blockchain}}$ satisfies $(2K + 2g_1 + K' + L, T)$-liveness w.r.t. $(\mathcal{A}, \mathcal{Z})$.*

Proof. We ignore the negligible fraction of views where relevant bad events take place. Let r' be the earliest round in which some honest chain reaches length at least $\ell + K + g_1 + K' + L + T$, and let chain^* be an honest chain in round r' of length at least $\ell + K + g_1 + K' + L + T$. By chain quality, in the window $\mathsf{chain}^*[\ell + K + g_1 + L + 1 : \ell + K + g_1 + K' + L]$, there must be an honest block denoted B such that $\mathcal{Z}$ input (a subset of) the contents of B to some honest node i in round $r \leq r'$ when its chain contains the prefix $\mathsf{chain}^*[:\ell + K + g_1 + 1]$. By chain growth upper bound, the longest honest chain in round t must be of length at most $\ell + K + g_1$, and thus B must be input to some honest and online node i by $\mathcal{Z}$ in some round r where $t \leq r \leq r'$. By assumption, B must contain m unless $\mathsf{chain}_i^r[:-T]$ already contains m. By consistency, it must be that chain^* and chain_i^r are no longer than $\ell + 2(K + g_1) + K' + L + T$. By consistency, for any honest chain ch in view of length at least $\ell + 2(K + g_1) + K' + L + T$, it must be that $\mathsf{chain}_i^r[:-T] \prec \mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$ and $\mathsf{chain}^*[:-T] \prec \mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$, and thus $\mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$ must contain m.

Blockchain Implies State Machine Replication. Given any blockchain protocol $\Pi_{\text{blockchain}}$, it is easy to construct a state machine replication protocol where (1) nodes run an instance of $\Pi_{\text{blockchain}}$; (2) an honest node broadcasts all newly seen transactions to each other; and (3) in every round, nodes remove the trailing T blocks from the present chain (where T is the consistency parameter) and output the truncated chain to the environment $\mathcal{Z}$ [38,40]. It is not difficult to see that consistency (of the resulting state machine replication protocol) follows directly from consistency of the blockchain; and liveness follows from chain quality and chain growth. The above intuition has been formalized in earlier works [38,40].

2.4 Preliminaries: Responsiveness

Responsiveness. Recall that throughout this paper we always assume that $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded delay for some Δ, i.e., $\mathcal{Z}$ informs the protocol of a delay upper bound Δ upfront and all honest messages are then delivered within Δ number of rounds. A state machine replication protocol is said to be *responsive* if the transaction confirmation time is independent of the a-priori known upper bound Δ of the network's delay, but depends only on the actual maximum network delay. To put our results in perspective, we formally define the notion of responsiveness below and state a known lower bound result suggesting the impossibility of responsiveness against $\frac{1}{3}$ fraction of corruption. In the remainder of the paper, we will show that if one *optimistically* hopes for responsiveness only in lucky situations, then we can have protocols that retains consistency and liveness even under more than $\frac{1}{3}$ corruption. In practice, this means that we can have protocols that are responsive most of the time, and even when more than $\frac{1}{3}$ nodes are corrupt, the protocol can still guarantee consistency and liveness although performance would degrade when under attaack.

Responsiveness. We define a technical notion called responsiveness for a state machine replication protocol. Intuitively, responsiveness requires that except for a (possibly non-responsive) warmup period in the beginning, all transactions input afterwards will perceive transaction confirmation delay that is independent of the a-priori set upper bound Δ on the network's delay. As shown in earlier works [15,38], responsive state machine replication is impossible if $\frac{1}{3}$ or more fraction of the nodes are corrupt (even in a permissioned, classical environment with static corruptions, and even assuming that a proof-of-work oracle exists).

Definition 3 (Responsive state machine replication [38]). *Suppose that $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded delay for some Δ. We say that a state machine replication protocol Π satisfies $(T_{\text{confirm}}, T_{\text{warmup}}$-responsiveness w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff Π satisfies $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness w.r.t. $(\mathcal{A}, \mathcal{Z})$, and moreover the function T_{confirm} does not depend on the a-prior delay upper bound Δ.*

We say that a protocol Π satisfies consistency (or responsiveness resp.) in (n, f, Δ)-classical, static environments iff for every p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair that

respects (n, f, Δ)-classical execution and static corruption, Π satisfies consistency (or responsiveness resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$. We can similarly define (n, ρ, Δ)-sleepy, static environments and (n, ρ, Δ)-permissionless, static environments.

Theorem 4 (Impossibility of responsiveness against $\frac{1}{3}$ corruption [38]). *For any n and f such that $n \leq 3f$ and for any polynomial T_{confirm} in κ and δ, and T_{warmup} in κ, Δ, and δ, there exists some polynomial function Δ in κ such that no state machine replication protocol no state machine replication protocol can simultaneously achieve consistency and $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness in (n, f, Δ)-classical, static environments even assuming the existence of a proof-of-work oracle.*

The proof of the above theorem was presented by Pass and Shi in a recent work [38] where they modified the classical lower bound proof by Dwork et al. [15] and made it work even in the proof-of-work setting.

Recall that permissioned-classical is expressed as constraints on $(\mathcal{A}, \mathcal{Z})$ in our formal framework. This means that a lower bound for $n \leq 3f$ in the classical setting immediately implies a lower bound in more permissive settings where $(\mathcal{A}, \mathcal{Z})$ need not respect the permissioned-classical constraints as long as $n \leq 3f$ (or the equivalent holds). In other words, the above impossibility also applies to sleepy and permissionless settings (we defer formal theorem statements for these settings to the online full version [36]).

Interestingly, how to achieve responsive state machine replication against fewer than $\frac{1}{3}$ fraction of corruption is also known in the in the permissioned setting assuming the existence of a PKI [9], as well as in the permissionless setting assuming proof-of-work [38] (and under additional technical assumptions).

3 Basic Thunderella Protocol with a Static Committee

We first describe the basic Thunderella protocol assuming a static committee that is known a-priori to all nodes. We will discuss how to perform committee reconfiguration in the online full version [36]. For conceptual simplicity, we describe a version of the protocol where the blockchain is also collecting transactions constantly in the background—in practical implementations, it will not be too difficult to optimize our theoretical construction further such that the blockchain need not store an additional copy of all transactions under optimistic conditions.

As mentioned, in general, the Thunderella paradigm can be instantiated with any suitable asynchronous protocol to serve as the optimistic path and any suitable synchronous protocol to serve as the fallback path. However, we believe that a particular attractive instantiation is to use a simple voting-based protocol for the optimistic path and a blockchain as the fallback. Thus for concreteness, we will describe Thunderella for this specific instantiation.

Terminology. Our basic approach assumes three logical entities:

- *miners* of the underlying blockchain $\Pi_{\text{blockchain}}$;
- a *leader*; and
- a *committee* denoted committee.

To retain consistency and worst-case liveness (i.e., confirmation in the speed of the underlying $\Pi_{\text{blockchain}}$), we need to assume that (1) the underlying blockchain $\Pi_{\text{blockchain}}$ retains security (and this would translate to different compliance rules depending on how we instantiate the underlying blockchain); (2) α fraction of the committee are assumed to remain honest (but not necessarily online) where α is a knob that effectively allows us to trade-off security and performance as is explained later. Notably, the leader need not be trusted for consistency and worst-case liveness.

For concreteness, in our description we will often assume that $\alpha = \frac{1}{2}$, but in fact our approach generalizes to any choice where $0 < \alpha < 1$; and whenever appropriate, we will remark how to generalize the scheme's parameters for arbitrary α.

For simplicity, in this section we start out by assuming a static committee. In a permissioned setting, this committee can be the set of all nodes. In a permissionless setting where the set of players are not known in advance, we can elect the committee dynamically from the underlying blockchain using known techniques [37,38] or have stake-holders act as the committee [6,11,32]—however we defer the discussion of committee election and reconfiguration to the online full version [36]. Although we assume a static committee in this section, our basic protocol supports leader reconfiguration. In our presentation below we focus on describing the mechanism that enables leader reconfiguration without specifying concretely what leader re-election policy to adopt—exactly what policy to adopt depends on the application context and we thus defer the discussion of policies to the online full version [36].

3.1 Our Basic Protocol in a Nutshell

We first describe the intuition behind our basic protocol. For simplicity, we focus our description on what happens within a single epoch in which the identity of the leader is common knowledge.

Optimistic Fast Path. The optimistic fast path consists of a single round of voting to confirm each transaction (or batch). The leader serves as a coordinator and sequences transactions in the optimistic path. It tags each freshly seen transaction (or a batch) with a sequence number that increments over time, and the resulting tuple $(\mathsf{seq}, \mathsf{tx})$ is referred to as a notarization request. Whenever the committee members hear a notarization request $(\mathsf{seq}, \mathsf{tx})$ from the leader, it will sign the tuple $(\mathsf{seq}, \mathsf{tx})$ as long as it has not signed any tuple for seq before. For consistency, it is important that an honest committee member signs only one unique tuple $(\mathsf{seq}, \mathsf{tx})$ for every sequence number seq.

Whenever an honest node observes that a notarization request $(\mathsf{seq}, \mathsf{tx})$ has collected votes from more than $\frac{3}{4}$ of the committee, $(\mathsf{seq}, \mathsf{tx})$ is considered notarized. Although any notarized transaction is ready to be confirmed, an honest node is only allowed to output a notarized $(\mathsf{seq}, \mathsf{tx})$ tuple iff for every $s < \mathsf{seq}$, a tuple $(s, \_)$ has already been output. In other words, the output sequence is not allowed to skip any sequence numbers (since transactions must be processed in a linearized order). Henceforth, we referred to a sequence of notarized transactions with contiguous, non-skipping sequence numbers as a *lucky sequence*. In other words, honest nodes always output the maximal lucky sequence they have observed.

It is not hard to see that the optimistic, fast path satisfies the following properties as long as *the majority of the online committee members are honest* (below, we focus our discussion for the specific case $\alpha = \frac{1}{2}$, although the argument can easily be generalized to arbitrary choices of α):

- The following *agreement* property is satisfied even when the leader is corrupt and the committee may not be online: if any two honest nodes have output $(\mathsf{seq}, \mathsf{tx})$ and $(\mathsf{seq}, \mathsf{tx}')$ respectively, it must be that $\mathsf{tx} = \mathsf{tx}'$ (except with negligible probability over the choice of view).
- The following *liveness* property is satisfied only when the leader is honest and online and moreover more than $\frac{3}{4}$ of the committee are honest and online (i.e., when the optimistic conditions hold): every transaction input to an honest node will appear in all nodes' output logs in $O(1)$ actual roundtrips—in other words, when optimistic conditions hold, not only do we achieve liveness but we in fact also achieve *responsiveness*.

Note that when the optimistic conditions do not hold, liveness is not guaranteed for the optimistic path. For example, a corrupt leader can propose different transactions to different nodes for the same sequence number, and thus no transaction will collect enough votes to become notarized. Further, progress can also be hampered if not enough committee members are honest and online to vote.

Summarizing the above, if the leader is honest and online and moreover more than $\frac{3}{4}$ fraction of the committee are honest and online, all nodes will confirm transactions responsively in the optimistic path. However, to make our protocol complete, we need to deal with the case when either the leader is corrupt (or not online), or the committee is not more than $\frac{3}{4}$ honest and online—in the latter case, we wish to fall back to the worst-case guarantees offered by the underlying blockchain. Below we describe how such fallback can be achieved.

Falling Back to the Blockchain. In the fallback slow path, nodes will confirm transactions using the slow blockchain. The most non-trivial part of our protocol is how to switch between the optimistic path and the fallback path. To this end, we must answer the following two questions.

1. How do nodes decide when to fall back to the slow path?
2. Once the above decision is made, what is the mechanism for achieving this fallback?

When to fall back. The idea is to use the underlying blockchain to collect evidence of the optimistic path not working (e.g., either due to corrupt or crashed leader or due to not sufficiently many committee members being honest and online). Such evidence must be robust such that the adversary cannot raise false alarms when the optimistic path is actually working.

For conceptual simplicity, we can imagine the following: (1) whenever honest nodes mine a block, they incorporate into the block their entire view so far, including all unnotarized transactions and notarized transactions they have seen—in the actual protocol, the transactions stored in the blockchain can be easily deduplicated and compressed; (2) honest nodes always gossip their views to each other, such that if one honest node sees some (notarized or unnotarized) transaction by round r, then all honest nodes will have seen it by round $r + \Delta$. Thus by the liveness property of the underlying blockchain, if any (notarized or unnotarized) transaction is observed by any honest node in round r, then in roughly $\epsilon\kappa$ blocks of time, the transaction will appear in the blockchain.

Now, we may use the following criterion to detect when the optimistic path is not working:

> *Fallback condition:* Assume that chain is the stabilized prefix of some honest node's blockchain. If some unnotarized transaction tx appears in the block $\mathsf{chain}[\ell]$ but tx still has not become part of a lucky sequence contained in $\mathsf{chain}[: \ell + \kappa]$ where κ is a security parameter[4], then we conclude that the optimistic path has failed, and that a fallback is necessary.

Note that if the optimistic conditions hold, then the leader would have observed the unnotarized tx when its blockchain is roughly ℓ in length, and the committee would have notarized tx quickly; thus tx will soon become part of a lucky sequence contained in every node's blockchain. If this has not happened within κ blocks of time, then the optimistic conditions must no longer hold.

We also note that using the above mechanism, all honest nodes will decide to fall back within Δ rounds from each other. We now reach our next question: what mechanism do we rely on for the falling back?

How to fall back. The challenge is that when honest nodes decide to fall back (within Δ rounds from each other), although their optimistic logs are prefixes of each other, the logs could be of different lengths. One decision to make during the fallback is where (i.e., at which sequence number) to end the optimistic log before switching to blockchain mode—importantly, for consistency, honest nodes must agree on this decision. We point out that agreeing on this decision actually requires a full agreement instance—unlike the optimistic path where we punted on liveness, here this decision must be made with both consistency and liveness.

Thus the most natural idea is to rely on the underlying blockchain to reach agreement regarding this decision. To this end, we introduce the notion of a grace period that serves as a cool-down period before we eventually fall back into slow mode. The grace period consists of κ number of consecutive blocks where κ is

[4] Transactions of a lucky sequence are allowed to appear out of order in the blockchain.

a security parameter. Let chain denote the stabilized part of an honest node's blockchain and suppose that ℓ^* is the first block such that $\mathsf{chain}[:\ell^*]$ triggers the "fallback condition" as described above. Then, the grace period will consist of the blocks $\mathsf{chain}[\ell^*+1:\ell+\kappa]$. Informally speaking, the grace period is utilized in the following manner:

- Let LOG^* be an honest node's output log at the moment that the grace period starts (thus LOG^* must be a lucky sequence);
- Let chain be the stabilized prefix of this honest node's chain:
  - If the grace period has not ended in chain, then the node outputs the longer of (1) LOG^*; and (2) the maximal lucky sequence contained in chain. Note that in this case, the node does not output any additional transactions that are not part of the lucky sequence.
  - Else if the grace period has ended in chain, then the node first outputs the maximal lucky sequence contained in chain; then it outputs every other transaction (notarized or unnotarized) contained in chain (in the order that they are included in chain). In other words, after the grace period is over, the nodes start confirming transactions based on the blockchain.

Let $\mathsf{LOG}_{\max}$ denote the maximal lucky sequence contained in an honest node's blockchain by the end of the grace period. Effectively, in the above mechanism, nodes agree on $\mathsf{LOG}_{\max}$ before falling to blockchain mode. Importantly, the following informal claim must hold:

Claim (Informal). Except with negligible probability, $\mathsf{LOG}_{\max}$ must be at least as long as any honest node's output log when the node detects the start of the grace period.

To see why, recall that as mentioned earlier, all honest nodes gossip always their protocol views to each other; and honest nodes always embed their entire protocol view into any block they mine (in the actual protocol, the messages can be compressed). Thus, by liveness, any honest node's output log when the grace period starts will be in the blockchain κ blocks later.

Initiating a New Optimistic Epoch. So far, we have described our protocol from the perspective of a single epoch in which the leader is common knowledge. Whenever the protocol is in a slow path, however, we would like to allow the nodes to try to reinitiate an optimistic epoch and try to be fast again. This is easy to achieve since our underlying blockchain is always up and live! Thus one can simply rely on the underlying blockchain to implement any policy-based decision to reinitiate a new epoch. For example, the blockchain can be used to agree on (1) at which block length to reinitiate a new epoch; and (2) who will act as the leader in the new epoch. Our Thunderella framework leaves it to the application layer to specify such policy decisions (e.g., such policies can be implemented through generic smart contracts running atop the underlying blockchain).

Our detailed description in the remainder of this section is aware of the existence of multiple epochs, and thus transactions' sequence numbers are tagged with the epoch number to avoid namespace collision.

3.2 Detailed Protocol Description

We now formally describe our basic Thunderella protocol with a static committee. Our description and proofs are modularized. Specifically, we first describe the minimal set of protocol instructions necessary for guaranteeing consistency (Sect. 3.2)—in an actual implementation, security audit should be prioritized for this part of the protocol. We then describe other surrounding mechanisms (e.g., how to concretely instantiate the chain state function and how the leader proposes transactions) that allow us to additionally achieve worst-case liveness (Sect. 3.2) and optimistic responsiveness (Sect. 3.2).

Concrete blockchain parameters. For concreteness, henceforth in this section we assume a blockchain protocol denoted $\Pi_{\text{blockchain}}$ that achieves $(0.05\kappa, g_0, g_1 = \frac{1}{c\Delta})$-chain growth for some positive g_0 and some positive constant c, $(0.05\kappa, 1, \mu)$-chain quality where μ is positive, 0.05κ-consistency, and $(0.05\kappa, 0.05\kappa)$-liveness w.r.t. to any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{blockchain}}$. For our later concrete instantiations in permissionless and permissioned settings, existing blockchains constructions [19,35,40] would satisfy the necessary security properties given the above these parameters. Although we assume these concrete parameters, our Thunderella framework can easily be generalized to other parameters.

Useful Definitions. Henceforth, let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ denote a digital signature scheme.

Notarized transactions. We say that a tuple (e, s, m, V) is a *notarized* transaction for epoch e and sequence number s w.r.t. committee iff

- For each $(\mathsf{pk}, \sigma) \in V$, $\mathsf{pk} \in \mathsf{committee}$ and moreover σ is a valid signature for (e, s, m) under pk—in this case, we also say that (pk, σ) is a *valid vote* for (e, s, m).
- There are more than $\frac{3}{4} \cdot |\mathsf{committee}|$ votes in V with distinct pks.

If (e, s, m, V) is a notarized transaction, we also say that V is a *valid notarization* for (e, s, m).

Remark 1. Note that the above definition works for $\alpha = \frac{1}{2}$. For a general $\alpha \in (0, 1]$, we can simply replace the constant $\frac{3}{4}$ with $1 - \frac{\alpha}{2}$.

Blockchain states. We assume that there is a deterministic and efficiently computable function Γ such that given an abstract blockchain chain, the function Γ divides chain into multiple *epochs* interspersed with *interims*. Each epoch is a sequence of consecutive blocks in chain, and f also outputs a unique epoch

number e for each epoch. A sequence of consecutive blocks that do not belong to any epoch are called interim blocks. Each epoch always contains two sub-phases, an *optimistic period* followed by a *grace period*, each of which contains at least κ consecutive blocks and thus each epoch contains at least 2κ consecutive blocks (unless end of chain is reached).

Formally, we say that $\Gamma(\kappa, \cdot, \cdot)$ is a chain-state function iff for any chain and $0 \leq \ell \leq |\mathsf{chain}|$, $\Gamma(\kappa, \mathsf{chain}, \ell)$ outputs one of the following:

- some $(e, \texttt{optimistic})$: in this case we say that $\mathsf{chain}[\ell]$ is an optimistic block belonging to epoch e (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$);
- some $(e, \texttt{grace})$: in this case we say that $\mathsf{chain}[\ell]$ is a grace block belonging to epoch e (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$);
- or `interim`: in this case we say that $\mathsf{chain}[\ell]$ is an interim block (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$).

We say that a chain-state function $\Gamma(\kappa, \cdot, \cdot)$ is admissible iff for any chain:

1. for any $0 \leq \ell \leq \ell' \leq |\mathsf{chain}|$, if $\mathsf{chain}[\ell]$ belongs to epoch e and $\mathsf{chain}[\ell']$ belongs to epoch e', then $e' \geq e$;
2. for every e: all blocks corresponding to epoch e in chain must appear in a consecutive window, and moreover, all optimistic blocks for epoch e must appear before grace blocks for epoch e;
3. for every epoch e appearing in chain: there must be at least κ grace blocks belonging to epoch e in chain unless chain ends at an epoch-e block.
4. for every chain and every $0 \leq \ell \leq |\mathsf{chain}|$, $\Gamma(\kappa, \mathsf{chain}, \ell)$ depends only on $\mathsf{chain}[: \ell]$ but not $\mathsf{chain}[\ell + 1 :]$.

Lucky sequence. A sequence of notarized transactions $\{(e_i, s_i, \mathsf{m}_i, V_i)\}_{i \in [m]}$ is said to be a lucky sequence for epoch e iff for all $i \in [m]$, $e_i = e$ and $s_i = i$.

Blockchain linearization. Given an abstract blockchain chain, we do not simply output all transactions in chain in the most natural way. Instead, we adopt an algorithm denoted $\mathsf{linearize}^{\Gamma(\kappa,\cdot,\cdot)}(\mathsf{chain})$ for chain linearization. Henceforth we often write $\mathsf{linearize}(\mathsf{chain})$ for simplicity without explicitly denoting the chain-state function $\Gamma(\kappa, \cdot, \cdot)$.

Our chain linearization algorithm $\mathsf{linearize}(\mathsf{chain})$ is defined as follows: scan through the chain from left to right, and output the following:

1. For each epoch $\mathsf{chain}[\ell : \ell']$ encountered with the epoch number e, output the following in order:
   - first extract the maximal lucky sequence TXs for epoch e from $\mathsf{chain}[: \ell']$ and output $\mathsf{strip}(\mathsf{TXs})$ where $\mathsf{strip}(\cdot)$ will be defined below;
   - if $\mathsf{chain}[\ell]$ is not the end of chain, let TXs' be all remaining records in $\mathsf{chain}[\ell : \ell']$ not contained in TXs, output $\mathsf{strip}(\mathsf{TXs}')$;
2. For each interim $\mathsf{chain}[\ell : \ell']$ encountered, extract all transactions TXs from $\mathsf{chain}[\ell : \ell']$ and output $\mathsf{strip}(\mathsf{TXs})$.

In the above, the function $\mathsf{strip}(\cdot)$ removes signatures from notarized transactions: for a notarized transaction $\mathsf{strip}(e, s, \mathsf{m}, V) := (e, s, \mathsf{m})$; for an unnotarized transaction we define $\mathsf{strip}(\mathsf{m}) := \mathsf{m}$. If the input to $\mathsf{strip}(\cdot)$ is a sequence of transactions, the same operation is applied to each transaction.

$\Pi_{\mathbf{thunder}}$: Core Protocol for Consistency

Additional notation. A node's view consists of every message (including blockchains) it has received from $\mathcal{Z}$ or over the network. Henceforth we say that a notarized transaction (e, s, m, V) is in a node's view iff (e, s, m) exists in the node's view, and every $(\mathsf{pk}, \sigma) \in V$ exists in the node's view (not necessarily appearing together in the node's view). Multiple notarized transactions can exist for a unique (e, s, m) by taking different subsets of V—but in our presentation below, we always take V to be all the valid votes for (e, s, m) in a node's view, such that if for some tuple (e, s, m) there is a notarized transaction (e, s, m, V) in a node's view, then the choice is unique.

Assumptions. Although not explicitly noted, henceforth in all of our protocols, we assume that whenever an honest node receives any message on the network, if the message has not been broadcast before, the honest node broadcasts the message.

Protocol Π_{thunder}. Below we describe the $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ protocol that is parametrized by an admissible chain-state function $\Gamma(\kappa, \cdot, \cdot)$. Henceforth in our scheme, we often omit explicitly writing the chain-state function $\Gamma(\kappa, \cdot, \cdot)$.

- *Initialize.*
  - Call $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(\kappa)$ to generate a signing key pair. Output pk to $\mathcal{Z}$.
  - Wait to receive $\mathsf{committee}$ from $\mathcal{Z}$, and henceforth, validity of votes and acceptability of chains will be defined w.r.t. $\mathsf{committee}$.
  - Fork an instance of the $\Pi_{\text{blockchain}}$ protocol with appropriate parameters determined by ρ, n and Δ[5].
- *Notarize.* Upon receiving notarization request (e, s, m) from $\mathcal{Z}$: if $\mathsf{pk} \in \mathsf{committee}$ and no signature has been produced for (e, s) earlier, compute $\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(e, s, \mathsf{m})$ and broadcast $((e, s, \mathsf{m}), \sigma)$.
- *Propose.* Every round, let chain be the output from the $\Pi_{\text{blockchain}}$ instance.
  - Let TXs be a set containing (1) every notarized transaction (e, s, m, V) in the node's view such that no notarized transaction $(e, s, \mathsf{m}, \_)$ has appeared in $\mathsf{chain}[: -0.5\kappa]$; and (2) every unnotarized transaction m in the node's view such that no m or notarized transaction $(e, s, \mathsf{m}, \_)$ has appeared in $\mathsf{chain}[: -0.5\kappa]$.
  - Propose TXs to $\Pi_{\text{blockchain}}$.

[5] Unless otherwise noted, all messages sent from the $\Pi_{\text{blockchain}}$ instance or destined for $\Pi_{\text{blockchain}}$ are automatically passed through, but these messages also count towards the view of the current Π_{thunder} protocol instance.

- *Output.* In every round, let chain be the output from $\Pi_{\text{blockchain}}$.
  - If $\mathsf{chain}[-0.5\kappa]$ is an optimistic block belonging to epoch e:
    (a) let $\mathsf{chain}[-\ell]$ be the starting block for epoch e in chain where $\ell \geq 0.5\kappa$.
    (b) extract the maximal lucky sequence TXs for epoch e from the node's view so far.
    (c) let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[: -(\ell+1)])||\mathsf{strip}(\mathsf{TXs})$.
  - Else, let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[: -0.5\kappa])$.
  - Let LOG be the previous output to $\mathcal{Z}$: if $\overline{\mathsf{LOG}}$ is longer than LOG, output $\overline{\mathsf{LOG}}$; else output LOG to $\mathcal{Z}$.
- *Mempool.* Upon receiving any other message from the network or $\mathcal{Z}$, record the tuple.

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ iff

- $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{blockchain}}$;
- in every view in the support of $\mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, $\mathcal{Z}$ always inputs the same $\mathsf{committee}$ to all honest nodes;
- in every view in the support of $\mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, more than $\frac{1}{2}$ fraction (or in general, more than α fraction) of the distinct public keys in $\mathsf{committee}$ are output by nodes that remain honest (but not necessarily online) forever.

The following theorem says that for any chain-state function f that is admissible, Π_{thunder}^{f} satisfies consistency under compliant executions.

Theorem 5 (Consistency). *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Then, $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ satisfies consistency as defined in Sect. 2.2 w.r.t. any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$.*

The proof of this theorem is presented in the online full version [36]

Concrete Chain-State Function and Worst-Case Liveness. We will adopt the following chain-state function $\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)$ that is parametrized by a polynomial-time boolean predicate pred henceforth referred to as the "next-epoch" function. Basically, the job of pred is to examine the prefix of some blockchain and decide whether we want to advance to a larger epoch. Specifically, for some chain prefix $\mathsf{chain}[: i]$ if $\mathsf{pred}(\mathsf{chain}[: i], e) = 1$ then the blockchain wants to advance to epoch e if it is not already in epoch e—if there are multiple such e's such that the above holds, then the blockchain wants to go the largest such epoch.

At this moment, we define the chain state function Γ while leaving the pred unspecified. We will show that worst-case liveness is satisfied in compliant executions regardless of the concrete policy pred. Intuitively, our concrete chain state function is very simple: If the blockchain is currently in some epoch e, then the chain will stay in epoch e unless one of the following things happen:

1. either pred (applied to the prefix of the blockchain) wants to go to a larger epoch; or
2. during the current epoch some transaction did not get confirmed for a long time.

If one of the above did happen, then the chain gracefully transitions to an interim ensuring that there are at least κ optimistic blocks for the current epoch e followed by at least κ grace blocks for epoch e. If the blockchain is in an interim and pred wants to go to a next epoch, then we advance to the next epoch immediately. We note that for consistency and worst-case liveness, we in fact only need that there are at least κ grace blocks for each epoch (but not necessarily κ or more optimistic blocks). Here we additionally require that there are at least κ optimistic blocks for each epoch too—this gives the new epoch some time such that the blockchain can pick up possibly stale transactions that ought to have been confirmed such that we do not exit from the current epoch too soon.

More formally, for any chain, $\Gamma^{\mathsf{pred}}(\kappa, \mathsf{chain}, \cdot)$ is inductively defined as the following:

- The $\mathsf{chain}[0] := \mathsf{genesis}$ block is considered an interim block;
- If $\mathsf{chain}[i]$ is an interim block, let e be the largest epoch number such that $\mathsf{pred}(\mathsf{chain}[:i+1], e) = 1$, but no prefix of $\mathsf{chain}[:i]$ was ever in epoch e:
  - If such an epoch e is found: then $\mathsf{chain}[i+1..i+\kappa]$ are all optimistic blocks for epoch e' (and if $|\mathsf{chain}| < i+\kappa$, then all of $\mathsf{chain}[i+1:]$ are optimistic blocks for epoch e').
  - Else $\mathsf{chain}[i+1]$ is also an interim block;
- If $\mathsf{chain}[i]$ is the ℓ-th optimistic block of some epoch e where $\ell \geq \kappa$:
  - If one of the following two conditions C1 or C2 hold, then $\mathsf{chain}[i+1..i+\kappa]$ are all grace blocks for epoch e, and $\mathsf{chain}[i+\kappa+1]$ is an interim block (and if $|\mathsf{chain}| \leq i+\kappa$ then all of $\mathsf{chain}[i+1:]$ are grace blocks for epoch e):
    C1: some m or some notarized transaction $(\_, \_, \mathsf{m}, \_)$ appears in $\mathsf{chain}[:i-0.5\kappa]$ but $\mathsf{linearize}(\mathsf{chain}[:i])$ does not contain m or $(\_, \_, \mathsf{m})$, i.e., if some transaction has not occurred in any lucky sequence even after a sufficiently long time;
    C2: there exists some $e' > e$ such that $\mathsf{pred}(\mathsf{chain}[:i+1], e') = 1$, i.e., if the next-epoch policy function wants to switch to a larger epoch than the current one.
  - Else $\mathsf{chain}[i+1]$ is an optimistic block of epoch e.

Theorem 6 (Worst-case liveness). *Let $\Gamma(\kappa, \cdot, \cdot) := \Gamma^{\mathsf{pred}}(\cdot, \cdot, \cdot)$ be the chain-state function as specified above for any polynomial-time boolean predicate pred. Let g_0 denote the underlying $\Pi_{\mathrm{blockchain}}$'s chain growth lower bound parameter, and let $T_{\mathrm{confirm}}(\kappa) := \frac{3\kappa}{g_0}$. For any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\mathrm{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathit{EXEC}^{\Pi_{\mathrm{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: suppose that $\mathcal{Z}$ inputs a transaction m to an honest node in round r, then in any round $r' \geq r + T_{\mathrm{confirm}}(\kappa)$, all honest and online nodes' output LOG to $\mathcal{Z}$ will contain some $(\_, \_, \mathsf{m})$ or m.*

The proof of the above theorem is deferred to the supplemental material.

Coordination Protocol Π_{ella} and Optimistic Responsiveness. We now describe the full protocol $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ that spells out the leader-based coordination mechanism on top of Π_{thunder} as well as the next-epoch function pred. We will then show under exactly what optimistic conditions our protocol achieves responsiveness.

Description of protocol Π_{ella}. Π_{ella} calls $\Pi_{\text{thunder}}^{\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)}$ where the chain state function $\Gamma(\kappa,\cdot,\cdot) := \Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)$ is as defined in Sect. 3.2. We spell out the next-epoch function pred and the rest of Π_{ella} below.

- *Next-epoch function.* The policy function $\mathsf{pred}(\mathsf{chain}, e)$ takes in an abstract blockchain denoted chain and an epoch number e. If there exists a notarized transaction for epoch e in chain, then output 1; else output 0.
- *Initialize*: fork an instance of the $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ protocol.
- *Leader switch*: upon input $\texttt{leader}(e, i)$: if no leader has been recorded for epoch e, record i as the leader for epoch e, and do the following:
  - if current node is i: send a notarization request for a special epoch-start transaction $(e, s = 1, \texttt{start})$, and let $s = 2$;
  - for every notarization request (e, s, m) received earlier from node i, act as if (e, s, m) has just been received from i.
- *Notarization*: upon receiving notarization request (e, s, m) from i: if i has been recorded as the leader for epoch e, forward the notarization request (e, s, m) to $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$; else ignore the request.
- *Leader*: every round: let e be the largest epoch recorded thus far and if current node is recorded as the leader for epoch e:
  - for every m in view such that no m or $(\_, \_, \mathsf{m})$ appears in $\mathsf{linearize}(\mathsf{chain}[: -\kappa])$, if a notarization request has not been broadcast for m earlier, then broadcast the notarization request (e, s, m) and let $s := s + 1$.
- *Other messages*: pass through all other messages between $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and $\mathcal{Z}$; similarly pass through all other messages between $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and the network.

Compliant executions. To guarantee consistency and worst-case liveness, basically we just need the same conditions as our earlier $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ iff $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$.

Lucky epoch. Below we will describe exactly under what optimistic conditions can we achieve responsiveness. Roughly speaking, whenever a lucky epoch begins, after a short warmup time, we can achieve responsiveness. Specifically, during a lucky epoch, the epoch's leader is online and honest and more than $\frac{3}{4}$ fraction or in general, $1 - \frac{\alpha}{2}$ fraction of the committee remain honest and online.

Formally, given a view, we say that $[T_{\text{start}}, T_{\text{end}}]$ belongs to a lucky epoch corresponding to epoch e and leader i iff the following hold:

- In any round $r \geq T_{\text{start}} + \Delta$, any honest and online node should have received $\texttt{leader}(e, i)$ where i is the common leader that all honest nodes receive for epoch e. Further, prior to T_{start}, no honest node has received from $\mathcal{Z}$ any $\texttt{leader}(e', \_)$ instruction where $e' \geq e$.

- the leader (i.e., node i) is honest and online at in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$;
- more than $\frac{3}{4}$ fraction (or in general, more than $1 - \frac{\alpha}{2}$ fraction) of committee are honest and online[6] in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$.

Optimistic responsiveness in lucky epochs. We say that a protocol Π satisfies $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$: for any duration $[T_{\text{start}}, T_{\text{end}}]$ in view that belongs to a lucky epoch, $[T_{\text{start}}+T_{\text{warmup}}, T_{\text{end}}]$ is a T_{opt}-responsive period in view.

Theorem 7 (Optimistic case responsiveness). *Let g_0 be the underlying $\Pi_{blockchain}$'s chain growth lower bound parameter. For every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$, $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ satisfies $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs for $T_{\text{warmup}} = O(\frac{\kappa}{g_0})$, and $T_{\text{opt}} = 3\delta$ where δ is the actual maximum network delay in view.*

The proof of the above theorem is deferred to the online full version [36]. We note that Theorem 7 implies the following: informally speaking, if throughout the execution more than $\frac{3}{4}$ fraction of the committee remain honest and online and moreover, the initial epoch's leader remains honest and online, then once nodes enter the initial epoch, after a short warmup period, our protocol Π_{ella} will achieve responsiveness throughout the remainder of the execution (assuming that the underlying blockchain is secure).

Remark 2 (Leader re-election mechanism). In our scheme earlier, we left it unspecified how the environment $\mathcal{Z}$ will decide when to issue leader-switch instructions of the form $\texttt{leader}(e, i)$ that will cause nodes to start a new leader epoch. This is an application-specific policy decision. At this point, our paper focuses on providing a general framework that enables any application-specific policy decisions. In the online full version [36], we will give some suggestions on leader re-election policies that are useful in practice.

Deferred materials. We defer the full proofs, the lower bounds, as well as how to concretely instantiate the Thunderella framework in permissioned and permissionless environments allowing *committee reconfiguration* and leader rotation in the online full version [36]. We now conclude with the related work.

4 Related Work

State machine replication: classical and blockchain-style approaches. State machine replication or atomic broadcast (referred to as consensus for short in this paper) is a central abstraction of distributed systems, and has been extensively

[6] We say that a public key $\mathsf{pk} \in \mathsf{committee}$ is honest and online in round r if some node that is honest and online in round r output pk to $\mathcal{Z}$ earlier.

investigated and widely adopted in real-world systems. Roughly speaking, there are two, technically speaking, fundamentally different approaches towards realizing state machine replication, classical-style consensus [9,14,15,25,30,31], and blockchain-style consensus [11,19,26,34,35,40]. For a while, it has been vaguely understood by the community that blockchain-style protocols and classical ones achieve different properties—but the community has only recently begun to formally understand and articulate these differences.

The recent work by Pass and Shi [40] point out one fundamental difference between classical style and blockchain-style consensus. Most classical protocols [9,14,15,25,30,31], synchronous and asynchronous ones alike, rely on nodes having collected sufficiently many votes to make progress; thus these protocols would fail in a model where *participation is sporadic* and the exact number of players that do show up cannot be predicted upfront. More specifically, classical models of consensus would pessimistically treat nodes that do not show up as faulty (also referred to as crash fault); and if too many nodes do not show up, the protocol fails to make progress. In comparison, blockchain-style protocols can make progress regardless of how many players actually show up. Moreover, blockchain-style consensus has also been shown to be secure in a setting where the number of players can vary over time [18].

Classical deployments of consensus protocols are typically in a relatively small-scale and permissioned setting. Consensus in the permissionless setting was first empirically demonstrated to be possible due to Bitcoin's ingenious Nakamoto blockchain [34]. While the original Nakamoto blockchain relies on proofs-of-work to solve the Sybil attack in the permissionless setting, other proposals have been suggested since then for securely establishing identities in a permissionless setting—for example, proof-of-stake [2,3,7,10,11,26,27,32,41] is a most-oft cited approach where the stake-holders of a cryptocurrency system are responsible for voting on transactions. Recent works [32] have also explored adopting classical style consensus in a permissionless setting where approaches such as proof-of-stake can be used to establish identities.

Other closely related works. Our work is also reminisient of recent works that combine classical consensus and blockchains [12,28,38] although these works are of a different nature as we explain below. Among these works, Hybrid Consensus [38] is the only known formally correct approach, and moreover the only known approach that achieves *responsiveness*. From a theoretical perspective, our results are incomparable to Hybrid Consensus: we tolerate up to $\frac{1}{2}$ corruption in the worst-case and offer responsiveness only in the optimistic case but not in the worst case; in comparison, Hybrid Consensus achieves responsiveness even in the worst case—but in exchange, their protocol can only tolerate up to $\frac{1}{3}$ corruption, and this turns out to be inherent for any worst-case responsive protocol even when assuming proof-of-work [15,38]. From a practical perspective, Thunderella is more likely to be the protocol of choice in a real-world implementation partly due to its simplicity—in comparison, Hybrid Consensus requires a full-fledged classical protocol such as PBFT and Byzantine Paxos as a subroutine, and thus inherits the complexity of these protocols.

A line of research [8,13,16,21,22,33] has investigated Byzantine agreement protocols capable of early-stopping when conditions are more benign than the worst-case faulty pattern: e.g., the actual number of faulty nodes turns out to be smaller than the worst-case resilience bound. However, these works are of a different nature than ours as we explain below. First, these earlier works focus on stopping in a fewer number of synchronous rounds, and it is not part of their goal to achieve *responsiveness*. Second, although some known lower bounds [13] show that the number of actual rounds must be proportional to the actual number of faulty processors—note that these lower bounds work only for deterministic protocols, and thus they are not applicable in our setting.

Finally, the idea of combining asynchrony and synchrony was described in earlier works [4]; other works have also proposed frameworks for composing multiple BFT protocols [20]. However, to the best of our knowledge, none of the earlier works combined a synchronous fallback path and an asynchronous optimistic path in the manner that we do, allowing us to tolerate more than $\frac{1}{3}$ corruptions in the worst-case while still be responsive most of the time in practice.

Acknowledgments. We thank Jian Xie and Youcai Qian for inspiring conversations. We also thank Lorenzo Alvisi and Robbert van Renesse for helpful discussions and moral support. This work is supported in part by NSF grants CNS-1217821, CNS-1314857, CNS-1514261, CNS-1544613, CNS-1561209, CNS-1601879, CNS-1617676, AFOSR Award FA9550-15-1-0262, an Office of Naval Research Young Investigator Program Award, a Microsoft Faculty Fellowship, a Packard Fellowship, a Sloan Fellowship, Google Faculty Research Awards, and a VMWare Research Award.

References

1. Attiya, H., Dwork, C., Lynch, N., Stockmeyer, L.: Bounds on the time to reach agreement in the presence of timing uncertainty. J. ACM **41**(1), 122–152 (1994)
2. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Financial Cryptography Bitcoin Workshop (2016)
3. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: extending bitcoin's proof of work via proof of stake. In: NetEcon (2014)
4. Birman, K.P., Joseph, T.A.: Exploiting virtual synchrony in distributed systems. In: SOSP (1987)
5. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: OSDI (2006)
6. Buterin, V. (2017). https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c
7. Buterin, V., Zamfir, V.: Casper (2015). https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/
8. Castañeda, A., Gonczarowski, Y.A., Moses, Y.: Unbeatable consensus. In: DISC (2014)
9. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI (1999)
10. User "cunicula", Rosenfeld, M.: Proof of stake brainstorming, August 2011. https://bitcointalk.org/index.php?topic=37194.0
11. Daian, P., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919 (2016)

12. Decker, C., Seidel, J., Wattenhofer, R.: Bitcoin meets strong consistency. In: ICDCN (2016)
13. Dolev, D., Reischuk, R., Raymond Strong, H.: Early stopping in byzantine agreement. J. ACM **37**(4), 720–741 (1990)
14. Dolev, D., Raymond Strong, H.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. SIAMCOMP **12**(4), 656–666 (1983)
15. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**, 288–323 (1988)
16. Dwork, C., Moses, Y.: Knowledge and common knowledge in a byzantine environment I: crash failures. In: TARK, pp. 149–169 (1986)
17. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: FC (2014)
18. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. Cryptology ePrint Archive, 2016/1048 (2016)
19. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
20. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 BFT protocols. In: Proceedings of the 5th European Conference on Computer Systems, EuroSys 2010, pp. 363–376. ACM, New York (2010)
21. Halpern, J.Y., Moses, Y., Waarts, O.: A characterization of eventual Byzantine agreement. SIAM J. Comput. **31**(3), 838–865 (2001)
22. Herlihy, M., Moses, Y., Tuttle, M.R.: Transforming worst-case optimal solutions for simultaneous tasks into all-case optimal solutions. In: PODC (2011)
23. Herzberg, A., Kutten, S.: Early detection of message forwarding faults. SIAM J. Comput. **30**(4), 1169–1196 (2000)
24. Junqueira, F.P., Reed, B.C., Serafini, M.: Zab: high-performance broadcast for primary-backup systems. In: DSN (2011)
25. Katz, J., Koo, C.-Y.: On expected constant-round protocols for Byzantine agreement. J. Comput. Syst. Sci. **75**(2), 91–112 (2009)
26. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
27. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-of-stake (2012). https://peercoin.net/assets/paper/peercoin-paper.pdf
28. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. CoRR, abs/1602.06997 (2016)
29. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.L.: Zyzzyva: speculative byzantine fault tolerance. In: SOSP (2007)
30. Lamport, L.: Fast paxos. Distrib. Comput. **19**(2), 79–103 (2006)
31. Lamport, L., Malkhi, D., Zhou, L.: Vertical paxos and primary-backup replication. In: PODC, pp. 312–313 (2009)
32. Micali, S.: Algorand: the efficient and democratic ledger (2016). https://arxiv.org/abs/1607.01341
33. Moses, Y., Raynal, M.: No double discount: condition-based simultaneity yields limited gain. Inf. Comput. **214**, 47–58 (2012)
34. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)

35. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 643–673. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_22
36. Pass, R., Shi, E.: Thunderella: blockchains with optimistic instant confirmation. https://eprint.iacr.org/2017/913
37. Pass, R., Shi, E.: Fruitchains: a fair blockchain. In: PODC (2017)
38. Pass, R., Shi, E.: Hybrid consensus: efficient consensus in the permissionless model. In: DISC (2017)
39. Pass, R., Shi, E.: Rethinking large-scale consensus (invited paper). In: CSF (2017)
40. Pass, R., Shi, E.: The sleepy model of consensus. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 380–409. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_14
41. User "QuantumMechanic": Proof of stake instead of proof of work, July 2011. https://bitcointalk.org/index.php?topic=27787.0
42. Song, Y.J., van Renesse, R.: Bosco: one-step byzantine asynchronous consensus. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 438–450. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87779-0_30