Hybrid Consensus: Efficient Consensus in the Permissionless Model*†

Rafael Pass[1] and Elaine Shi[2]

1 **CornellTech, New York, NY, USA**
`rafael@cs.cornell.edu`
2 **Department of Computer Science, Cornell University, Ithaca, NY, USA**
`elaine@cs.cornell.edu`

Abstract

Consensus, or state machine replication is a foundational building block of distributed systems and modern cryptography. Consensus in the classical, "permissioned" setting has been extensively studied in the 30 years of distributed systems literature. Recent developments in Bitcoin and other decentralized cryptocurrencies popularized a new form of consensus in a "permissionless" setting, where anyone can join and leave dynamically, and there is no a-priori knowledge of the number of consensus nodes. So far, however, all known permissionless consensus protocols assume network synchrony, i.e., the protocol must know an upper bound of the network's delay, and transactions confirm slower than this a-priori upper bound.

We initiate the study of the feasibilities and infeasibilities of achieving responsiveness in permissionless consensus. In a responsive protocol, the transaction confirmation time depends only on the *actual* network delay, but not on any a-priori known upper bound such as a synchronous round. Classical protocols in the partial synchronous and asynchronous models naturally achieve responsiveness, since the protocol does not even know any delay upper bound. Unfortunately, we show that in the permissionless setting, consensus is impossible in the asynchronous or partially synchronous models.

On the positive side, we construct a protocol called *Hybrid Consensus* by combining classical-style and blockchain-style consensus. Hybrid Consensus shows that responsiveness is nonetheless possible to achieve in permissionless consensus (assuming proof-of-work) when 1) the protocol knows an upper bound on the network delay; 2) we allow a non-responsive warmup period after which transaction confirmation can become responsive; 3) honesty has some stickiness, i.e., it takes a short while for an adversary to corrupt a node or put it to sleep; and 4) less than 1/3 of the nodes are corrupt. We show that *all these conditions are in fact necessary* – if only one of them is violated, responsiveness would have been impossible. Our work makes a step forward in our understanding of the permissionless model and its differences and relations to classical consensus.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Distributed Consensus, Permissionless, Responsiveness

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.39

1 Introduction

State machine replication, also commonly referred to as atomic broadcast, has always been a central distributed systems abstraction. In a state machine replication protocol, a set

* A full version of the paper is available at `https://eprint.iacr.org/2016/917`.
† This research is funded in part by NSF through grant number CNS-1561209.

31st International Symposium on Distributed Computing (DISC 2017).
Editor: Andréa W. Richa; Article No. 39; pp. 39:1–39:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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.

For more than a decade, companies such as Google and Facebook rely on state machine replication protocols (in the crash fault model) to replicate a significant fraction of their database and computing infrastructure. For simplicity, henceforth we use the term *consensus* to refer to state machine replication protocols throughout this paper.

Traditionally, consensus protocols were studied in a permissioned setting where the set of participating nodes is known a-priori. The enormous success of decentralized cryptocurrencies such as Bitcoin and Ethereum have brought to our attention a new model of distributed consensus, that is, the *permissionless* model. Informally speaking, a permissionless model has the following notable differences from the classical permissioned model:

1. anyone can join the protocol and the network provides no authentication;
2. nodes come and go;
3. there may not be a priori knowledge how many nodes will actually show up; and
4. the number of nodes participating can vary over time.

Although recent (and concurrent) works have begun to explore the theoretical (in)feasibility of reaching consensus in the permissionless model [13, 17, 19, 21], our understanding of the permissionless model is nonetheless relatively little in comparison with our rich body of knowledge for permissioned consensus. In this paper, we make an endeavor at furthering our understanding of the permissionless model.

All known permissionless consensus protocols work in the synchronous model. So far, it is somewhat well-understood that proofs-of-work can be employed to thwart Sybil attacks [5, 13,14,18,19] and thus circumvent earlier known theoretical infeasibilities for the permissionless model [7] – but all known consensus protocols in the permissionless model rely on *network synchrony* [5, 13, 14, 18, 19]. Remarkedly, the famous Nakamoto blockchain [13, 18, 19] (that underlies Bitcoin) also *works only in the synchronous model* where the protocol must know an a-priori upper bound of the network delay (henceforth denoted Δ) [19]. Otherwise, if a delay upper bound Δ is unknown, Pass et al. [19] shows that Nakamoto blockchain's security can be broken. Pass et al. [19] also show that the expected block interval of Nakamoto's blockchain must be set to be, roughly speaking, a constant factor larger than Δ for consistency.

Relying a synchronous model, however, can be undesirable in practice. Since the synchrony parameter Δ must be set conservatively to leave a sufficient safety margin, in practice the actual network's delay is typically much better than this pessimistic upper bound. Unfortunately, any protocol that must wait for at least one synchronous round (or one block-interval) for transaction confirmation cannot benefit from the network's actual performance. We naturally desire a protocol that can confirm transactions as fast as the network makes progress, a notion that we shall henceforth refer to as *responsiveness*. The notion of responsiveness was first defined by Attiya et al. [6]: in a responsive consensus protocol, we require that the transaction confirmation time is a function of the actual network's delay δ, but not of the a priori upper bound Δ that is provided to the protocol as input [6].

We thus ask the following natural question:

Can we achieve responsiveness in permissionless consensus? In other words, *is slow confirmation inherent in permissionless consensus?*

1.1 Our Results and Contributions

Impossibility of permissionless consensus in the partially synchronous model. Traditionally, permissioned state machine replication protocols achieve responsiveness by adopting a partially synchronous or asynchronous model[1], where the protocol does not know any a-priori upper bound of the network delay Δ.

Thus, to answer our earlier question, a first attempt is to design a permissionless consensus protocol for the partially synchronous or asynchronous setting. Unfortunately, this task turns out to be impossible – we show that no consensus protocol unaware of a delay upper bound Δ can simultaneously guarantee consistency and liveness in the permissionless setting – yet another reason why the permissionless setting is fundamentally different from permissioned!

Is responsiveness hopeless for permissionless? At first sight, this pessimistic observation seems to have closed the question: perhaps waiting for the synchronization delay is inevitable in permissionless consensus. Even more discouragingly, it is easy to see that when the number of nodes n is unknown a priori, this lower bound generalizes to one-shot consensus even when the protocol knows Δ but must be responsive – by one-shot consensus, we mean that we need to confirm exactly one transaction proposed to honest nodes upfront, and responsiveness requires that the confirmation time be independent of Δ.

Hybrid consensus. One might now be attempted to think that responsiveness is impossible in a permissionless setting. Perhaps somewhat surprisingly, we construct a protocol called Hybrid Consensus through which we show that responsiveness is actually possible in permissionless state machine replication (assuming proof-of-work), provided that

1. less than $\frac{1}{3} - \epsilon$ fraction of nodes are corrupt where ϵ is an arbitrarily small constant;
2. the protocol is aware of the upper bound Δ, and moreover, we allow a warmup period that depends on Δ – transactions start to confirm as fast as the actual network's performance after this warmup period but not before; and
3. honest nodes stick around and remain honest for a while even when an adversary may try to adaptively corrupt or crash the node – in other words, corruption is not an instant operation but requires some time to take effect. We shall formally define this requirement as the τ-mildly adaptive corruption model where τ is the stickiness parameter.

Lower bounds. *Are the above constraints necessary for achieving responsiveness in the permissionless model?* Earlier we have argued that requirement (2) is necessary since responsiveness is impossible for one-shot consensus even when the protocol knows Δ. Later we will prove several simple but hopefully insightful lower bounds, and show that indeed, all the remaining conditions are necessary as well!

1. We prove that $\frac{1}{3}$ corruption is the optimal resilience parameter for responsiveness – even assuming proof-of-work, static corruption, and even when the number of players is known in advance and all players spawn upfront.
2. If honest nodes do not stick around at least for a while (i.e., if the stickiness parameter $\tau = 0$), responsiveness would also be impossible when the number of players can vary by a factor of 2 (or more) in adjacent rounds.

In summary, our paper gives a somewhat comprehensive answer towards understanding the feasibilities and infeasibilities of achieving *responsiveness* in permissionless consensus.

[1] For the purpose of this paper, the technical difference between partial synchrony and asynchrony is non-essential.

Our results contribute to the further understanding of the permissionless model, how it differs from the classical permissioned model, and how techniques from permissioned consensus can lend to permissionless consensus.

2 Preliminaries

2.1 Permissionless Execution Model

We consider a standard, *round-based* Interactive Turing Machine (ITM) execution model. The execution proceeds in atomic time steps called *rounds* – henceforth in our paper the terms *time* and *round* are used interchangably.

Corruption model. We consider the following corruption model.

- *Spawning.* The adversary is allowed to *spawn* new nodes in any round during the execution. Newly spawned nodes can either be honest or corrupt.
- *Corruption.* To corrupt a node during the protocol, the adversary must first issue a "target corrupt" instruction to an honest node i – let t denote this round. This "target corrupt" instruction will take effect only τ rounds later, i.e., the node i actually becomes corrupt in round $t + \tau$. When a node actually becomes corrupt, its internal states are exposed to the adversary and the adversary fully controls the node's actions. Henceforth, the parameter τ is referred to as the agility parameter. If $\tau = 0$, we model fully adaptive corruptions; if $\tau = \infty$, we model static corruption; anywhere in between, corruption is said to be *mildly adaptive*.
- *Killing.* The adversary can issue a "kill" instruction to kill a corrupt node, and the instruction takes effect immediately. A killed node is no longer considered live. The adversary is not allowed to corrupt honest nodes directly without corrupting them first.

Henceforth, in every round, all nodes that have been spawned but have not been killed are considered live. We say that the adversary is subject to a ρ corruption budget iff in every round, the number of honest nodes that have not received "target corrupt" is more than $1 - \rho$ fraction of the total number of live nodes in that round.

Communication model. The adversary is responsible for delivering all messages sent by nodes (honest or corrupt) to *all* other nodes. The adversary cannot modify the contents of messages broadcast by honest nodes, *but it may arbitrarily delay or reorder the delivery of a message* subject to the following δ-bounded delay constraint: if an honest node sends a message at time t, then in any round $r \geq t + \delta$, any honest node that is live 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. We assume that the identity of the sender is not known to the recipient.

Henceforth in this paper, we assume that the protocol is provided with an input Δ which is an a-priori upper bound of the actual network delay δ.

Number of nodes. For simplicity we make a mild assumption about how the number of players may vary. We assume that

1. the protocol is given an estimate n^* of the number of players; and
2. the number of players in every round is between $[\frac{n^*}{2}, n^*]$.

We stress that our partial synchrony impossibility and fully adaptive impossibility results hold as long as the protocol's estimate of the number of players can be off by a factor of 2 (or more)[2].

2.2 State Machine Replication

In this paper, a *permissionless consensus* protocol is a protocol that realizes the "state machine replication" abstraction in a permissionless environment. In a state machine replication protocol, in every round, every honest node receives one or more transactions as input, and outputs a log. Henceforth let LOG_i^r denote node i's output log in round r. A state machine replication protocol is said to satisfy *consistency*, iff except with negligible probability,

1. for any node i honest in round r and any node j honest in round t, either $\mathsf{LOG}_i^r \prec \mathsf{LOG}_j^t$ or $\mathsf{LOG}_j^t \prec \mathsf{LOG}_i^r$, where $\prec$ denotes "is a prefix of" (by convention, we assume that $x \prec x$ for any x); and
2. any honest node's log should not shrink over time.

A state machine replication protocol is said to satisfy $(T_{\text{warm}}, T_{\text{confirm}})$-*liveness*, iff except with negligible probability, if an honest node receives some transaction tx as input or has tx in its output log in round $r \geq T_{\text{warm}}$, then in round $r + T_{\text{confirm}}$ or later, tx appears in all honest nodes' output logs.

Responsiveness. A state machine replication protocol that satisfies $(T_{\text{warm}}, T_{\text{confirm}})$-*liveness* is said to be responsive, iff the function T_{confirm} depends only on the actual maximum network delay δ but not on the a-priori upper bound Δ. Note that we allow the warmup period T_{warmup} to be non-responsive, i.e., dependent on the a-priori upper bound Δ.

3 Limits on Responsiveness in Permissionless Consensus

We present our lower bounds for responsiveness. Due to space constraints, we present proof intuitions in this section and defer full, formal proofs to our online full version [20].

Impossibility of partial synchrony for permissionless. One obvious approach towards achieving responsiveness is to rely on a partially synchronous or asynchronous model – indeed, classical, permissioned consensus protocols in the partially synchronous or asynchronous models naturally provide responsiveness, since the protocol does not even know a network delay upper bound. Unfortunately, as it turns out, the same approach would fail in the permissionless setting.

We prove the following theorem where a partially synchronous protocol is defined to be one whose protocol instructions do not depend on the a-priori delay upper bound Δ.

▶ **Theorem 1** (Impossibility of partial synchrony). *No partially synchronous, permissionless consensus protocol can achieve both consistency and liveness in an execution environment where the protocol is provided with an a-priori estimate of the number of players that can be off by a factor of 2 (or more) – even when no node is corrupt.*

[2] On the other hand, for upper bounds: although the earlier blockchain analysis work by Pass et al. [19] and Fruitchain [21] assume that the number of players in every round stays fixed and is known to the protocol – we observe that it is straightforward to extend these earlier works to the case when the number of players may vary by any known constant factor, and that the protocol is given an estimate that can be off by a known constant factor (see online full version [20] for additional details).

Proof sketch. We explain the proof intuition but defer the formal proof to the online full version [20]. Imagine an execution where honest nodes P_0 and P_1 have a slow network link. P_0 cannot distinguish whether its network link to P_1 is slow, or if P_1 simply did not show up. Had it been the latter case, P_0 must output a decision quickly (and for a sufficiently large choice of Δ, the decision will be output before the end of Δ rounds). Thus P_0 must output a decision quickly in the current execution too (i.e., former case). By symmetry, P_1 must output a decision quickly too. Now, if P_0 and P_1 received different, high-entropy transactions as inputs, they would have resulted in disagreement except with negligible probability. ◀

Resilience lower bound. We next ask the question, suppose that we do allow the protocol to know Δ, and moreover we allow a non-responsive warmup period – in this case, can we achieve responsive permissionless consensus, and if so, what fraction of corruption can we tolerate? We prove the following lower bound.

▶ **Theorem 2** (Resilience lower bound)**.** *No responsive, permissionless consensus protocol can tolerate $\frac{1}{3}$ or more corruptions, even assuming the existence of a proof-of-work oracle, static corruptions, and even when assuming that all nodes are spawned upfront (i.e., no late spawning), and moreover, the protocol is provided with the exact number of players.*

Proof sketch. Our proof is inspired by the well-known $\frac{1}{3}$-corruption lower bound by Dwork et al. [11]. Dwork et al.'s proof considers an execution with three nodes, honest nodes P_0, P_1 and a corrupt node Q. The corrupt node Q simulates two nodes Q_0 and Q_1 in its head where Q_b plays with P_b for $b \in \{0, 1\}$. Messages between P_0 and P_1 are delayed for a sufficiently long time. Dwork et al. argues that in such an execution, P_0's view is the same as an alternate execution where P_1 is the corrupt node, and thus P_0 should output a decision responsively. By symmetry, so will P_1. Now, if P_0 and P_1 received different, high-entropy transactions as inputs, then they would disagree.

The main challenge in our proof is that we must essentially prove the same statement, but assuming a proof-of-work oracle. Since a corrupt node can query the proof-of-work oracle only once in each time step, the adversary cannot simulate two parallel executions without causing any slowdown. Fortunately, there is a way to temporally interleave the two simulated executions, such that the second execution waits for the first one to finish before starting – and the lapse in time till the second execution starts can be charged to the network delay. We defer the full, formal proof to the online full version [20]. ◀

Impossibility of responsiveness with a fully adaptive adversary. We show that responsiveness is impossible if the adversary is fully adaptive, i.e., $\tau = 0$, and moreover, if the number of players in every round can differ by a factor of 2. The intuition is the following: when the agility parameter $\tau = 0$, i.e., if "honesty" has no stickiness, then the adversary can corrupt and kill honest nodes instantly. In other words, the adversary can kill an old batch of honest nodes and spawn a new batch instantly, such that the nodes in two adjacent rounds are completely disjoint – notice that the adversary can do this without even charging to the corruption budget ρ! Thus every round behaves like the start of the execution where the number of online nodes is unpredictable by a factor of $2\times$. This means that even a non-responsive warmup period will not help. We formalize this intuition in the following theorem. It is interesting to note why this lower bound no longer holds when honesty does have some stickiness. In this case, swapping out an old batch of nodes and swapping in a new batch is no longer for free – since when an honest node receives a "target corrupt" instruction, it is charged to the corruption budget ρ.

▶ **Theorem 3** (Impossibility of responsiveness with a fully adaptive adversary)**.** *No protocol (even in the proof-of-work model), given an estimate n^*, can achieve responsive permissionless consensus in a fully adaptive environment (i.e., $\tau = 0$) where the number of nodes in each round is allowed to vary between $[\frac{n^*}{2}, n^*]$ – moreover this holds for any corruption budget ρ.*

We defer the formal proof of the above theorem to our online full version [20].

4 Hybrid Consensus

4.1 Blockchain Preliminaries

An abstract blockchain protocol. One way to realize state machine replication is through a blockchain protocol. In a blockchain protocol, in every round, every honest node receives one or more transactions as input. In every round, every honest node outputs a chain consisting of linearly ordered blocks, where each block is a sequence of logical records (e.g., transactions). As defined by Garay et al. [13] and Pass et al. [19], a blockchain protocol is expected to satisfy three properties except with negligible probability:

1. *consistency*, i.e., all honest nodes' output chains agree with each other except for the trailing λ blocks where λ is a security parameter;
2. *Q-chain quality*, i.e., in any λ consecutive blocks in an honest node's output chain, more than Q fraction must be contributed by honest nodes that have not received "target corrupt"; and
3. *(G, G')-chain growth*, i.e., over any duration of length t where $Gt \geq \lambda$, honest nodes' chains grow by at least Gt and at most $G't$.

It is not hard to see that given a blockchain protocol snailchain, we can construct a non-responsive permissionless consensus protocol, where we simply run the blockchain protocol snailchain, and have each node's output log be its chain minus the trailing $\Theta(\lambda)$ blocks. In particular, the resulting permissionless consensus protocol's liveness can be inferred from snailchain's (positive) chain quality and chain growth lower bound[3].

Nakamoto's blockchain. Earlier, Garay et al. [13] and Pass et al. [19] analyze Nakamoto's blockchain protocol [18] for the case when the number of players n is fixed – we observe that it is straightforward to extend these earlier results to deal with the case when n varies by a known constant factor (see our online full version [20] for additional details). In other words, the following statement is immediately implied by these earlier works [13, 19]: assuming the existence of an "idealized" proof-of-work oracle, there exists a blockchain protocol, as long as

- **(i)** the protocol is provided with an estimate of the number of players that is off by a known constant factor;
- **(ii)** the number of players in every round varies by a known constant; and
- **(iii)** the protocol is aware of a upper bound Δ of the network's delay.

[3] Although a blockchain abstraction may resemble the state machine replication definition in Section 2.2, the blockchain abstraction additionally allows us to express a rough notion of time through chain growth, and express "fairness" through chain quality.

4.2 Our Ideas in a Nutshell

To clarify our contributions, the high-level idea of combining classical- and blockchain-style consensus has been proposed before [9] or in concurrent work [15]. However, these earlier works do not achieve responsiveness or are flawed (see Section 5 for detailed discussions). Our hybrid consensus scheme is different in nature from these other works [9, 15] despite the superficial resemblance in the high-level idea. We now explain intuitively how to derive our hybrid consensus scheme step by step.

At a very high level, the idea is to run an underlying blockchain protocol denoted snailchain – for the time being, imagine that we are running Nakamoto's blockchain as the snailchain. We rely on the blockchain to re-elect committees over time where each committee consists of recently online miners. Each committee will now execute a classical, partially synchronous consensus instance (e.g., PBFT) henceforth referred to as "daily consensus" to confirm transactions. To periodically elect committees, we can do the following: whenever the blockchain advances by λ number of blocks, we re-elect a committee in the following way:

1. first, chop off $\Theta(\lambda)$ number of trailing, unstable blocks; and
2. in the remaining chain, the most recent λ blocks' miners' are elected as the new committee.

Now, we make the following useful observations:

1. Due to the *consistency* property of the blockchain, as long as we removed the trailing $\Theta(\lambda)$ blocks, except with negligible probability, all nodes will agree on the committee – for this reason, removing the trailing $\Theta(\lambda)$ blocks is important, and previous works that neglected this [15] could lead to inconsistency.
2. If the underlying blockchain satisfies $\frac{2}{3}$-*chain quality*, then in every window of λ consecutive blocks in an honest node's chain, at least $\frac{2}{3}$ fraction of blocks that are contributed by honest nodes (that have not received "target corrupt").
3. Finally, due to *chain growth*, it does not take too long to re-elect each new committee.

To make our scheme fully work, however, various non-trivial challenges must be addressed, including

1. how to achieve (near) optimal resilience;
2. how to smoothly switch between multiple daily consensus instances and compose their output logs; and
3. how to deal with a posterior corruption attack.

As we discuss more in the Section 5, some earlier works [9, 15] neglect a subset to all of these issues, making their claims somewhat incomplete or flawed. We now discuss how to address these non-trivial technicalities one by one.

Achieving optimal resilience. For near optimal resilience, our hope is that we can achieve $\frac{2}{3}$-chain quality as long as more than $\frac{2}{3}$ of the nodes are honest (and have not received "target corrupt") – however, perhaps somewhat surprisingly, this is fact *false* for Nakamoto's blockchain! Due to a well-known selfish mining attack [12], Nakamoto's blockchain in fact does not achieve "perfect" chain quality. Specifically, for Nakamoto's blockchain to achieve $\frac{2}{3}$ chain quality, we must in fact assume that more than $\frac{3}{4}$ of the nodes are honest!

To aid understanding, we briefly explain the selfish mining attack. When a corrupt node mines a block, it withholds the block without releasing it to honest nodes. Later, when an honest node mines a equal-length fork, the adversary now immediately releases his private block to race against the honest node's. If the adversary additionally controls network delivery, he can perform a rushing attack such that his private fork is guaranteed to arrive

first at other nodes, and thus everyone will now mine off the adversary's private fork. Such an attack effectively "erases" a portion of the honest node's mining power, a direct effect of which is degraded chain quality as mentioned above.

As argued above, if we use Nakamoto's blockchain as the underlying snailchain, we can tolerate only $\frac{1}{4}$ corruption. However, our goal is to tolerate $\frac{1}{3} - \epsilon$ corruption and thus achieve near optimal resilience. To this end, we rely on the recent Fruitchains [21] as a drop-in replacement for Nakamoto's blockchain, i.e., we will instantiate snailchain with Fruitchains [21]. Interestingly, Fruitchains realizes exactly the same abstraction as Nakamoto's blockchain but achieves near optimal chain quality – more specifically, it achieves this by piggybacking two independent mining processes on top of each other, one for mining fruits, and one for mining blocks. In Fruitchains, blocks contain fruits and fruits in turn contain transactions. One can show that the underlying blockchain's liveness guarantees that honest nodes' work in mining fruits cannot be erased by the adversary. Thus, if we regard the fruits as the new blocks, Fruitchain achieves near optimal chain quality.

Switching between daily consensus instances. Another technicality is how to smoothly transition between multiple daily consensus instances without causing "glitches" in responsive transaction confirmation. To this end, our idea is the following: whenever the blockchain advances by another λ number of blocks, we initiate a stopping procedure for the present daily consensus instance while simultaneously starting the next daily consensus instance. Since the stopping procedure may take some time to complete, during a short window, two or more daily consensus instances may be executing concurrently. Although the new daily consensus instance may start accumulating a log, we defer including this log in the output until the previous daily consensus instance has fully terminated and its log fully output. It is not hard to see that as long as this stopping procedure is responsive, all transactions will confirm responsively without any glitches during the switch. In our online full version [20], we show that subtle composition issues arise when composing multiple daily consensus instances.

On-chain stamping and posterior corruption defense. Although it takes a while for the adversary to adaptively corrupt nodes, it is possible for the adversary to eventually corrupt entire past committees. We would like to retain consistency even when entire past committees can be corrupt – henceforth we refer to this problem as *posterior corruption*. The challenge with posterior corruption is that the corrupt past committee can sign equivocating transactions; and whenever a new node joins the protocol, it cannot distinguish which signature is real and which signature was generated a-posteriori by a corrupt past committee – this can lead to inconsistency. To defend against such a posterior corruption attack, we introduce an on-chain stamping mechanism. When a committee completes its term of appointment (where each term is said to be a day), it will propose hash of the daily log, signed by more than $\frac{1}{3}$ of the committee, to be stamped on the blockchain. Now, although the entirety of a past committee can become corrupt sometime after its term of appointment and can sign arbitrary messages of the past – at this point, these signatures will be too late to deceive anyone, since the hash of the committee's daily log will already be stamped on the blockchain, and thus the corrupt committee of the past can no longer equivocate about their past daily log unless they can find hash collisions.

4.3 Detailed Protocol

We present our protocol below but defer formal definition of building blocks and proofs to the online full version [20]. For modular composition, we make use of a global signing

functionality $\mathcal{G}_{\text{sign}}^{\Sigma}$ (parametrized by a signature scheme Σ) shared by all protocol instance. $\mathcal{G}_{\text{sign}}$ provides the following interfaces:

1. upon receiving `keygen` from a node i: $\mathcal{G}_{\text{sign}}$ calls Σ's key generation to generate a new $(\mathsf{pk}, \mathsf{sk})$ pair and returns pk;
2. upon receiving `mykeys` from a node i, output the set of all public keys that have been generated for i;
3. upon receiving $\texttt{sign}(\mathsf{pk}, \mathsf{msg})$ from a node i, if pk was recorded as a public key generated for i, call $\Sigma.\mathsf{Sign}(\mathsf{sk}, (sid, \mathsf{msg}))$ where sid is the current session identifier, and sk is the secret key corresponding to pk, and return the signature;
4. when i becomes corrupt: return all of node i's secret keys to $\mathcal{A}$.

4.3.1 Offchain BFT

We call each committee's term of service a *day*. For modular protocol composition, we define an intermediate abstraction called a daily offchain consensus protocol, denoted DailyBFT. In DailyBFT, committee members run an offchain BFT instance to decide a daily log, whereas non-members count signatures from committee members. A DailyBFT$[R]$ parametrized by the session identifier R (representing the day) has the following syntax.

Inputs and outputs. In each time step, the environment $\mathcal{Z}$ can provide the following types of inputs multiple times:

1. $\texttt{start}(\mathsf{comm})$ where $\mathsf{comm} = \{\mathsf{pk}_i\}_{i \in [m]}$;
2. TXs; and
3. `stop`.

Honest nodes output the following to $\mathcal{Z}$:

- In each time step t, honest nodes output to the environment $\mathcal{Z}$ $\texttt{notdone}(\mathsf{log}^t)$, until in one final step t^*, it outputs $\texttt{done}(\mathsf{log}^{t^*}, \mathsf{recs})$, where recs is either $\emptyset$ or a set of signed tuples vouching for the hash of the final daily log. After outputting $\texttt{done}(\mathsf{log}^{t^*}, \mathsf{recs})$, honest nodes stop outputting in future time steps.

Construction. In Figure 1, we construct DailyBFT from a "strongly secure BFT" protocol denoted BFT. A strongly secure BFT protocol is a strengthening of the classical, property-based definition of a state machine replication protocol – this strengthening is necessary to defend against a selective opening attack as we discuss in more detail in our formal proofs in the online full version. We give an overview of the protocol below.

- *BFT virtual nodes and selective opening of committee.* When DailyBFT receives a $\texttt{start}(\mathsf{comm})$ command, if comm contains one or more of its own public keys, then the node is elected as a committee member. In this case, the node will fork a BFT virtual node for each public key in comm that belongs to itself. Here the committee is selectively opened by the environment through the $\texttt{start}(\mathsf{comm})$ command, later our proof will need to leverage the strong security of BFT.
- *Member and non-member basic operations.* Committee members populate their daily logs relying on the BFT protocol, whereas committee non-members count signatures from committee members to populate their logs.
- *Termination.* Nodes implement a termination procedure as follows: whenever an honest committee member receives a `stop` instruction, it inputs a special, signed `stop` transaction to each of its BFT virtual node. As soon as the inner BFT instance outputs a log containing `stop` transactions signed by at least $\lceil |\mathsf{comm}|/3 \rceil$ distinct committee public keys, the log is finalized and output. All transactions after the first $\lceil |\mathsf{comm}|/3 \rceil$ `stop` transactions (with distinct committee public keys) are ignored.

- *Signed daily log hashes.* When committee members output `done`, they also output a signed digest of the final daily log – later, our HybridConsensus protocol will stamp this digest onto the snailchain.

4.3.2 Hybrid Consensus

We now describe our hybrid consensus protocol built on top of a blockchain protocol denoted snailchain. The description of our hybrid consensus relies on snailchain being any protocol that realizes an abstract blockchain (see Section 4.1). For conceptual simplicity, we recommend that the reader first think of snailchain as being Nakamoto's original blockchain protocol [13, 18, 19]. However, later we will actually plug in the Fruitchains [21] protocol as a drop-in replacement of Nakamoto to instantiate snailchain. Since Fruitchains [21] has near-optimal chain quality, the resulting hybrid consensus protocol, instantiated with Fruitchains, will have almost-optimal resilience.

Hybrid consensus consumes multiple instances of DailyBFT where rotating committees agree on daily logs. Hybrid consensus primarily does the following:

- It manages the spawning and termination of DailyBFT instances effectively using snailchain as a global clock that offers weak synchronization among honest nodes;
- Recall that each DailyBFT instance does not ensure security for nodes that spawn too late, since committee members can become corrupt far out in the future at which point they can sign arbitrary tuples. Therefore, hybrid consensus introduces an on-chain stamping mechanism to extend security guarantees to even nodes that spawn late.

Figure 2 is an algorithmic description of the HybridConsensus protocol. Each node maintains a history of all past transcripts denoted history – we assume this for simplicity of formalism, and it can be optimized away in practice. Nodes that newly spawn obtain the historical transcripts instantly (in practice this can be instantiated by having honest nodes offer a history retrieval service). When a new node spawns, it populates its LOG as follows:

- *Matching on-chain valid tuples.* A newly spawned node first identifies all on-chain valid tuples of the form (R, h), where R is the day number and h is the hash of the daily log. Then, the node will search history and identify an appropriate daily log log_R that is consistent with h. The node populates LOG with these daily logs. This on-chain matching process effectively provides a safe mechanism for a newly spawned node to catch up and populate old entries of its output LOG.
- *Through daily offchain consensus.* Once this catch-up process is complete, the node will henceforth rely on DailyBFT instances to further populate remaining entries of its output LOG. In each DailyBFT instance, a node can act as a committee member or non-member. To do this, a node monitors its chain from the snailchain instance. As soon as the chain length exceeds $\mathsf{csize} \cdot R + \lambda$, the R-th day starts, at which point the node inputs `stop` to the previous DailyBFT$[R-1]$ instance (if one exists), and inputs `start`(MinersOf(chain[lower(R) : upper(R)])) to the DailyBFT$[R]$ instance. There is typically a period of overlap during which both DailyBFT$[R-1]$ and DailyBFT$[R]$ instances are running and outputting daily logs simultaneously. When nodes assimilate their daily logs into the final output LOG, they make sure that LOG is always contiguous leaving no gaps in between. Due to the timely termination property of DailyBFT, the old DailyBFT$[R-1]$ will terminate fairly soon at which point the new DailyBFT$[R]$ instance fully takes over.

Concrete instantiation. The recent work Fruitchains [21] showed the following informal theorem: under a corruption budget of $\rho < \frac{1}{2}$ (where ρ is a constant), for any $\tau \geq 0$, and any arbitrarily small positive constant η, there is a blockchain protocol (called Fruitchains) in the

Subprotocol DailyBFT$[R]$

On input `init`: $\ell := 0$, $\mathsf{log} := \emptyset$, henceforth let mykeys be an alias for $\mathcal{G}_{\text{sign}}.\texttt{mykeys}$

On input `stop` : for each $\mathsf{BFT}^{\mathsf{pk}}$ virtual node forked: input $\mathsf{TXs} := \{\{\texttt{stop}\}_{\mathsf{pk}^{-1}}\}$ to $\mathsf{BFT}^{\mathsf{pk}}$

On input `start`(comm): if $\mathsf{mykeys} \cap \mathsf{comm} \neq \emptyset$: $\mathsf{isMember} = \mathsf{true}$, else $\mathsf{isMember} = \mathsf{false}$

Case $\mathsf{isMember} = \mathsf{true}$

For each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: fork a BFT virtual node, and $\mathsf{BFT}.\texttt{start}(\mathsf{pk}, \mathsf{comm})$.
// henceforth this BFT virtual node is denoted $\mathsf{BFT}^{\mathsf{pk}}$

On input TXs: input TXs to each BFT virtual node forked

Let BFT_0 denote the first such BFT virtual node forked

Let $\mathsf{complete}(\mathsf{log}) = \mathsf{true}$ iff log contains `stop` correctly signed by $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm

Every time step t if `start` has been received and `done` has not been output:
Receive output log^* from BFT_0
If $\mathsf{complete}(\mathsf{log}^*)$ then $\mathsf{log}^* :=$ shortest prefix of log^* such that $\mathsf{complete}(\mathsf{log}^*)$
For each $\mathsf{tx} \in \mathsf{log}^* - \mathsf{log}$ that is not a `stop` transaction:
Let $\ell := \ell + 1$, for each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: gossip $\{R, \ell, \mathsf{tx}\}_{\mathsf{pk}^{-1}}$
$\mathsf{log} := \mathsf{log}^*$
If $\mathsf{complete}(\mathsf{log})$: call `Finalize`; else output $\texttt{notdone}(\mathsf{log})$

`Finalize`:
$\mathsf{recs} := \emptyset$, remove all `stop` transactions from log
For each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: let $x := \{R, \mathsf{hash}(\mathsf{log})\}_{\mathsf{pk}^{-1}}$, $\mathsf{recs} := \mathsf{recs} \cup \{x\}$, gossip x
Output $\texttt{done}(\mathsf{log}, \mathsf{recs})$, and stop outputting in future time steps.

Case $\mathsf{isMember} = \mathsf{false}$

On receive $\{R, \ell\}_{\mathsf{pk}^{-1}}$ or $\{R, \ell, \mathsf{tx}\}_{\mathsf{pk}^{-1}}$: add message to history and check the following:

On collect (r, ℓ, tx) and signatures s.t. $r = R$ and $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm signed the tuple correctly:
If $\mathsf{log}[\ell]$ has not been set, let $\mathsf{log}[\ell] := \mathsf{tx}$

On collect (r, ℓ) and signatures s.t. $r = R$ and $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm signed the tuple correctly:
Wait till $\mathsf{log}[: \ell]$ all populated
Output $\texttt{done}(\mathsf{log}, \emptyset)$, and stop outputting in future time steps.

Each time step until `done` is output:
let log' := longest contiguous prefix of log, output $\texttt{notdone}(\mathsf{log}')$

Figure 1 Daily offchain consensus protocol. All signing operations are achieved by calling $\mathcal{G}_{\text{sign}}.\texttt{sign}$ which signs the message tagged with the session identifier.

Protocol $\mathsf{HybridConsensus}^{\lambda}$, parametrized by λ

On `init`: let $R := 0$, $\mathsf{LOG}_s := \emptyset$, $\mathsf{LOG} := \emptyset$, $\mathsf{csize} := \lambda$, let the "warmup window" $W_0 := 2\lambda$.

Mempools:

- Fork an instance of mempool denoted snailpool that stores pending records for snailchain.
- Fork another instance of mempool that stores pending transactions, denoted txpool.
- On input TXs: txpool.`propose`(TXs).

Snailchain: Fork an instance of snailchain, in each time step:

- let chain denote the current local chain
- let $\mathsf{pk} := \mathcal{G}_{\text{sign}}$.`keygen`
- let recs := snailpool.`query`(ExtractRecs(chain$[: -\lambda]$)), input (recs, pk) to snailchain

Preprocess:

Wait till $|\mathsf{chain}| \geq W_0$

$L :=$ Find in history maximal, ordered list of (R, log_R) tuples such that R is incrementing with no gaps, and $(R, \mathsf{hash}(\mathsf{log}_R))$ is on-chain valid w.r.t. chain

$\mathsf{LOG} := \mathsf{log}_1 || \mathsf{log}_2 || \ldots || \mathsf{log}_{|L|}$

Daily Offchain Consensus:

Let $R := |L|$

Loop:

Wait till $|\mathsf{chain}| \geq \mathsf{upper}(R+1) + \lambda$, let $R := R + 1$

Let $\mathsf{comm}_R :=$ MinersOf(chain[lower(R) : upper(R)]) where MinersOf parses each block as $(\mathsf{recs}_i, \mathsf{pk}_i)$ and returns a list containing all pk_is

If an instance $\mathsf{DailyBFT}[R-1]$ was started, $\mathsf{DailyBFT}[R-1]$.`stop`

$\mathsf{DailyBFT}[R]$.`start`(comm_R)

Each time step: let TXs := txpool.`query`(LOG), input TXs to $\mathsf{DailyBFT}[R]$

Output: In each time step: let R denote the current day. Let $\mathsf{isdone}(r) = \mathsf{true}$ if $\mathsf{DailyBFT}[r]$ has output `done` in this or earlier time steps.

- If $\mathsf{DailyBFT}[R-1]$ outputs `done`(log_{-1}, recs) in this time step: $\mathsf{LOG}_s := \mathsf{LOG}_s || \mathsf{log}_{-1}$, snailpool.`propose`(recs)
- Let log_{-1} and log be the output logs of $\mathsf{DailyBFT}[R-1]$ and $\mathsf{DailyBFT}[R]$ in this time step respectively (or $\emptyset$ if nothing is output)
- If $\mathsf{isdone}(R-1)$: $\mathsf{LOG} := \mathsf{LOG}_s || \mathsf{log}$; else $\mathsf{LOG} := \mathsf{LOG}_s || \mathsf{log}_{-1}$. Output LOG

On-chain valid: A tuple (R, h) is on-chain valid w.r.t chain iff the following holds

- For at least $\frac{1}{3}$ fraction of distinct $\mathsf{pk} \in$ MinersOf(chain[lower(R), upper(R)]): $\{R, h\}_{\mathsf{pk}^{-1}}$ is the first occurrence in chain$[: -\lambda]$ where pk signed some tuple of the form $(R, \_)$.

Figure 2 Main HybridConsensus protocol. A newly spawned, honest node starts running this protocol. We assume history is the set of all historical transcripts sent and received. We assume that message routing to subprotocol instances is implicit: whenever any subprot[sid] instance is forked, history[subprot[sid]] and protocol messages pertaining to subprot[sid] are automatically routed to the subprot[sid] instance.

proof-of-work model that satisfies consistency, $(1-\eta)(1-\rho)$-chain quality, and $(\frac{1}{c_0\Delta}, \frac{1}{c_1\Delta})$ for appropriate constants c_0 and c_1. As mentioned, although the original Fruitchains work assumes a pre-determined and fixed n, it is straightforward to extend their result to the case when n can vary by a known constant factor, and moreover the protocol knows only an estimate of n that can be off by a known constant factor (see our full version for details [20]).

We will instantiate hybrid consensus with Fruitchains as the underlying snailchain, since Fruitchains has almost ideal chain quality. If we do so, and assuming that the agility parameter τ is sufficiently large, we have the following theorem whose proof is deferred to the online full version [20].

▶ **Theorem 4** (Hybrid consensus over Fruitchains)**.** *Assume the existence of a proof-of-work oracle and one-way functions. Further, assume that $\tau > C\lambda\Delta$ for some appropriate constant C, and assume that the adversary is subject to $\rho < \frac{1}{3}$ corruption budget where ρ is a constant. Then, there exists a permissionless consensus protocol that achieves consistency and $(T_{\text{warm}}, T_{\text{confirm}})$-liveness for $T_{\text{warm}} = O(\lambda\Delta)$ and $T_{\text{confirm}} = O(\lambda\delta)$.*

5 Related Work

Although the idea of combining permissionless consensus and permissioned consensus has been discussed in the community (e.g., the recent work by Decker et al. [9] and the concurrent and independent work ByzCoin [15]), these other works are of a different nature. The concurrent work ByzCoin [15] (Usenix Security'16) does not remove trailing unstabilized blocks from the blockchain for committee election; consequently the nodes may not agree on the committee (and thus consistency can be broken). Although their paper claims to tolerate $\frac{1}{3}$ corruption, the claim is incorrect – due to a well-known selfish mining attack, the adversary can control up to a half of the blocks under $\frac{1}{3}$ corruption, and thus the adversary will control a half of the committee in Byzcoin. Indeed, the authors of Byzcoin themselves acknowledged flaws of their protocol in subsequent blog posts [1, 2], and acknowledged that they need to rely on some of our ideas to fix their incorrect claim. Even with the fixes in their blog posts, it remains unclear what properties their protocol guarantees since they do not offer formal proofs of security – for example, their protocol overlooks the issue of posterior corruption which, as we show, requires non-trivial techniques to handle. The prior work by Decker et al. [9] does not aim to achieve responsiveness which is our primary goal. Specifically, Decker et al. [9] relies on classical consensus such as PBFT to vote on each block as it is mined – thus they have to wait for at least one "block interval". Moreover, Decker et al.'s blockchain variant [9] suffers from the same type of selfish mining attack [12] that Nakamoto's blockchain is prone to. Thus, they cannot tolerate $\frac{1}{3}$ corruption due to degraded chain quality.

Earlier works on permissioned consensus have also considered group reconfiguration. For example, Vertical Paxos [16] and BFT-SMART [8] allow nodes to be reconfigured in a dynamic fashion. These works consider group reconfiguration for a related but somewhat different purpose. A line of research starting from Dolev et al. [10] 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: 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 [10] show that the number of actual rounds must be proportional to the actual number of faulty processors – these lower bounds work only for deterministic protocols, and thus they are not applicable in our setting.

The concurrent work SPETRE [22] also aims to achieve responsiveness in permissionless consensus by relaxing the consistency requirement. The subsequent work Solidus [4] aims to achieve responsive permissionless consensus and additionally obtain incentive compatibility guarantees – their paper does not precisely articulate under what model their protocol retains security, but all the lower bounds we prove should apply to their setting, so even if their protocol could be proven secure, it would require the same type of restrictions on the model that we impose. Their incentive compatibility guarantees are heuristic and without a formal proof [3] – we also point out for regarding incentive compatibility, by combining the reward distribution mechanism proposed in Fruitchains [21] with Hybrid Consensus, it is straightforward how to distribute rewards in a manner that *provably* achieves ϵ-Nash equilibrium against any $\frac{1}{3} - \epsilon'$ coalition. This means that no attacker wielding less than $\frac{1}{3}$ hashpower can gain more than ϵ fraction more rewards than its fair share even if he is allowed to arbitrarily deviate from the honest protocol. We leave it as an open question how to *provably* achieve *strict* Nash equilibrium.

Acknowledgements. We would like to thank the reviewers for their insightful comments, especially Christian Cachin whose feedback was critical in helping us improve the paper.

References

1 Byzcoin: Securely scaling blockchains. `http://hackingdistributed.com/2016/08/04/byzcoin/`.

2 Untangling mining incentives in bitcoin and byzcoin. `http://bford.github.io/2016/10/25/mining/`.

3 Personal communication with Kartik Nayak and Ling Ren.

4 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. *CoRR*, abs/1612.02916, 2016.

5 Marcin Andrychowicz and Stefan Dziembowski. Pow-based distributed cryptography with no trusted setup. In *CRYPTO*, pages 379–399, 2015.

6 Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, 1994.

7 Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, pages 361–377, 2005.

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

9 Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *ICDCN*, 2016.

10 Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, October 1990.

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

12 Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, 2014.

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

14 Jonathan Katz, Andrew Miller, and Elaine Shi. Pseudonymous secure computation from time-lock puzzles. *IACR Cryptology ePrint Archive*, 2014:857, 2014.

15 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Usenix Security*, 2016.

16 Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In *PODC*, pages 312–313, 2009.

17 Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

18 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

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

20 Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Technical report version, `https://eprint.iacr.org/2016/917`, 2016.

21 Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.

22 Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.