Optimal Flexible Consensus and its Application to Ethereum

Joachim Neu
Stanford University
jneu@stanford.edu

Srivatsan Sridhar
Stanford University
svatsan@stanford.edu

Lei Yang
Massachusetts Institute of Technology
leiy@csail.mit.edu

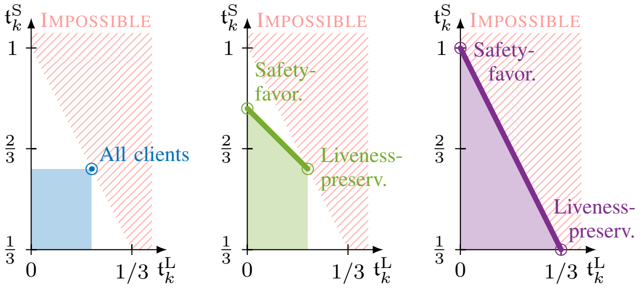
David Tse
Stanford University
dntse@stanford.edu

***Abstract*—Classic BFT consensus protocols guarantee safety and liveness for *all* clients if fewer than one-third of replicas are faulty. However, in applications such as high-value payments, some clients may want to prioritize safety over liveness. *Flexible* consensus allows *each* client to opt for higher safety resilience, albeit at the expense of reduced liveness resilience. We present the first construction that allows *optimal safety–liveness tradeoff for every client simultaneously*. This construction is modular and is realized as an add-on applied on top of an existing consensus protocol. The add-on consists of an *additional* round of voting and *permanent* locking done by the replicas, to sidestep a sub-optimal quorum-intersection-based constraint present in previous solutions. We adapt our construction to the existing Ethereum protocol to derive optimal flexible confirmation rules that clients can adopt unilaterally *without* requiring system-wide changes. This is possible because existing Ethereum protocol features can double as the extra voting and locking. We show an implementation using Ethereum's consensus API.**

1. Introduction

1.1. Flexible Consensus

A state-machine replication (SMR) *consensus protocol* has two groups of participants: *clients* and *replicas*. Continuously, clients input *transactions* to the replicas. The replicas partake in a distributed algorithm to develop a common ordering among the transactions, and report back to the clients. At any time, each client outputs a *log* which is the ordered sequence of transactions it deems *confirmed* so far. Two desiderata for consensus protocols are: *safety*, meaning that logs are consistent across clients and across time; and *liveness*, meaning that transactions submitted to all replicas eventually appear in all clients' logs. Byzantine-fault tolerant (BFT) consensus protocols guarantee these properties if the fraction $\mathfrak{f}$ of *adversary* replicas out of all n replicas is not too large, where adversary replicas may deviate from the protocol in any arbitrary and coordinated fashion, and non-adversary *honest* replicas follow the protocol as specified.[1] Based on terminology of [1], [2], [3], [4], [5], a protocol's *safety resilience* $\mathfrak{t}^{\mathrm{S}}$ is defined as the maximum fraction of adversary replicas the protocol can tolerate while still guaranteeing safety. Similarly, the *liveness resilience* $\mathfrak{t}^{\mathrm{L}}$ is the maximum fraction of adversary replicas the protocol can tolerate while guaranteeing liveness. Classical *PBFT-style* protocols [6], [7], [8], [9], [10], [11] provide balanced safety and liveness resiliences $\mathfrak{t}^{\mathrm{L}} = \mathfrak{t}^{\mathrm{S}} = \frac{1}{3}$ [6], which is the highest that both the resiliences can simultaneously be in the partially-synchronous setting [12] of interest here.



(a) PBFT-style prot. (b) FBFT, SFT-DiemBFT (c) Our OFlex prot.

Figure 1. Pareto front (•, ▬, ▬) and region (🟦, 🟩, 🟪) of liveness-/safety-resilience pairs $(\mathfrak{t}_k^{\mathrm{L}}, \mathfrak{t}_k^{\mathrm{S}})$ that each client k can choose from, for three flexible consensus protocols under partial synchrony. Recall the classic impossibility result $2\mathfrak{t}_k^{\mathrm{L}} + \mathfrak{t}_k^{\mathrm{S}} \leq 1$ (▨) [6], [13]. (a) PBFT-style protocols [6], [8], [9], [10], [11] use a system-wide replica quorum $\mathfrak{q}$. For any fixed $\mathfrak{q}$ ($= \frac{4}{5}$ here), these protocols have no flexibility (•) to support liveness-preserving clients and safety-favoring clients *simultaneously*; all clients operate at $(1-\mathfrak{q}, 2\mathfrak{q}-1)$. (b) FBFT [14] and SFT-DiemBFT [15] additionally use client-side confirmation quorums $\mathfrak{q}_k \in [\mathfrak{q}, 1]$ to allow for some flexibility (▬) between the liveness-preserving client at $(1-\mathfrak{q}, 2\mathfrak{q}-1)$ for $\mathfrak{q}_k = \mathfrak{q}$ and the most safety-favoring client at $(0, \mathfrak{q})$ for $\mathfrak{q}_k = 1$. (c) Our OFlex protocols with only client-side confirmation quorums $\mathfrak{q}_k \in [\frac{2}{3}, 1]$ provide *optimal* flexibility (▬ vs. ▨) between the liveness-preserving client at $(\frac{1}{3}, \frac{1}{3})$ for $\mathfrak{q}_k = \frac{2}{3}$ and the most safety-favoring client at $(0, 1)$ for $\mathfrak{q}_k = 1$.

However, users (*i.e.*, clients) of a system may not desire equal resilience to safety and liveness faults [1], [2], [3], [4], [5]. In fact, different clients may prefer different (liveness-/safety-)*resilience pairs* altogether [14], [15]![2] For instance, for a distributed ledger such as a cryptocurrency, clients

JN, SS, LY are listed alphabetically.

[1] Since no guarantee can be given for logs produced by adversary clients following arbitrary logic, the focus is on clients that follow the protocol.

[2] Note that like in [14], 'clients' are a logical abstraction for whenever physical users require different resilience pairs. Thus, one physical user can achieve different resilience pairs at different points in time or to confirm different transactions, and would then manifest as multiple clients.

who perform high-value transfers may prefer a higher safety resilience,[3] even at the expense of lower liveness resilience, because business lost during a system downtime may cause less harm than a double spend. In the same vein, by definition, liveness only breaks if transaction inclusion is denied *forever* [21]. Thus, liveness can be 'restored' relatively easily, *e.g.*, through exogenous reconfiguration and removal of adversary replicas. In contrast, safety already breaks if inconsistency occurs *ever*. These considerations suggest that guaranteeing consensus in the *high-safety regime*, where $\mathfrak{t}^{\mathrm{S}} \geq \mathfrak{t}^{\mathrm{L}}$, is of particular interest to some *safety-favoring* clients. At the same time, other *liveness-preserving* clients may want to retain $\mathfrak{t}^{\mathrm{L}} = \frac{1}{3}$, even at the expense of not increasing $\mathfrak{t}^{\mathrm{S}}$ beyond $\frac{1}{3}$.[4]

This motivates *flexible consensus* [14] (also called *strengthened fault tolerance* [15]), where each client k chooses a resilience pair $(\mathfrak{t}^{\mathrm{L}}_k, \mathfrak{t}^{\mathrm{S}}_k)$ with $\mathfrak{t}^{\mathrm{S}}_k \geq \mathfrak{t}^{\mathrm{L}}_k$, and the consensus properties are guaranteed for all clients who have chosen adequate resiliences. That is, for all clients k, k' if the adversary fraction $\mathfrak{f} \leq \mathfrak{t}^{\mathrm{S}}_k$ and $\mathfrak{f} \leq \mathfrak{t}^{\mathrm{S}}_{k'}$, their logs are guaranteed to be consistent across time (safety); and transactions submitted to all replicas eventually appear in the log of every client k with $\mathfrak{f} \leq \mathfrak{t}^{\mathrm{L}}_k$ (liveness).

1.2. Quest for Optimal Flexible Consensus

Given the flexible consensus problem, it is natural to ask: *What is the 'maximum' flexibility a protocol can provide?* The classic impossibility result $2\mathfrak{t}^{\mathrm{L}}_k + \mathfrak{t}^{\mathrm{S}}_k \leq 1$ [12], [13] shows that we cannot hope to do better than for each client to achieve a resilience pair $(\mathfrak{t}^{\mathrm{L}}_k, \mathfrak{t}^{\mathrm{S}}_k)$ on the straight line between $(0, 1)$ and $(\frac{1}{3}, \frac{1}{3})$, shown in Fig. 1 (▨). *But is there a protocol which allows clients to achieve all such pairs simultaneously, or are there some further limits to flexibility?*

In typical PBFT-style protocols, all clients have the same pair of resiliences $\mathfrak{t}^{\mathrm{L}}_k = \mathfrak{t}^{\mathrm{S}}_k = \frac{1}{3}$. In that sense, these protocols are a degenerate case of flexible consensus where the set of resilience pairs $(\mathfrak{t}^{\mathrm{L}}_k, \mathfrak{t}^{\mathrm{S}}_k)$ that clients can simultaneously 'choose' from is the singleton $\{(\frac{1}{3}, \frac{1}{3})\}$. In these protocols, one could vary the quorum size $\mathfrak{q}$ used in the protocol such that for any given quorum $\mathfrak{q}$, all clients achieve a single resilience pair $(\mathfrak{t}^{\mathrm{L}}, \mathfrak{t}^{\mathrm{S}})$ that is different from $(\frac{1}{3}, \frac{1}{3})$. Fig. 1(a) shows one such achievable pair. However, these protocols still have *no flexibility* to support liveness-preserving and safety-favoring clients simultaneously.

To add some flexibility, for any fixed system-wide *replica quorum* $\mathfrak{q}$, FBFT [14] and SFT-DiemBFT [15] (Fig. 1(b)) allow every client k to opt for higher $\mathfrak{t}^{\mathrm{S}}_k$ by using a higher client-specific *confirmation quorum* $\mathfrak{q}_k \in [\mathfrak{q}, 1]$ when confirming its output log. This construction results in $\mathfrak{t}^{\mathrm{S}}_k = \mathfrak{q}_k + \mathfrak{q} - 1$ and $\mathfrak{t}^{\mathrm{L}}_k = 1 - \mathfrak{q}_k$, allowing the trade-off exemplified in Fig. 1(b), between $(0, \mathfrak{q})$ for the most safety-favoring client with $\mathfrak{q}_k = 1$, and $(1 - \mathfrak{q}, 2\mathfrak{q} - 1)$ for the liveness-preserving client with $\mathfrak{q}_k = \mathfrak{q}$. By tuning the replica quorum $\mathfrak{q}$, it is possible to run *different instances* of the protocol which support *different regions of resilience pairs*, but no single instance can simultaneously support a maximally safety-favoring client at $(0, 1)$ and a liveness-preserving client at $(\frac{1}{3}, \frac{1}{3})$.

TABLE 1. Realization of key mechanisms in different protocols of the OFlex family (Sec. 1.3): addition of logic vs. re-use of existing logic.

	Replica logic (system-wide)		Client logic (local)
Construction	**Post-vote**	**Perma-lock**	**Confirmation rule**
Generic (add-on to any protocol)	**add** post-votes external to the base protocol	**add** constraint on post-votes	**new** (quorum $\mathfrak{q}_k$ fraction of post-votes)
Streamlined PBFT-style (*e.g.*, OFlex-Streamlet)	**existing** votes (streamlining)	**add** constraint on votes	**new** (quorum $\mathfrak{q}_k$ fraction of votes)
OFlex conf. rules for Ethereum	**existing** votes (streamlining)	**existing** performance optimization	**new** (quorum $\mathfrak{q}_k$ fraction of votes)

1.3. Contribution: OFlex Family of Resilience-Optimal Flexible Consensus Protocols

We present the OFlex family of flexible consensus protocols, the first *resilience-optimal* flexible consensus protocols. Each protocol in this family can support all clients on the straight line between $(0, 1)$ and $(\frac{1}{3}, \frac{1}{3})$ simultaneously (Fig. 1(c)). Each protocol in this family is obtained by applying a modification to an existing consensus protocol, and our constructions are applicable to a broad class of existing protocols.

Our constructions extend the approach from FBFT and SFT-DiemBFT [14], [15] to decouple the consensus protocol $\mathbf{\Pi}$ into two phases: an interactive replica logic Π that depends only on a system-wide quorum $\frac{2}{3}$ (unaware of clients' choices of resilience pairs), and a confirmation rule $\mathcal{C}$ run locally by clients that additionally depends on client-specific quorums $\mathfrak{q}_k \geq \frac{2}{3}$. The key new idea in OFlex is to augment the replica logic with an additional round of voting (called *post-voting*) and permanent locking (*perma-locking*). Replicas hold on to their perma-lock *permanently* and never perma-lock or post-vote anything inconsistent with their perma-lock, regardless of how powerful the adversary is. In contrast to FBFT, where safety is limited by a quorum-intersection-based constraint between a large client quorum and a small replica quorum ($\mathfrak{t}^{\mathrm{S}}_k \leq \mathfrak{q}_k + \mathfrak{q} - 1$), OFlex due to the perma-lock and post-vote only needs a constraint between two large client quorums ($\mathfrak{t}^{\mathrm{S}}_k \leq 2\mathfrak{q}_k - 1$). This enhances the safety resilience of OFlex. Even though the perma-lock appears to restrict a replica's post-votes, OFlex in fact preserves FBFT's liveness resilience of $\mathfrak{t}^{\mathrm{L}}_k = 1 - \mathfrak{q}_k$. This is because for all resilience pairs of interest (Fig. 1(c)), the liveness resilience $\mathfrak{t}^{\mathrm{L}}_k \leq \frac{1}{3}$, and when the adversary fraction is smaller than this, all replicas' perma-locks are

[3] For example, in Ethereum as of 06/2023, a single organization (Lido) controls more than one-third of replicas, causing concerns about a possible coordinated failure, *e.g.*, from technical malfunction or social engineering, exceeding Ethereum's one-third resilience [16], [17], [18], [19], [20].

[4] In contrast, the regime $\mathfrak{t}^{\mathrm{L}} > \mathfrak{t}^{\mathrm{S}}$ is subsequently disregarded because its 'live but inconsistent' logs are of questionable utility [1], [15].

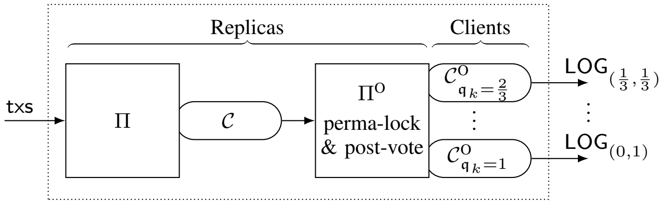

Replicas
Clients
txs
Π
$\mathcal{C}$
Π^{O}
perma-lock
& post-vote
$\mathcal{C}^{\mathrm{O}}_{\mathsf{q}_k=\frac{2}{3}}$
⋮
$\mathcal{C}^{\mathrm{O}}_{\mathsf{q}_k=1}$
$\mathsf{LOG}_{(\frac{1}{3},\frac{1}{3})}$
⋮
$\mathsf{LOG}_{(0,1)}$

Figure 2. Block diagram of our generic OFlex construction (Sec. 3). Starting from an *unmodified* consensus protocol with replica logic Π and confirmation rule $\mathcal{C}$ with resiliences $\mathsf{t}^{\mathrm{L}} = \mathsf{t}^{\mathrm{S}} = \frac{1}{3}$, the construction applies a round of permanently locking (*perma-locking*) and voting (*post-voting*) by replicas (Π^{O}) on the log output by $(\Pi, \mathcal{C})$, the unmodified protocol, followed by client-specific confirmation quorums q_k on the post-vote messages for confirmation ($\mathcal{C}^{\mathrm{O}}$).

consistent with each other, so honest replicas continue voting in a manner that ensures liveness.

We present three OFlex protocol constructions, that differ in how they implement perma-locking and post-voting (see Tab. 1 for a summary):

- The *generic OFlex construction (Sec. 3, Fig. 2)* adds both perma-locking and post-voting as separate replica-side logic that acts on the output log of any $(\frac{1}{3}, \frac{1}{3})$-resilient consensus protocol for partial synchrony to make it optimally flexible in a closed-box manner. *This is a modular addition to the replica logic.*
- To *modify* textbook *PBFT-style protocols for OFlex (Sec. 4)*, post-voting requires no replica-side changes, because existing votes can be 'reused' as post-votes (cf. streamlining [9], [10], [11]). Only perma-locking needs to be added by means of an additional constraint for replicas while voting. We demonstrate the application of OFlex to Streamlet [11] to obtain OFlex-Streamlet in Sec. 4, and the technique readily carries over to other PBFT-style protocols. *This non-modular modification of the replica logic preserves the amount of communication of the original PBFT-style protocol.*
- Many *implementations* of PBFT-style protocols employ a performance optimization that de-facto already implements the perma-locking mechanism. For OFlex variants of such protocols, *no replica-side changes are needed.* An example is Ethereum (which is based on Casper [9], a PBFT-style protocol), for which we derive optimally flexible confirmation rules (Sec. 5) that clients can adopt without requiring any system-wide changes. We demonstrate an implementation using Ethereum's consensus API, and study the practical safety vs. confirmation latency trade-offs, as summarized in Fig. 3.

Our OFlex protocols also have other practically relevant qualities such as stronger consistency guarantees between clients with different safety resiliences, accountable safety, and the possibility of preserving safety even after external repair ('social consensus'). We discuss these in Sec. 7.

1.4. Methods

1.4.1. Generic OFlex Construction (Sec. 3). Our generic OFlex construction best highlights the key ideas that are used in all three of our constructions. The generic construction has two stages (Fig. 2). First, any off-the-shelf $(\frac{1}{3}, \frac{1}{3})$-resilient consensus protocol is invoked in a closed-box manner to sequence incoming transactions into a log. In the second stage, whenever the log output by the first stage changes, lock *permanently* (*perma-lock*) on the new log (only if the new log extends the earlier perma-lock). They then cast a post-vote vote for the latest perma-locked log. Finally, clients use local quorums $\mathsf{q}_k \geq \frac{2}{3}$ among the second stage's post-vote votes to decide which log entries can be confirmed.

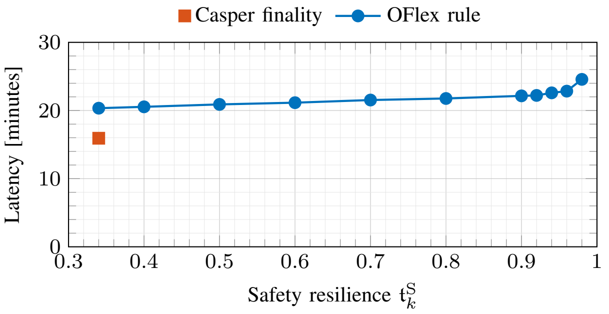


Figure 3. Average latency to confirm Ethereum mainnet blocks between slots 5,970,000 and 6,970,000 (*i.e.*, most recent 1,000,000 slots as of this writing, corresponding to March 10, 2023 to July 27, 2023) by Casper finality [9] (the confirmation rule as per Ethereum's specifications) and by OFlex confirmation rules (Sec. 5) with different safety levels. See Sec. 5.3 for details of the setup. OFlex incurs approx. 4.25 min (21 slots) increase in latency over Casper finality, even at the same safety resilience $\mathsf{t}^{\mathrm{S}}_k = \frac{1}{3}$, due to post-voting. However, beyond that, OFlex achieves extremely high safety resilience while incurring only a modest increase in latency. Each client can operate on any point along the blue line to match its desired trade-off between safety and latency.

Unlike in FBFT, the client-local confirmation rules are not applied to votes preexistent in PBFT-style protocols, but rather to votes from an *additional* round of post-votes which respect the perma-lock (Fig. 2). This is crucial to achieve the optimal quorum-interesection constraint $\mathsf{t}^{\mathrm{S}}_k = 2\mathsf{q}_k - 1$ because an honest replica will never post-vote two inconsistent logs, *no matter what the first stage outputs*.

This generic construction applies to any $(\frac{1}{3}, \frac{1}{3})$-resilient consensus protocol such as [8], [22], [23], [24], [25], [26], including DAG-based protocols [27], [28], [29], [30]. While it also applies to streamlined PBFT-style protocols, we also provide a specific construction for them.

1.4.2. OFlex PBFT-Style Protocols (Sec. 4). With the insight that optimal flexibility can be achieved with an extra round of perma-lock and post-vote, we revisit PBFT-style protocols and modify them for optimal flexibility.

To this end, recall that PBFT proceeds in views, where in each view a batch of new transactions is proposed, and after meeting quorum in multiple voting phases, the proposal is confirmed. Some recent PBFT-style protocols [9], [10], [11] streamline these phases into a streamlined protocol, where in each round a proposal and voting takes place. Then, votes simultaneously serve for different voting phases with respect to different proposals [10, Sec. 5]. It is easy to see that the additional round of voting (post-votes) required for OFlex is easily integrated into such streamlined PBFT-style protocols using subsequent votes. Besides the system-wide replica quorum $\mathfrak{q} = 2/3$, clients impose client-local confirmation quorums $\mathfrak{q}_k$ for the additional voting round.

On the other hand, a mechanism like perma-lock is not readily found in the *textbook version* of PBFT-style protocols. But the generic OFlex construction suggests the following modification which turns out to be sufficient: in the place where the original PBFT-style protocol would have confirmed a block (using a system-wide confirmation rule), replicas perma-lock the respective block (and never vote for anything inconsistent with their perma-lock).

1.4.3. OFlex Confirmation Rules for Ethereum (Sec. 5). While a mechanism like perma-lock is not commonly found in the pseudo-code of popular PBFT-style protocols, it is, however, already present in many *implementations* of PBFT-style protocols in the form of an unrelated performance optimization. For instance, Ethereum replicas (called 'validators') will forever lock on a block they view as 'finalized' according to the rules of Casper [9], [31]. They will then refuse to consider any conflicting block going forward, even if the conflicting block has also received enough votes to become finalized (such conflicting finalizations can only occur if more than $1/3$ of replicas are adversary). This 'pruning' of the block tree improves validators' computational efficiency.

Since Ethereum has already implemented an equivalent of perma-locking, and post-voting is easily accommodated in a PBFT-style protocol like Casper, as discussed above, applying the OFlex construction to Ethereum requires no system-wide changes to the replica logic. Client-side changes to the confirmation logic suffice.

1.5. Outline

After recapitulating the model and problem formulation in Sec. 2, we describe and prove secure the generic OFlex construction in Sec. 3. We show in Sec. 4 how to use OFlex's key insight to modify PBFT-style protocols to provide optimal flexibility. In Sec. 5, we apply OFlex to obtain optimal-resilience flexible-consensus confirmation-rules for Ethereum, and report on an implementation thereof. We conclude with a comparison to related work in Sec. 6, and a discussion of extensions of OFlex in Sec. 7.

2. Model and Problem Formulation

SMR consensus is run by n *replicas* and a group of *clients*. Out of these, f replicas are *adversary*[5] ('Byzantine faults'; assumed to be chosen at the start of the execution before global randomness is drawn), and the remaining replicas are *honest* and follow the specified *replica logic* Π to receive transactions from the environment, interact with other replicas, and send updates to clients. At all times τ, clients (indexed by k) use a *confirmation rule*, which only involves local computation on the updates received from replicas, to confirm and output a log LOG_k^τ of transactions. In classical consensus, there is one confirmation rule $\mathcal{C}$. In flexible consensus, each client k may use a different confirmation rule $\mathcal{C}_k$. The replica protocol and the confirmation rule(s) together are called the *consensus protocol* $\mathbf{\Pi}$.

Communication among replicas and from replicas to clients is authenticated using digital signatures. We assume the *eventual-synchrony* variant of the *partially-synchronous* network model [6], *i.e.*, there is an adversarially chosen *global stabilization time* (GST), *unknown* to the protocol, before and after which the network is *asynchronous* and *synchronous*, respectively. During asynchrony, the adversary can delay messages arbitrarily. During synchrony, the adversary can delay messages up to a delay bound Δ, *known* to the replicas and clients. Protocols may instruct replicas to *gossip* received messages, so that after GST every message received by any honest replica by τ is received by all honest replicas by $\tau + \Delta$.

Notation: Let $\mathsf{LOG} \preceq \mathsf{LOG}'$ denote that LOG is a prefix of or identical to LOG'. Two logs are *consistent* iff $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$.

We now formally define the flexible consensus problem.[6]

Definition 1. Protocol $\mathbf{\Pi} = (\Pi, \{\mathcal{C}_k\})$ provides *flexible consensus* with resilience pairs $\left\{(t_k^{\mathrm{L}}, t_k^{\mathrm{S}})\right\}$ iff:

- **Liveness:** For every tx input to all honest replicas, eventually, for all clients k with $f \leq t_k^{\mathrm{L}}$, $\mathsf{tx} \in \mathsf{LOG}_k$.
- **Safety:** For all clients k, k' with $f \leq \min\{t_k^{\mathrm{S}}, t_{k'}^{\mathrm{S}}\}$, for all times τ, τ', LOG_k^τ and $\mathsf{LOG}_{k'}^{\tau'}$ are consistent.

We use three special cases of the above definition. We refer to the case where for all k, $t_k^{\mathrm{L}} = t^{\mathrm{L}}$, $t_k^{\mathrm{S}} = t^{\mathrm{S}}$, as $(t^{\mathrm{L}}, t^{\mathrm{S}})$*-consensus*. We call the further specialized case $t^{\mathrm{L}} = n/3 - 1$, $t^{\mathrm{S}} = n/3$ *classical consensus*. The third case is the following:

Definition 2. A flexible consensus protocol provides *optimal flexible consensus* if it supports any set $\left\{(t_k^{\mathrm{L}}, t_k^{\mathrm{S}})\right\}$ of resilience pairs with for all k, $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$ and $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$.

3. Generic OFlex Construction

We describe how to construct an optimal high-safety flexible consensus protocol from any classical consensus protocol in a generic closed-box manner. The key ingredient is new replica logic we add, which we call an *optimal-flexibility (OFlex) gadget*: an additional round of voting (called 'post-vote' to distinguish from votes in the classical protocol) in which replicas follow a permanent lock ('perma-lock') constraint.

[5] To state results with precision, we henceforth denominate $f, q, t^{\mathrm{L}}, t^{\mathrm{S}}$ in number of replicas (instead of fractions $\mathfrak{f}, \mathfrak{q}, \mathfrak{t}^{\mathrm{L}}, \mathfrak{t}^{\mathrm{S}}$ as in Sec. 1).

Algorithm 1 Optimal-flexibility (OFlex) gadget $\mathbf{\Pi}^{\mathrm{O}}$

```
 1: ▷ Replica-side logic $\Pi^O$
 2: on INIT()
 3:     $\mathcal{C}$.INIT()                ▷ Replica runs base protocol confirmation rule
 4:     permalock ← []           ▷ Initialize perma-lock to the empty log
 5: forever
 6:     LOG ← $\mathcal{C}$.GETLOG()
 7:     if permalock $\prec$ LOG         ▷ LOG (strictly) extends perma-lock
 8:         permalock ← LOG                              ▷ Perma-lock
 9:         Broadcast ⟨postvote, LOG⟩ to all clients     ▷ Post-vote LOG

10: ▷ Client-side confirmation rule $\mathcal{C}^O$ for resilience pair $(t^{\mathrm{L}}_k, t^{\mathrm{S}}_k)$
11: on INIT($q_k$) where $q_k$ shall satisfy $(n + t^{\mathrm{S}}_k)/2 < q_k \leq n - t^{\mathrm{L}}_k$
12:     Set local quorum size $q_k$
13:     postvotes ← $\emptyset$                      ▷ Set of received post-votes
14: on receiving ⟨postvote, LOG⟩ from replica $i$
15:     postvotes ← postvotes $\cup$ $\{(\mathsf{LOG}, i)\}$        ▷ Record post-vote
16: on GETLOG()
17:     ▷ Confirm LOG* if $q_k$ replicas post-voted LOG* or an extension
18:     LOG* ← $\arg\max_{\mathsf{LOG}}$ |LOG| subject to postvotes contains at
        least $q_k$ pairs $(\mathsf{LOG}_i, i)$ for distinct $i$ with some $\mathsf{LOG}_i \succeq \mathsf{LOG}$
19:     return LOG*
```

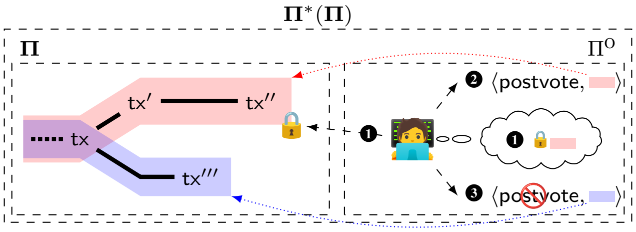


Figure 4. Illustration of a replica's role in OFlex gadget Π^{O} (Alg. 1), which can augment any classical consensus protocol $\mathbf{\Pi}$ to make an optimal flexible consensus protocol $\mathbf{\Pi}^*(\mathbf{\Pi})$. * *Protocol rules:* ❶ Honest replicas perma-lock the log they obtain from $\mathbf{\Pi}$ iff it extends their previous perma-lock, ❷ post-vote the log that they perma-lock, and ❸ never post-vote any log that does not extend their perma-locked log. * *Safety intuition:* A client k with $q_k = n$ confirms a log only when it sees n post-votes for the log. Due to steps ❶ to ❸, no conflicting log can receive n post-votes and be confirmed by another client k' with $q_{k'} = n$, *unless all replicas are adversary*, implying $t^{\mathrm{S}}_k = t^{\mathrm{S}}_{k'} = n - 1$.

3.1. Construction

A high-level block diagram of this construction is shown in Fig. 2. We start with any protocol $\mathbf{\Pi} = (\Pi, \mathcal{C})$ that provides $(n/3 - 1, n/3)$-consensus. We call $\mathbf{\Pi}$ the *base protocol*. Replicas running the replica protocol Π additionally run the OFlex gadget protocol Π^{O} (which implements 'perma-lock and post-vote'). Clients obtain their flexible logs by running new confirmation rules $\mathcal{C}^{\mathrm{O}}_k$ on messages received as part of Π^{O} from replicas. The entire construction, encompassing $\mathbf{\Pi}$, Π^{O}, and the family of confirmation rules $\{\mathcal{C}_k\}$, is denoted $\mathbf{\Pi}^*(\mathbf{\Pi})$.

Pseudocode of Π^{O} and $\mathcal{C}^{\mathrm{O}}_k$ is given in Alg. 1. The replicas' role in Π^{O} is illustrated in Fig. 4. Each replica uses the confirmation rule of the base protocol $\mathcal{C}$ to obtain a confirmed log at each time step. Each replica maintains a perma-locked log. Whenever the replica sees an update to the output log of the base protocol according to the confirmation rule $\mathcal{C}$, the replica checks if the new log *strictly extends* its perma-locked log (Alg. 1 l. 7). Only if so, the replica updates its perma-lock to the new log (Fig. 4 ❶), and broadcasts a 'post-vote' message for the new log (Fig. 4 ❷ ❸).

The client's confirmation rule $\mathcal{C}^{\mathrm{O}}_k$ is parameterized by a client-specific confirmation quorum q_k, chosen as $(n + t^{\mathrm{S}}_k)/2 < q_k \leq n - t^{\mathrm{L}}_k$ to satisfy the client's resilience pair $(t^{\mathrm{L}}_k, t^{\mathrm{S}}_k)$. The client confirms a log LOG if at least q_k replicas post-vote LOG or a log that extends LOG (Alg. 1 l. 18).

3.2. Security Analysis

We will prove that $\mathbf{\Pi}^*(\mathbf{\Pi})$, *i.e.*, our OFlex gadget applied to any classical consensus protocol $\mathbf{\Pi}$, provides optimal high-safety flexible consensus. To build intuition, consider the simple example of a client who chooses a confirmation quorum $q_k = n$. *Safety* of $\mathbf{\Pi}^*(\mathbf{\Pi})$ depends purely on the OFlex gadget, and does not rely on safety or liveness of the base protocol. When an honest replica post-votes a log (Fig. 4 ❷), it signals to clients that it has perma-locked that log (Fig. 4 ❶); hence, it has never post-voted and will never post-vote an inconsistent log (Fig. 4 ❸). Thus, when the client confirms a log which all $q_k = n$ replicas post-vote, then no client k' with $q_{k'} = n$ will ever confirm an inconsistent log, unless all replicas are adversary, thus achieving $t^{\mathrm{S}}_k = t^{\mathrm{S}}_{k'} = n - 1$. *Liveness* of $\mathbf{\Pi}^*(\mathbf{\Pi})$, on the other hand, depends on both liveness and safety of the base protocol $\mathbf{\Pi}$, and liveness of the OFlex gadget. However, we require $t^{\mathrm{S}}_k \geq t^{\mathrm{L}}_k$ and we know that it is impossible to achieve both resiliences $\geq n/3$. Therefore, clients can only expect $t^{\mathrm{L}}_k < n/3$, and the base protocol is guaranteed to be live and safe in this regime. Then, all honest replicas obtain live logs from $\mathbf{\Pi}$, and also eventually post-vote them because they never see inconsistent logs confirmed by $\mathbf{\Pi}$. Therefore, if all replicas are honest, the client in our example eventually sees $q_k = n$ post-votes for logs that were obtained by honest replicas from $\mathbf{\Pi}$, and as a result this client also confirms the live output log of $\mathbf{\Pi}$, thus achieving $t^{\mathrm{L}}_k = 0$. Simultaneously, another client that chooses a confirmation quorum $q_{k'} = 2n/3 + 1$ obtains resilience guarantees $t^{\mathrm{S}}_{k'} = n/3$, $t^{\mathrm{L}}_{k'} = n/3 - 1$ (the classical parameters). Thus, our OFlex gadget simultaneously supports clients with resiliences $(0, n-1)$ and $(n/3 - 1, n/3)$ (see Fig. 1(c)), while earlier FBFT and SFT-DiemBFT could not support both simultaneously (Fig. 1(b)).

Theorem 1. *If* $\mathbf{\Pi} = (\Pi, \mathcal{C})$ *provides* $(n/3 - 1, n/3)$*-consensus, then the construction* $\mathbf{\Pi}^*(\mathbf{\Pi}) = (\Pi^O \circ \mathcal{C} \circ \Pi, \{\mathcal{C}^O_k\})$*, where the OFlex gadget is applied to* $\mathbf{\Pi}$*, provides optimal high-safety flexible consensus.*

Proof. We show that all clients with $f \leq t^{\mathrm{S}}_k$ and confirmation quorums $q_k > (n + t^{\mathrm{S}}_k)/2$ have safety, and all clients with $f \leq t^{\mathrm{L}}_k \leq t^{\mathrm{S}}_k$ and $q_k \leq n - t^{\mathrm{L}}_k$ have liveness. Therefore, by changing the quorum size q_k, clients can achieve any

[6] As is common practice, we prove which security properties hold when the resiliences are satisfied *throughout the execution*. In Sec. 7, we discuss how security may be recovered after resiliences are violated.

pair of resiliences that satisfies $(n + t_k^{\mathrm{S}})/2 < n - t_k^{\mathrm{L}}$, *i.e.*, $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$, and $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$.

Safety: Suppose, for contradiction, that for two clients k, l, $f \leq \min\{t_k^{\mathrm{S}}, t_{k'}^{\mathrm{S}}\}$, and they confirm inconsistent logs LOG_k^{τ} and $\mathsf{LOG}_{k'}^{\tau'}$. Consider any honest replica that post-voted LOG_k^{τ} or an extension thereof (cf. Fig. 4 ❷). (The quorum intersection argument below shows that such a replica exists.) Then this replica can never post-vote a log LOG' that is inconsistent with LOG_k^{τ} (cf. Fig. 4 ❸). It does not do so after it post-voted LOG_k^{τ} because it has already perma-locked LOG_k^{τ} or an extension thereof (cf. Fig. 4 ❶). It could not have done so before it post-voted LOG_k^{τ} because then it must have perma-locked LOG' and hence would not have post-voted LOG_k^{τ}.

For confirmation by client k, at least q_k replicas post-voted LOG_k^{τ} or an extension thereof. Similarly, at least $q_{k'}$ replicas post-voted $\mathsf{LOG}_{k'}^{\tau'}$ or an extension thereof. Therefore, at least $q_k + q_{k'} - n$ replicas post-voted two inconsistent logs. Due to the preceding argument, these must all be adversary replicas, so $f \geq q_k + q_{k'} - n$. Due to the choice of quorums $q_k > (n + t_k^{\mathrm{S}})/2$ and $q_{k'} > (n + t_{k'}^{\mathrm{S}})/2$, safety violation between clients k, l requires $f \geq (t_k^{\mathrm{S}} + t_{k'}^{\mathrm{S}})/2 + 1$ adversary replicas. This is a contradiction to the assumption that $f \leq \min\{t_k^{\mathrm{S}}, t_{k'}^{\mathrm{S}}\}$.

Liveness: Suppose that $f \leq t_k^{\mathrm{L}}$. Note that since $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$, the feasibility condition $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$ also implies that $t_k^{\mathrm{L}} < n/3$. Therefore, $f < n/3$, which means that $\mathbf{\Pi}$ is safe and live. Due to the safety of $\mathbf{\Pi}$, the log obtained by a replica is always consistent with the logs it obtained in the past, and hence consistent with the replica's perma-lock. Therefore, whenever a replica updates its log from $\mathbf{\Pi}$, it will either post-vote the new log, or has already post-voted that log or an extension thereof. Thus, an honest replica post-votes every log that it obtains from $\mathbf{\Pi}$. Furthermore, since $\mathbf{\Pi}$ is live, transactions eventually appear in every honest replica's log of $\mathbf{\Pi}$. Since $f \leq t_k^{\mathrm{L}}$, client k eventually sees enough post-votes to confirm every log output by $\mathbf{\Pi}$, since post-votes from all honest replicas are enough to satisfy the quorum $q_k \leq n - t_k^{\mathrm{L}}$. □

This construction comes with a communication overhead of one additional round of voting (post-votes) per confirmed batch of transactions. However, in streamlined PBFT-style protocols such as Casper [9], HotStuff [10], or Streamlet [11], we can re-use existing successive rounds of votes in the base protocol to also serve as post-votes, and this eliminates the communication overhead. We describe this modification for Streamlet in Sec. 4, and for Ethereum's implementation of Casper in Sec. 5.

4. OFlex PBFT-Style Protocols

In this section, we use Streamlet [11] as an example to show how the extra round of perma-locking and post-voting for OFlex can be realized in PBFT-style protocols. Specifically, we show how *existing* votes can be reused to double as post-votes (so no extra votes are added), and how perma-locking is implemented as an additional constraint in the protocol's voting rule. The construction carries over rather straightforwardly to other PBFT-style protocols like Hotstuff [10], Casper [9], Tendermint [8].

We first recap Streamlet [11] (Sec. 4.1), then describe the modifications required for OFlex (Sec. 4.2). We call the resulting protocol OFlex-Streamlet. We prove optimal flexibility of OFlex-Streamlet in Sec. 4.3. To contrast OFlex and FBFT in Sec. 4.4, we compare OFlex-Streamlet with an adaptation of FBFT [14] to Streamlet ('FBFT-Streamlet').

4.1. Recap of PBFT-Style Consensus: Streamlet

Streamlet [11] (Alg. 3) proceeds in *epochs* of duration 2Δ, where Δ is the bound on message delays after GST. There are two message types: *blocks* and *votes*. Every message is signed by the replica that creates it. A block consists of a hash pointer to its *parent* block (transitively forming a *chain* back to a commonly known *genesis block*), an epoch number, and transaction payload. A vote consists of a hash pointer to a block. Based on the relation of a block and its parent, 'ancestor', 'descendant', and 'child' are defined in the canonical way. Two blocks are *adjacent* if one is the other's parent. A block is *notarized* in the view of a replica/client when it has seen votes for that block from a quorum of at least $q = 2n/3 + 1$ replicas. A chain is notarized if every block in the chain is notarized. Throughout, replicas echo every message they receive.

The *replica logic* Π consists of two steps: *1) Propose:* Each epoch has a *leader* replica elected using public randomness. At the start of the epoch, the leader broadcasts a new block containing unconfirmed transactions and the current epoch; the parent of the new block is the tip of any one of the longest notarized chains in the *leader's* view. *2) Vote:* During the epoch, each replica broadcasts a vote for the first block received from the epoch's leader, *if* the block's parent is the tip of any one of the longest notarized chains in the *replica's* view.

Confirmation rule $\mathcal{C}$: If a client sees, in a *notarized* chain, three *adjacent* blocks A, B, C from *consecutive* epochs, then it *confirms* B, *i.e.*, sets its log to the sequence of transactions as ordered in the chain from the genesis block to B.

4.2. OFlex-Streamlet Protocol

We modify Streamlet (Alg. 3) to OFlex-Streamlet (Alg. 2). Changes are highlighted green in Alg. 2.

To implement perma-locking, we extend the replica logic with a $\mathsf{permalock}$, initialized to the genesis block. Whenever a replica sees a block B extending its $\mathsf{permalock}$ such that a client would have confirmed B using Streamlet's original confirmation rule $\mathcal{C}$, then the replica updates its $\mathsf{permalock}$ to B (Fig. 5 ❶, Alg. 2 l. 19). A replica's voting is constrained to blocks that are consistent with the $\mathsf{permalock}$ (Fig. 5 ❷ ❸).

In OFlex-Streamlet's confirmation rule $\mathcal{C}'(q_k)$, clients require q_k votes for a block D which is a descendant of the three blocks A, B, C required by Streamlet's rule $\mathcal{C}$ (l. 26). These q_k votes serve as post-votes (Fig. 5 ❷), and allow clients to reason that honest replicas voting for

Algorithm 2 OFlex-Streamlet protocol Π' (changes relative to Alg. 3 are highlighted green)

```
 1: ▷ (permalock $\prec B$) $\triangleq$ chain of permalock is a prefix of chain of $B$
 2: ▷ Replica-side logic $\Pi'$
 3: on INIT()
 4:     $\mathcal{B}, \mathcal{V} \leftarrow \{B_0\}, \{\}$   ▷ Background task: receive blocks and votes into $\mathcal{B}$ and $\mathcal{V}$, respectively, subject to canonical validation (Alg. 3)
 5:     permalock $\leftarrow B_0$   ▷ Initialize perma-lock to genesis block $B_0$
 6: for each epoch $e = 1, 2, 3, \ldots$
 7:     ▷ Propose (done by epoch leader at the start of the epoch)
 8:     $B' \leftarrow$ tip of any one longest notarized chain in $(\mathcal{B}, \mathcal{V})$
 9:     $h \leftarrow \mathrm{Hash}(B')$
10:     txs $\leftarrow$ transactions not present in chain of $B'$
11:     Sign and broadcast block $(h, e, \mathsf{txs})$
12:     ▷ Vote (done by all replicas once during the epoch)
13:     $B \leftarrow$ first block from epoch $e$ in $\mathcal{B}$ signed by epoch leader
14:     $B' \leftarrow$ parent block of $B$ in $\mathcal{B}$
15:     if $B'$ is tip of any longest notarized chain in $(\mathcal{B}, \mathcal{V})$
16:         if permalock $\prec B'$
17:             $h \leftarrow \mathrm{Hash}(B)$
18:             Sign and broadcast vote $h$
19:     ▷ Perma-lock (done by all replicas throughout the epoch)
20:     if $(\mathcal{B}, \mathcal{V})$ contains a notarized chain with three adjacent blocks $A, B, C$ from consecutive epochs, and permalock $\prec B$
21:         permalock $\leftarrow B$
22: ▷ Client-side confirmation rule $\mathcal{C}'$ for resilience pair $(t_k^{\mathrm{L}}, t_k^{\mathrm{S}})$
23: on INIT($q_k$) where $q_k$ shall satisfy $(n + t_k^{\mathrm{S}})/2 < q_k \leq n - t_k^{\mathrm{L}}$
24:     Set local quorum size $q_k$
25:     $\mathcal{B}, \mathcal{V} \leftarrow \{B_0\}, \{\}$   ▷ Background task: receive blocks and votes into $\mathcal{B}$ and $\mathcal{V}$, respectively, subject to canonical validation (Alg. 3)
26: ▷ Confirmation
27: if $(\mathcal{B}, \mathcal{V})$ contains a notarized chain with three adjacent blocks $A, B, C$ from consecutive epochs, and $(\mathcal{B}, \mathcal{V})$ contains a block $D$ such that $C \prec D$ and $D$ has received $q_k$ votes
28:     Choose $A, B, C$ as such blocks with maximum height
29:     LOG $\leftarrow$ sequence of transactions as ordered in chain of $B$
```

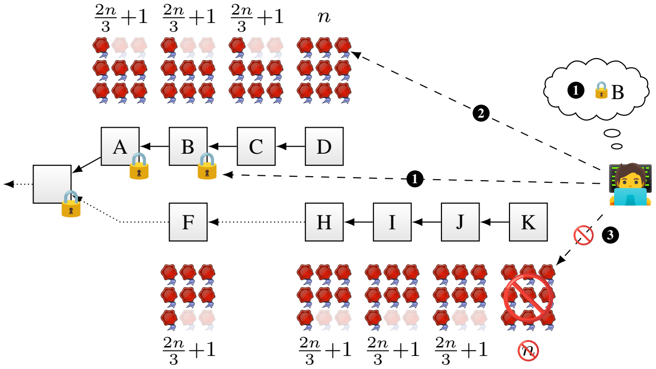


Figure 5. Illustration of OFlex-Streamlet (cf. Fig. 4). Each ● represents a vote, and the number of votes for each block is indicated. * *Protocol rules:* ❶ If a replica sees a block B that would have been confirmed according to Streamlet's original confirmation rule $\mathcal{C}$ (*i.e.*, adjacent blocks A, B, C from consecutive epochs in notarized chain), then the replica perma-locks B if B extends its current perma-lock. ❷ From a replica's vote for D, clients can infer that the replica, if honest, must be perma-locked on B. ❸ This, in turn, means the replica, if honest, will never vote for blocks inconsistent with B. * *Safety intuition:* A client k with high quorum $q_k = n$ confirms B when it sees q_k votes on D. Due to steps ❶ to ❸, no block K conflicting with B can receive n votes and get confirmed by another client k' with $q_{k'} = n$, *unless all replicas are adversary*, implying $t_k^{\mathrm{S}} = t_{k'}^{\mathrm{S}} = n - 1$.

D are perma-locked on B (Fig. 5 ❶) and will not vote for anything conflicting with B (Fig. 5 ❸). This reprises the safety argument of OFlex in Sec. 3, and is key to OFlex-Streamlet's safety argument and improved resilience compared to FBFT and SFT-DiemBFT.

Note that we did not have to modify Streamlet's proposing rule, which is in line with the generic OFlex construction.

4.3. Security Analysis

To build intuition, in Fig. 5, we show an example of how OFlex-Streamlet provides safety with $t_k^{\mathrm{S}} = n-1$ for clients that confirm according to the rule $\mathcal{C}'(q_k = n)$. Such a client confirms the block B after seeing the blocks A, B, C, D with the required votes. When the client sees a replica vote on the block D (Fig. 5 ❷), it knows that this replica, if honest, must have seen the blocks A, B, C notarized (honest replicas only vote for blocks whose parent chain they have seen notarized), and therefore must have perma-locked the block B (Fig. 5 ❶), and will thus never vote for a block that is inconsistent with B (Fig. 5 ❸). Thus, if a client sees all n replicas vote for block D, then it knows that unless all these replicas are adversary, a block inconsistent with B cannot obtain votes from all replicas. Thus, clients that confirm blocks with $q_k = n$ remain safe even when $f = n-1$ replicas are adversary.

To intuit liveness, recall that the requirement $t_k^{\mathrm{S}} \geq t_k^{\mathrm{L}}$ implies $t_k^{\mathrm{L}} < n/3$. Thus, clients can expect liveness only when $f < n/3$. Tracing Streamlet's safety analysis in this regime shows that every replica's permalock is a prefix of every longest notarized chain in its view at all times. As a result, when $f < n/3$, the permalock constraint in OFlex-Streamlet is inactive (cf. l. 15), and OFlex-Streamlet 'behaves like' Streamlet. Therefore, for a client with $q_k \leq n - t_k^{\mathrm{L}}$, if $f \leq t_k^{\mathrm{L}} < n/3$, after sufficiently many successive honest leaders, the client eventually sees blocks with q_k votes, in particular, blocks that satisfy the confirmation rule.

Theorem 2. *OFlex-Streamlet provides optimal flexible consensus.*

Proof. We prove safety and liveness in Lems. 1 and 2, respectively, for clients who choose their confirmation quorums appropriately. For any client k choosing resilience pair $(t_k^{\mathrm{L}}, t_k^{\mathrm{S}})$ such that $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$ and $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$, there exists a confirmation quorum q_k that satisfies the requirements of Lems. 1 and 2, *i.e.*, $(n + t_k^{\mathrm{S}})/2 < q_k \leq n - t_k^{\mathrm{L}}$. Thus, by changing the confirmation quorum q_k, clients can achieve any $(t_k^{\mathrm{L}}, t_k^{\mathrm{S}})$ that satisfies $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$ and $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$. □

Lemma 1 (Safety)**.** *For all clients k, k' such that $q_k > (n + t_k^{\mathrm{S}})/2$ and $q_{k'} > (n + t_{k'}^{\mathrm{S}})/2$, if $f \leq \min\{t_k^{\mathrm{S}}, t_{k'}^{\mathrm{S}}\}$, then for all times τ, τ', LOG_k^{τ} and $\mathsf{LOG}_{k'}^{\tau'}$ are consistent.*

Proof. Suppose, for contradiction, that there exist two clients k, k' and two time instants τ, τ' such that LOG_k^{τ}

and $\mathsf{LOG}_{k'}^{\tau'}$ are not consistent. Then, there must exist two notarized chains that satisfy the respective confirmation rules of the clients. So in the first chain, there are four blocks A, B, C, D from epochs $e-x-2, e-x-1, e-x, e$ for some $x \geq 1$, where A, B, C are adjacent, and D is a descendant of C and has received q_k votes in client k's view. In the second chain, there are four blocks H, I, J, K from epochs $e'-y-2, e'-y-1, e'-y, e'$ for some $y \geq 1$, where H, I, J are adjacent, and K is a descendant of J and has received $q_{k'}$ votes in client k''s view. Furthermore, B and I are inconsistent. See Fig. 5 for illustration.

Without loss of generality, we may assume that $e' \geq e$. Honest replicas only vote for a block if they have seen its parent chain notarized. Therefore, an honest replica who voted for the block D in epoch e (cf. Fig. 5 ❷) must have seen blocks A, B, C notarized. Furthermore, an honest replica only votes for the block D if it is also a descendant of the replica's perma-lock. Therefore, either the block B is a descendant of the replica's perma-lock, or the replica must have already perma-locked B. In either case, the replica will have perma-locked the block B by the end of epoch e (cf. Fig. 5 ❶). An honest replica that votes for the block D in epoch e does not vote for any other block in epoch e. Furthermore, it does not vote for any block that is inconsistent with block B in epochs $> e$, due to its perma-lock on B (cf. Fig. 5 ❸). Since K is a descendant of I, but I and B are inconsistent, we conclude that no honest replica votes for both blocks D and K.

Since blocks D and K have received at least q_k and $q_{k'}$ votes respectively, at least $q_k + q_{k'} - n$ replicas must have voted for both blocks. Due to the preceding argument, these must all be adversary replicas, so $f \geq q_k + q_{k'} - n$. Due to the choice of quorums $q_k > (n + t_k^{\mathrm{S}})/2$ and $q_{k'} > (n + t_{k'}^{\mathrm{S}})/2$, safety violation between clients k, k' requires $f \geq (t_k^{\mathrm{S}} + t_{k'}^{\mathrm{S}})/2 + 1$ adversary replicas. This is a contradiction to the assumption that $f \leq \min\{t_k^{\mathrm{S}}, t_{k'}^{\mathrm{S}}\}$. □

The key new step towards proving liveness of OFlex-Streamlet is proving that when $f < n/3$ (recall that we only require liveness for clients with $f \leq t_k^{\mathrm{L}} < n/3$), the additional constraint on voting of OFlex-Streamlet is never active (Lem. 4). This follows from a safety property of Streamlet (which also holds in OFlex-Streamlet) that once a block is confirmed according to Streamlet (or perma-locked according to OFlex-Streamlet), no block inconsistent with it ever gets notarized (Lem. 3). Thus, under the regime of interest for liveness ($f < n/3$), the voting rule of OFlex-Streamlet behaves exactly like that of Streamlet, and the rest of the proof for liveness follows using techniques from [11], except that OFlex-Streamlet requires one additional epoch to confirm blocks compared to Streamlet (Lem. 5). Liveness (Lem. 2) follows immediately. Proofs of Lems. 3 and 5 are given in App. A as they mostly follow steps from [11].

Lemma 2 (Liveness)**.** *For every* tx *input to all honest replicas, eventually, for all clients k such that $q_k \leq n - t_k^{\mathrm{L}}$ (quorum choice), $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$, $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$ (optimal flexible resiliences), and $f \leq t_k^{\mathrm{L}}$,* $\mathsf{tx} \in \mathsf{LOG}_k$.

The following lemmas build up to the proof of Lem. 2.

Lemma 3 (cf. [11, Lem. 2])**.** *If $f < n/3$, then if some honest replica sees three adjacent blocks A, B, C with consecutive epoch numbers on a notarized blockchain, then there cannot be a block $F \neq B$ at the same height as B that is also notarized in an honest replica's view.*

Proof is given in App. A.

Lemma 4. *If $f < n/3$, then if a block B's parent's chain is one of the longest notarized chains in a replica's view, then B is also a descendant of that replica's perma-lock.*

Proof. Suppose that in epoch e, a block B is proposed whose parent chain is one of the longest notarized chains seen by an honest replica at epoch e. Let B' be this honest replica's perma-lock as of epoch e. If B' is the genesis block, then B must be its descendant. Otherwise, at the end of some epoch $e' < e$, this replica saw three adjacent blocks A', B', C' with consecutive epoch numbers on a notarized blockchain. For a contradiction, suppose that B is not a descendant of B'. Since the replica saw block C' notarized at epoch e', it must be that B is at a greater height than B'. Then there must be a block on the parent chain of B at the same height as B' that is also notarized. Due to Lem. 3, this is a contradiction. □

Lemma 5 (cf. [11, Thm. 4])**.** *After* GST*, suppose that there are 6 consecutive epochs $e, e+1, ..., e+5$ with honest leaders. Then by the beginning of epoch $e+6$, every client k such that $f \leq t_k^{\mathrm{L}} < n/3$ and $q_k \leq n - t_k^{\mathrm{L}}$ will have confirmed a new block which it had not confirmed at the beginning of epoch e. Moreover, this new block was proposed by an honest leader.*

Proof is given in App. A.

Proof of Lem. 2. The conditions $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} < n$ and $t_k^{\mathrm{L}} \leq t_k^{\mathrm{S}}$ imply $t_k^{\mathrm{L}} < 1/3$. Then, Lem. 5 directly implies liveness. The conditions required in Lem. 5 occur eventually, *i.e.*, after GST and after $\left(\frac{n}{n-f}\right)^6$ epochs in expectation. Due to the honest replicas' propose rule, the new block by an honest leader (referred to in Lem. 5) includes any tx input to all honest replicas, that were not already in its parent chain. This block and its parent chain are confirmed by all clients k for which $q_k \leq n - t_k^{\mathrm{L}}$. □

4.4. Comparison with FBFT

We briefly describe a straightforward and truthful adaptation[7] of FBFT [14] to Streamlet (which we call FBFT-Streamlet) to facilitate a clear comparison of FBFT's and OFlex's paradigms. FBFT-Streamlet is identical to Streamlet, except for its confirmation rule: "When client k sees three adjacent blocks A, B, C from consecutive epochs in a notarized chain, $\underline{\text{each with } q_k \text{ votes}}$, then it confirms

[7] While the FBFT protocol of [14] is inspired by HotStuff, the similarity between Streamlet and HotStuff [10], [15], [32] enables this adaptation.

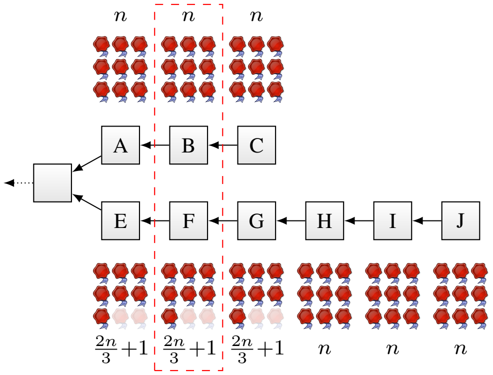


Figure 6. An execution showing that in FBFT-Streamlet with $q = \frac{2n}{3} + 1$ clients with $q_k = n$ are not safe if $f = \frac{2n}{3} + 1$. Blocks A, B, C receive n votes each, which leads a client k with ‘heavy quorum’ $q_k = n$ to confirm the block B. Since the adversary controls $\frac{2n}{3} + 1$ replicas, it can notarize (with ‘light quorum’ $q = \frac{2n}{3} + 1$) inconsistent blocks E, F, G. Subsequently, leaders propose blocks H, I, J and all honest replicas vote for these blocks, which is in accordance with the protocol rules. Including the adversary’s votes, these blocks meet a ‘heavy quorum’ $q_{k'} = n$ needed for another client k' to confirm, leading to a safety violation.

B.” Note that this is Streamlet’s original confirmation rule, except with q_k instead of q votes for each block.

First, for liveness, observe that with $q_k \geq q$, both FBFT-Streamlet and OFlex-Streamlet require $f \leq t_k^{\mathrm{L}} \leq n - q_k$. For safety, however, FBFT-Streamlet requires $f \leq t_k^{\mathrm{S}} < q_k + q - n$, while OFlex-Streamlet only requires $f \leq t_k^{\mathrm{S}} < 2q_k - n$. This enables OFlex-Streamlet’s optimal flexibility.

To understand where FBFT-Streamlet’s sub-optimal bound on t_k^{S} comes from, consider how FBFT-Streamlet’s safety breaks in the example of $q = \frac{2n}{3} + 1$, clients with $q_k = n$, and f just exceeding the clients’ safety resilience in FBFT-Streamlet, *i.e.*, $f = q_k + q - n = \frac{2n}{3} + 1$ (illustrated in Fig. 6): Suppose blocks A, B, C received votes from all replicas, leading a client k with $q_k = n$ to confirm block B. Other clients may not hear of these blocks and votes for a while (before GST). Since the adversary controls $\frac{2n}{3} + 1$ replicas, it can, by its own efforts, notarize inconsistent blocks E, F, G. In the next slot, the leader proposes block H with parent G. This indeed follows the protocol rules because G is ‘the tip of one of the longest notarized chains’. When H is proposed, all honest replicas will vote for H because H is the only block proposed by the leader and its parent G is the tip of one of the longest notarized chains (cf. Sec. 4.1). The adversary replicas also vote for H which results in n votes for H in total. Similarly, leaders propose blocks I, J and they too gather n votes each. Ultimately, the block I satisfies the confirmation rule for $q_{k'} = n$ in another client k' (who has not heard of B). This is a safety violation between k and k' because B and I are inconsistent.

Key issue at hand: the adversary was able to by-pass the heavy quorum q_k on A, B, C with a light quorum q on E, F, G, and honest replicas subsequently helped the adversary achieve a conflicting heavy quorum $q_{k'}$ on H, I, J.

To rule this out, FBFT-Streamlet imposes $f \leq t_k^{\mathrm{S}} < q_k + q - n$, which (by a standard quorum intersection argument) ensures that no block F can reach quorum q on the same height as B when A, B, C reach quorum q_k (red dashed in Fig. 6). Thus, no by-passing, and no conflicting heavy quorum. But the constraint turns out to be sub-optimal.

Instead, OFlex-Streamlet addresses the key issue with perma-locking and post-voting, leaving the adversary the power to reach light quorum q on a block F conflicting with B, but preventing honest replicas from contributing to the conflicting heavy quorum $q_{k'}$ that would be necessary for a conflicting confirmation. See Fig. 5 for OFlex-Streamlet in a situation similar to Fig. 6, where, however, honest replicas do not vote for blocks H, I, J. First, when honest replicas see blocks A, B, C notarized ($q = \frac{2n}{3} + 1$ votes are enough for this), they perma-lock the block B (Fig. 5 ❶). But, we also require the client to know that honest replicas have perma-locked B, so that the client can safely confirm B, knowing honest replicas will not contribute votes in favor of a conflicting block (Fig. 5 ❸). This is why in OFlex-Streamlet, the confirmation rule requires the client to see an additional block D with n votes (Fig. 5 ❷), from which the client infers that all honest replicas must have perma-locked block B. Thus, it would require n adversary replicas to confirm an inconsistent block such as I. In OFlex-Streamlet, only $\frac{2n}{3} + 1$ votes are required for the blocks A, B, C and q_k votes are required only for block D (Fig. 5), which suffice for the client to infer replicas’ perma-locks.

5. OFlex Confirmation Rules for Ethereum

Ethereum is a proof-of-stake blockchain built [31] on Casper [9], a PBFT-style consensus protocol very similar [33] to Streamlet.[8] Ethereum allows participants to lock up 32 ETH tokens to participate as replicas (called ‘validators’ in Ethereum) in Casper.

In the generic OFlex construction, we added extra perma-locking and post-voting to the replica logic and designed a new confirmation rule. In OFlex-Streamlet, we reused Streamlet’s votes as post-votes, and only required modifying the replica voting rule to introduce the perma-lock constraint, and designing a new confirmation rule. Due to Casper’s similarity with Streamlet, we could do the same for Casper. However, we observe that Ethereum’s *implementation* of Casper already has an unrelated performance optimization that de-facto implements the required perma-

[8] Casper [9] is only proven to satisfy a weaker liveness notion called ‘plausible liveness’ [9, Thm. 2]. Indeed, [34], [35], [36] show liveness attacks on the Casper component of Ethereum. The OFlex confirmation rules for Ethereum make the liveness and safety notions guaranteed by Casper optimally flexible. Once Ethereum upgrades to ensure standard liveness for Casper, the OFlex rule will guarantee standard liveness accordingly.

lock and voting constraint.[9] As a result, to provide optimal flexible consensus on top of Ethereum as-is today, we do not require *any* modifications to the validators, but only new client-local confirmation rules.

We describe how Ethereum's implementation provides the perma-lock and voting constraint and state our flexible OFlex confirmation rules in Sec. 5.1. We describe our implementation of the confirmation rule in Sec. 5.2 and show experimental results in Sec. 5.3.

5.1. Confirmation Rules

Due to Casper's similarity [33] to Streamlet, OFlex confirmation rules for Casper closely follow OFlex-Streamlet.

We first briefly describe the features already implemented by Ethereum validators which enable designing client-side OFlex confirmation rules without having to change the validator logic. A detailed description of Casper and of the Ethereum protocol can be found in [9], [31], [37].

1) Validators maintain a 'finalized checkpoint', which is a block that satisfies the 'default confirmation rule' called 'finality' of Casper (cf. $\mathcal{C}$ in Fig. 2). The finalized checkpoint is safe up to $n/3$ adversary validators, *i.e.*, no other block inconsistent with a finalized checkpoint will ever be finalized if $\leq n/3$ validators are adversary.
2) When the finalized checkpoint is updated, the new finalized checkpoint must extend the old one.
3) Validators only vote for blocks that extend the finalized checkpoint in their view.
4) Votes for a block are included on-chain in descendant blocks. Validators consider only votes included on-chain to update their finalized checkpoint. Blocks with votes deemed invalid are deemed invalid.

We see below how the above features realize the required perma-lock and voting constraint, just as in OFlex-Streamlet (Sec. 4.2). Items 1 and 2 show that a validator's finalized checkpoint behaves as its perma-lock, *i.e.*, the finalized checkpoint satisfies Casper's original confirmation rule, and once a block is finalized, further finalized checkpoints must extend that block. The voting rule in Item 3 shows that validators never vote for a block that is inconsistent with their finalized checkpoint, *i.e.*, perma-lock. Just as in OFlex-Streamlet, the safety of Casper's finality when $f \leq n/3$ (Item 1) guarantees that the voting constraint in Item 3 is never active when $f \leq n/3$ (recall that an analogous property was required for liveness of OFlex-Streamlet).[10]

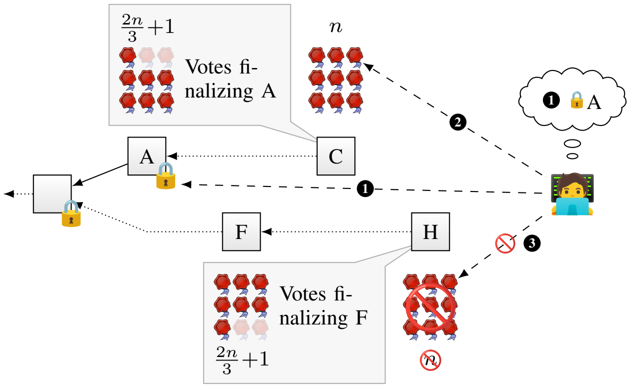


Figure 7. Illustration of OFlex confirmation rules for Ethereum. * *Protocol rules:* ❶ If a validator sees a block C that contains enough votes to finalize block A, then the validator sets A as its finalized checkpoint. ❷ From a validator's vote on block C, clients can infer that the validator, if honest, must have set A as its finalized checkpoint. ❸ An honest validator that voted for C will never vote for F, H that are inconsistent with its finalized checkpoint A. * *Safety intuition:* Seeing votes from n validators on block C, a client with $q_k = n$ confirms block A. Due to steps ❶ to ❸, when $\frac{2n}{3}+1 \leq f < n$, blocks F, H may be finalized (by adversarial votes alone), but will never obtain votes from all n validators. Therefore, no client k' with $q_{k'} = n$ confirms a block inconsistent with A, unless *all* validators are adversary, implying $t_k^{\mathrm{S}} = t_{k'}^{\mathrm{S}} = n-1$.

Recall from OFlex-Streamlet, that we also needed clients to be able to infer how many validators have perma-locked a certain block and when it is thus safe to confirm that block. In Ethereum's Casper implementation, having votes on chain (Item 4) provides the evidence for the client to know when a validator must have perma-locked, because if a validator votes for a block, it must have seen (and deemed valid) all votes contained in the chain leading up to its vote target.

Since perma-locking and post-voting are de-facto already implemented in Ethereum's Casper, the following suffices for optimal flexible consensus in Ethereum:

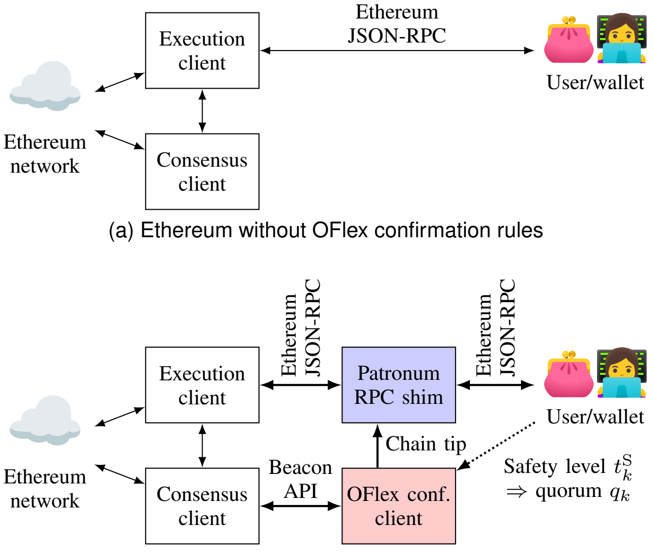
OFlex confirmation rules $\mathcal{C}'(q_k)$ for Ethereum: A client k confirms block A iff: (1) it sees a block C descending from A that contains votes (in C or its prefix) to finalize block A, and (2) it sees votes from q_k validators for C (included on-chain or received otherwise).

Fig. 7 illustrates how this rule provides safety in an example with clients with $t_k^{\mathrm{S}} = n-1$. The example proceeds analogously to OFlex-Streamlet (Fig. 5). If a client sees n validators vote for a block C which contains enough votes to finalize a previous block A, then the client confirms A. When the client sees a validator vote for C (Fig. 7 ❷), it knows that this validator, if honest, must have seen the votes included in the chain of C resulting in the finalization of A, and therefore must have set A as its finalized checkpoint (Fig. 7 ❶). This validator will thus never vote for a block inconsistent with A (Fig. 7 ❸). Thus, if a client sees all n replicas vote for C, then it knows that, unless all replicas are adversary, a block inconsistent with A cannot obtain votes from all replicas. Thus, clients that confirm blocks with $q_k = n$ remain safe even when $f = n-1$.

Note that clients using OFlex rules can operate along-

[9] In Ethereum's implementation, validators ignore blocks that conflict with a block that they previously considered irreversible ('finalized'), thus effectively perma-locking that block. This improves computational efficiency. Furthermore, this perma-lock is not present in the *textbook* version of Casper [9]. We therefore believe that this perma-locking mechanism was added as an unrelated performance optimization and was previously not considered an integral part of Ethereum's consensus protocol. It seems plausible that other PBFT-style consensus deployments feature a similar perma-lock for performance optimization, and therefore can also be upgraded to OFlex confirmation rules with only client-local changes.

[10] This justifies why Item 3 is a correct performance optimization: it does not cause the optimized system to behave differently from the unoptimized system under normal conditions when $f \leq n/3$.



(a) Ethereum without OFlex confirmation rules

(b) Ethereum with OFlex confirmation rules

Figure 8. OFlex confirmation rules can be adopted by Ethereum users unilaterally without any changes to internals or interfaces of execution clients, consensus clients, or wallets. The OFlex confirmation client (this work) determines a chain tip satisfying the user-provided safety level. The Patronum RPC shim [38], [39] can provide responses to queries from the wallet based on that chain tip.

side clients that continue to use Casper finality as their confirmation rule. Specifically, all clients simultaneously enjoy the flexible consensus guarantees (Def. 1) with their respective resiliences ($t_k^{\mathrm{L}} = \frac{n}{3} - 1, t_k^{\mathrm{S}} = \frac{n}{3}$ for clients using finality). Although Casper finality and the OFlex rule with $q_k = \frac{2n}{3} + 1$ enjoy the same resiliences, the logs confirmed by them are not identical. The OFlex rule requires an additional block (C in Fig. 7).

5.2. Implementation

Before describing our implementation of the OFlex rule, we first briefly explain the software stack that a user runs in order to interact with the Ethereum blockchain, and how our implementation fits in the current system. There are two essential pieces of software: an Ethereum *consensus* client, and an *execution* client, as shown in Fig. 8(a). The consensus client connects to the Ethereum peer-to-peer network to obtain latest blocks, and tries to confirm blocks according to the consensus protocol (*i.e.*, Casper finality). It feeds confirmed blocks to the execution client, which executes the transactions inside to obtain the system state (*e.g.*, account balances) with regard to the tip of the confirmed log. User applications such as wallets can then query the execution client for latest system state through the JSON-RPC API.

We notice that existing Ethereum consensus clients already expose all the data required to run the OFlex confirmation rule, namely the finalized checkpoint according to votes in the chain leading up to any block C, and the fraction of validators voting for C. As a result, the OFlex rule can be implemented as a standalone program that runs alongside an existing, unmodified consensus client, subscribes to the client for chain data, and outputs the tip of the chain confirmed by the OFlex rule with any desired safety resilience. Fig. 8(b) shows one way to integrate such a program into the existing system. Instead of connecting directly to the execution client (which by default answers queries with regard to the chain tip confirmed by Casper finality), user applications connect to Patronum [38], [39], an RPC proxy. Patronum learns the confirmed tip from the OFlex rule, and rewrites user queries so that the execution client always answers them with regard to *this* tip. This process is transparent to the user as well as the consensus and the execution clients, so that applications can benefit from the new rule without any change to their code.

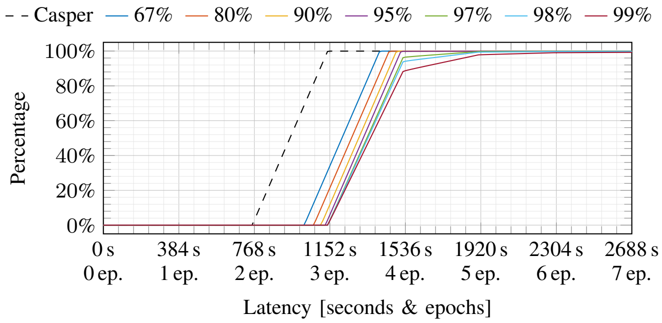


Figure 9. Empirical cumulative distribution function of latency to confirm Ethereum mainnet blocks between slots 5,970,000 and 6,970,000 by the OFlex rule with different quorums, and by Casper finality.

We implemented the OFlex confirmation rule following this design in 1,100 lines of Rust code.[11] For any block C, we use the `/states/finality_checkpoints` endpoint of the Ethereum Beacon API [40] to query the latest block finalized by Casper according to votes in the chain leading to C, and use the `/states/committees` endpoint to query the set of validators selected to vote for C. We then use the `/blocks` endpoint to fetch subsequent blocks and examine the votes included to count the number of validators who actually vote for C. We tested our implementation against Prysm and Lighthouse, the two most popular Ethereum consensus clients as of now [41].

5.3. Experiments

To evaluate the behavior of OFlex rules in the real world, we use our implementation to apply the rule with various quorum sizes on a section of Ethereum mainnet between slots 5,970,000 (March 10th, 2023) and 6,970,000 (July 27th, 2023), covering roughly the most recent 1/7 of the Ethereum PoS mainnet history as of when this paper is written. We choose this section because it is recent enough to reflect current statistics, and it covers two rare but interesting events: the finality outage incident that happened on May 12th, 2023 [43], and the Shanghai hard fork that happened on April 12th, 2023. During both events, participation of

[11] Source code: https://github.com/tse-group/flexible-eth

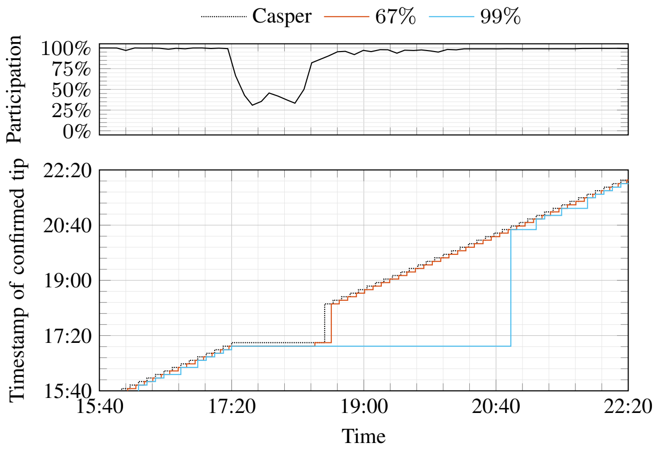


Figure 10. Ethereum mainnet chain confirmed by Casper finality (·······) and by our OFlex confirmation rule with low quorum (67%, ——) and high quorum (99%, ——). Shown are slots 6,423,500 to 6,425,500, covering a finality outage incident [42], [43] on May 12th, 2023 (times in UTC).

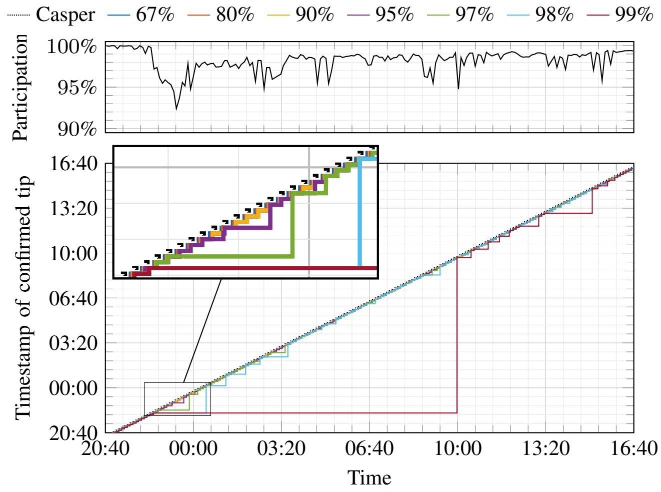


Figure 11. Ethereum mainnet chain confirmed by Casper finality and by our OFlex confirmation rule with different quorums (cf. legend). Shown are slots 6,209,000 to 6,215,000, covering the Shanghai hard fork at slot 6,209,536 (on April 12th, 2023 at 22:27 UTC on the plot).

validators dipped, allowing us to demonstrate the behavior of OFlex rules under non-standard conditions.

Fig. 9 shows how the distribution of confirmation latency changes with confirmation rule and quorum. Compared to Ethereum's Casper finality, OFlex rules incurs an extra latency of approximately one epoch due to the extra round of voting. This extra latency is proportional to the quorum size, because a quorum of q_k requires waiting for q_k validators to vote, which takes roughly (q_k/n)-fraction of an epoch to happen. This effect is shown by the curves shifting towards right as quorum size increases. The key takeaway is that for Ethereum mainnet, adopting OFlex rules results in only modest increase of confirmation latency, even when opting for extremely high quorum (*e.g.*, 99%). For example, the 95th-percentile tail latency to confirm blocks with 99% quorum is 30 minutes, an increase of 60% compared to Casper finality. We argue that this is a cost well worth paying, because confirming with 99% quorum offers resilience against an extremely powerful 98% adversary, while Casper finality is safe only against a 33% adversary. Similarly, once adopting our rule, increasing the quorum size results in only slight further increase of latency. For example, adjusting the quorum size from 80% to 99% increases the tail latency by 25%, while boosting the safety resilience from 60% to 98%. Operationally, we expect users adopting our rule to use very high quorums to maximize the safety benefit.

Ethereum mainnet typically runs in a healthy steady-state with close to 100% of validators correctly participating, allowing our rule to achieve good confirmation latency even with high quorums, as just shown. Now, we zoom into two events when participation dipped, and examine how our rules behave under such abnormal conditions. For each event, we plot over time the growth of the logs confirmed by our rules with different quorums, as well as that by Casper finality. We also plot the participation rate (*i.e.*, fraction of validators online and actively voting) for reference.

Fig. 10 shows an incident that happened on May 12th, 2023, when a bug in some validators' software was triggered and brought them offline [43]. Before the incident, the logs confirmed by OFlex rules closely tracked Casper finality, as shown on the left side of the plot. At around 17:20, the bug was triggered and participation quickly dropped below the threshold required by Casper (67%) and OFlex rules (67% and 99%, respectively), causing all three logs to stop growing (the horizontal stretch of the lines). At around 18:10, validators started to patch their software, and participation surpassed 67% at around 18:30, allowing the Casper log and the 67%-quorum log to recover. The 99%-quorum log stalled for a while longer as there were not enough validators to form a 99% quorum, and recovered at around 20:50 when participation surpassed 99%.

Fig. 11 shows the Shanghai hard fork, when validators had to upgrade their software in order to keep participating post-fork. Unlike in the other incident, since this was a scheduled transition, participation rate after the hard fork never dropped below 90%. As a result, logs confirmed by OFlex rules with quorum sizes no larger than 90% experienced no pause and kept tracking Casper finality. The other logs except for the one confirmed by 99%-quorum quickly recovered as shown in the scope. It took significantly longer for the participation rate to recover above 99% and for the 99%-quorum log to resume growth.

We remark that each user can choose to confirm with different quorums at different times and for different transactions, while maintaining high safety for all transactions that were confirmed with a high quorum. For instance, a user can choose to confirm with a 99% quorum most of the time but switch to confirming with a lower quorum during incidents such as the ones described above.

6. Related Work

Multi-threshold consensus: Multi-threshold Byzantine fault tolerance [1], [2], [3], [4] can be viewed as a degenerate case of flexible consensus in which all clients have the same resilience pair $(\mathrm{t^L}, \mathrm{t^S})$ which may be different from $(\frac{1}{3}, \frac{1}{3})$. This is done for SMR consensus in [1], [4], with a weaker notion of safety in [44], and for reliable broadcast in [1], [2], [3].

Multi-threshold BFT [1] argues that for some applications one may desire a higher safety resilience than liveness resilience ($t^{\mathrm{S}} \geq t^{\mathrm{L}}$). The same problem is stated from another point of view in [3], [5], [14], [45], [46]: 'rational' adversaries may be willing to attack safety but not liveness (cf. alive-but-corrupt faults [14], deceitful faults [5], Byzantine merchants [46]) because an adversary can expect sizeable profits from double spends by attacking safety, but stands to gain little (and instead loses protocol rewards) if it were to attack liveness. Tolerating alive-but-corrupt adversaries in addition to Byzantine adversaries is then equivalent to having a higher safety resilience than liveness resilience. In contrast, some argue that crash faults are more common than Byzantine faults, and therefore, they design protocols to tolerate crash faults in addition to Byzantine faults [47]. Relatedly, [48], [49], [50] consider other classes of adversaries.

Orthogonal to our work, protocols in [1], [51], [52], [53] achieve the respective optimal safety and liveness resiliences under different network conditions (synchrony, asynchrony), but without allowing clients flexibility to choose resilience pairs. In GearBox [54], replicas optimistically sub-sample a small committee to run the protocol, and adaptively tune the committee size and quorum threshold in order to retain worst-case safety and liveness resilience of $1/3$ of the full replica set. There are two key differences to our work. First, the goal of GearBox is not multi-threshold or flexible consensus; thus, safety and liveness of GearBox break *for all clients* once $f \geq n/3$. Second, the quorum threshold is tuned in a system-wide fashion, not client-local/flexible.

Flexible protocols: FBFT [14] allows clients to not only choose resilience pairs, but also choose different network assumptions (synchronous or partially synchronous). Protocols with an optimistic fast path [55], [56], [57] allow clients to trade-off between the liveness resilience and confirmation latency, but not the safety resilience. In this work, we focus on achieving the optimal trade-off between safety and liveness resiliences for a fixed partially synchronous network assumption. Moreover in FBFT, the replica logic is essentially the same as in prior protocols such as Hotstuff [10] (excluding changes geared towards flexibility in the network assumptions). In this work, we instead modify (in OFlex-Streamlet) the replica voting rule with a restriction based on perma-lock, which is crucial to achieving optimal flexibility. Highway [45] also aims to provide flexibleresiliences but has no proven liveness guarantees when the safety resilience exceeds $\frac{1}{3}$ as per [45, Thm. 2]. The idea of flexible quorums, used by [14] and our work, also appears in [58].

Strenghtened fault tolerance: SFT [15] extends [14] from one-shot Byzantine agreement to a full SMR protocol while maintaining 'linear message complexity'. We discuss how our generic OFlex construction can also be made to achieve linear message complexity: In every 'round' (whose duration will be determined later), replicas should take turns being an 'aggregator'. During a round, all replicas send their post-votes to the aggregator. The aggregator then collects all post-votes it receives during the round for any log that is the same as or an extension of the log that the aggregator itself post-voted, and packages these post-votes into a certificate. This solution preserves safety because post-votes cannot be forged by the aggregator. Liveness is preserved because, after GST, eventually, all replicas see the log that the aggregator post-voted. Moreover, in the regime where liveness needs to be provided (*i.e.*, if $f < n/3$), the base protocol is safe, so honest replicas only post-vote logs consistent with the aggregator's post-vote. Thus, if the round is long enough (confirmation latency of the base protocol), then all honest replicas are guaranteed to post-vote some log extending the aggregator's post-vote within that round. With this construction, if the base protocol has linear message complexity, then so does the combined OFlex protocol (the only additional messages are post-votes). Since all post-votes collected by the aggregator extend a common prefix, the aggregator can use a SNARK to reduce the message to constant size, thus achieving linear *communication* complexity.

It is equally easy to pipeline the above process in a linear-communication protocol such as DiemBFT [59] (which [15] uses) such that round leaders of the protocol also serve as aggregators. In SFT-DiemBFT, the main difficulty was how to re-use votes for a block as votes for its ancestors without double-counting conflicting votes. To do this, replicas in SFT-DiemBFT [15] attach a 'marker' to votes through which the replica signals to have not voted for an inconsistent block in the recent past. In OFlex, such markers are not needed since the perma-lock already provides the function of the marker (the post-voting replica has never post-voted and will never post-vote an inconsistent block). By adapting the other techniques used in [15], one readily obtains an 'OFlex-DiemBFT' with linear message complexity, without even the message overhead of the markers.

Snap-and-chat protocols: Snap-and-chat protocols [34] provide a construction of flexible consensus where a 'more live' consensus protocol and a 'more safe' one are concatenated to yield two confirmation rules, a 'more live' one and a 'more safe' one. This construction can be extended to yield a resilience-optimal flexible consensus protocol: a concatenation of $\frac{n}{3}$ consensus protocols, each with a different system-wide quorum, starting from one with a quorum $q = \frac{2n}{3} + 1$ all the way up to one with $q = n$. A more detailed description is given in Fig. 12. Clearly, running $\frac{n}{3}$ protocols simultaneously is impractical for the replicas. Our generic construction (Fig. 2) points out that all these protocols except the first one ($q = \frac{2n}{3} + 1$) can be collapsed into a single round of post-vote and perma-lock.

Asymmetric quorums: Works [60], [61], [62], [63] model

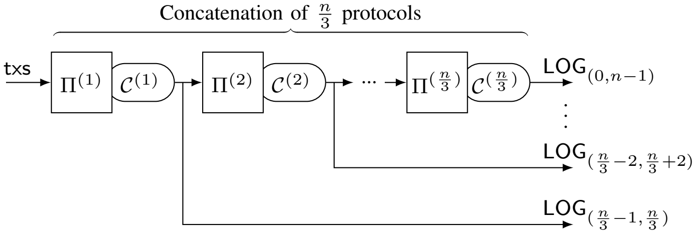

Concatenation of $\frac{n}{3}$ protocols
txs
$\Pi^{(1)}$
$\mathcal{C}^{(1)}$
$\Pi^{(2)}$
$\mathcal{C}^{(2)}$
$\cdots$
$\Pi^{(\frac{n}{3})}$
$\mathcal{C}^{(\frac{n}{3})}$
$\mathsf{LOG}_{(0,n-1)}$
$\vdots$
$\mathsf{LOG}_{(\frac{n}{3}-2,\frac{n}{3}+2)}$
$\mathsf{LOG}_{(\frac{n}{3}-1,\frac{n}{3})}$

Figure 12. Block diagram of an optimal flexible consensus protocol inspired by snap-and-chat (cf. [34, Fig. 5]). This construction is a serial concatenation of $\frac{n}{3}$ consensus protocols (replica logic Π, confirmation rule $\mathcal{C}$). The first protocol $\mathbf{\Pi}^{(1)} = (\Pi^{(1)}, \mathcal{C}^{(1)})$ uses a system-wide quorum $q = \frac{2n}{3}+1$ and achieves system-wide resiliences $t^{\mathrm{L}} = \frac{n}{3} - 1, t^{\mathrm{S}} = \frac{n}{3}$. At each subsequent protocol, the system-wide quorum is increased by 1, which decreases t^{L} by 1 and increases t^{S} by 2 for that protocol, all the way up to $\mathbf{\Pi}^{(\frac{n}{3})} = (\Pi^{(\frac{n}{3})}, \mathcal{C}^{(\frac{n}{3})})$ which has a system-wide quorum $q = n$ and resiliences $t^{\mathrm{L}} = 0, t^{\mathrm{S}} = n - 1$. The log output using $\mathbf{\Pi}^{(1)}$ is treated as input (*i.e.*, 'transactions') to be sequenced by the next protocol $\mathbf{\Pi}^{(2)}$ (a replica 'boycotts' proposals in $\mathbf{\Pi}^{(2)}$ inconsistent with its view of the log of $\mathbf{\Pi}^{(1)}$, cf. [34, Fig. 5]). Thus, $\mathbf{\Pi}^{(2)}$ generates a 'log of logs' which is then flattened and de-duplicated to produce the output log of $\mathbf{\Pi}^{(2)}$, which is then input to $\mathbf{\Pi}^{(3)}$ and so on. Replicas run all protocols simultaneously, which makes this inefficient. A client k with resiliences $(t^{\mathrm{L}}_k, t^{\mathrm{S}}_k)$ confirms $\mathsf{LOG}_{(t^{\mathrm{L}}_k, t^{\mathrm{S}}_k)}$ output by the corresponding protocol and ignores the rest.

that not all replicas may be equally trusted, thereby generalizing the resilience from a fraction of replicas to sets of replicas (called fail-prone sets). In these works, there is no flexiblity. Federated consensus [64], [65] adds flexibility by allowing replicas and clients to choose their own fail-prone sets. However, clients have the same fail-prone sets for both safety and liveness (*i.e.*, both properties fail together when replicas outside the fail-prone sets are adversary), making this an orthogonal direction to flexible consensus.

7. Discussion

7.1. Stronger Consistency Guarantees for OFlex

In the flexible consensus formulation (Def. 1), consistency between clients k, k' is guaranteed only when $f \leq \min\{t^{\mathrm{S}}_k, t^{\mathrm{S}}_{k'}\}$. This follows the definitions in [14], [15]. The previous impossibility result $2t^{\mathrm{L}}_k + t^{\mathrm{S}}_k < n$ bounds the resiliences when $t^{\mathrm{S}}_k = t^{\mathrm{S}}_{k'}$; thus the security guarantees of OFlex protocols are optimal for clients with equal resiliences. But can consistency be guaranteed when $f > \min\{t^{\mathrm{S}}_k, t^{\mathrm{S}}_{k'}\}$ for clients with unequal resiliences? In fact, a closer examination of the proofs of Thm. 1 and Lem. 1 show that the OFlex protocols guarantee consistency between any clients k, k' up to $f = (t^{\mathrm{S}}_k + t^{\mathrm{S}}_{k'})/2$. Conversely, we show in App. B that when the resilience pairs $(t^{\mathrm{L}}_k, t^{\mathrm{S}}_k)$ and $(t^{\mathrm{L}}_{k'}, t^{\mathrm{S}}_{k'})$ are chosen optimally (*i.e.*, each satisfying $2t^{\mathrm{L}}_k + t^{\mathrm{S}}_k = n-1$), then consistency between clients k, k' is impossible if $f > (t^{\mathrm{S}}_k + t^{\mathrm{S}}_{k'})/2$.

7.2. Flexible Accountable Safety

The works [4], [9], [31], [66], [67], [68], [69], [70], [71], [72] strengthen consensus protocols' safety property to *accountable safety*, which guarantees safety if few replicas deviate from the protocol, *and* if safety is ever violated, then a sizable number of replicas are identified to have provably violated the protocol. Our OFlex protocols provide *flexible accountable safety*: if two clients k, k' confirm inconsistent logs (a safety violation), then at least $(t^{\mathrm{S}}_k + t^{\mathrm{S}}_{k'})/2$ replicas must have provably violated the protocol (see Sec. 7.1 to understand the increased resilience compared to Def. 1).

The following rule identifies adversary replicas in the generic OFlex construction: a replica is detected adversary if it sends two *equivocating* post-votes, *i.e.*, two post-votes for inconsistent logs LOG and LOG'. No such post-votes will exist for any honest replica, due to perma-locking, and authentication using signatures. On the other hand, as the quorum-intersection-based safety arguments for OFlex show (cf. Thm. 1 and Lem. 1), a necessary condition for a safety violation between clients k, k' is that at least $(t^{\mathrm{S}}_k + t^{\mathrm{S}}_{k'})/2$ replicas have sent equivocating post-votes.

This rule readily makes safety accountable in the generic OFlex construction (Sec. 3). Adapting the rule to OFlex-Streamlet (Sec. 4.2), if a replica votes for a block whose parent chain contains three adjacent blocks from consecutive epochs, it shall never vote for a block inconsistent with the second block of them (because it is perma-locked on that block). For Ethereum (Sec. 5), if a validator votes for a block whose state commits a certain block as finalized, it shall never vote for a block inconsistent with that finalized block (because it is, effectively, perma-locked on that block).

7.3. Recovering from Liveness Outages

If f exceeds t^{L}_k, client k may lose liveness. If the adversary replicas are only crash faults, then the client can regain liveness either by temporarily increasing t^{L}_k (by decreasing q_k), or by waiting until the crashes are restored. This is because crash faults cannot make honest replicas perma-lock inconsistent logs, so once the number of faults is below t^{L}_k, the client's confirmation quorum will eventually be fulfilled. Similarly, if there are Byzantine faults, but not too many ($t^{\mathrm{L}}_k < f < n/3$), then liveness can be regained because $f < n/3$ adversary replicas cannot make honest replicas perma-lock inconsistent logs.

If $f > n/3 > t^{\mathrm{L}}_k$ are Byzantine, then the base protocol may output inconsistent logs, which the OFlex gadget's perma-locking would 'convert' into a (permanent) liveness violation (note that all clients' liveness resiliences are violated in this case anyway). But in case of such a permanent stall via inconsistency of the base protocol, if the base protocol provides accountable safety (as Casper, Streamlet, and HotStuff do [9], [70], [71]), then at least $n/3$ adversary replicas can be identified. In proof-of-stake blockchains, this makes such an attack expensive, as external repair can be used to slash the stake of the identified replicas.

7.4. Preserving Safety after External Repair

Clients that use OFlex confirmation rules for Ethereum with high q_k retain safety up to high f, *throughout the* *consensus protocol's execution*. However, in case inconsistencies occur in Casper finality or if liveness is lost (which may occur when f exceeds $n/3$), Ethereum intends to use a process *external to the consensus protocol* ('social consensus' [73], [74]) to 'repair' the protocol. Interestingly, this intervention can be carried out in a manner that preserves consistency of high-safety OFlex rules *despite the intervention*, namely if care is taken to not revert transactions deemed confirmed by OFlex rules with high q_k for which no conflicting confirmations have been observed.

Acknowledgments

We thank Daniel J. Aronoff, Dan Boneh, Christian Cachin, Neha Narula, and Ertem Nusret Tas for fruitful discussions. JN, SS, LY, and DT are supported by gifts from the Ethereum Foundation. JN, SS, and DT are supported by a gift from Input Output Global. JN is supported by the Protocol Labs PhD Fellowship.

References

[1] A. Momose and L. Ren, "Multi-threshold Byzantine fault tolerance," in *CCS*. ACM, 2021, pp. 1686–1699.

[2] M. Hirt, A. Kastrati, and C. Liu-Zhang, "Multi-threshold asynchronous reliable broadcast and consensus," in *OPODIS*, ser. LIPIcs, vol. 184. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 6:1–6:16.

[3] M. Raynal, "On the versatility of Bracha's Byzantine reliable broadcast algorithm," *Parallel Process. Lett.*, vol. 31, no. 3, pp. 2 150 006:1–2 150 006:9, 2021.

[4] J. Neu, E. N. Tas, and D. Tse, "The availability-accountability dilemma and its resolution via accountability gadgets," in *Financial Cryptography*, ser. LNCS, vol. 13411. Springer, 2022, pp. 541–559.

[5] A. Ranchal-Pedrosa and V. Gramoli, "Basilic: Resilient-optimal consensus protocols with benign and deceitful faults," in *CSF*. IEEE, 2023, pp. 91–106.

[6] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *OSDI*. USENIX Association, 1999, pp. 173–186.

[7] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A scalable and decentralized trust infrastructure," in *DSN*. IEEE, 2019, pp. 568–580.

[8] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," arXiv:1807.04938v3 [cs.DC], 2018. [Online]. Available: http://arxiv.org/abs/1807.04938v3

[9] V. Buterin and V. Griffith, "Casper the friendly finality gadget," arXiv:1710.09437v4 [cs.CR], 2017. [Online]. Available: http://arxiv.org/abs/1710.09437v4

[10] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *PODC*. ACM, 2019, pp. 347–356.

[11] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *AFT*. ACM, 2020, pp. 1–11.

[12] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[13] J. Neu, E. N. Tas, and D. Tse, "The availability-accountability dilemma and its resolution via accountability gadgets," arXiv:2105.06075v1 [cs.CR], 2021. [Online]. Available: http://arxiv.org/abs/2105.06075v1

[14] D. Malkhi, K. Nayak, and L. Ren, "Flexible Byzantine fault tolerance," in *CCS*. ACM, 2019, pp. 1041–1053.

[15] Z. Xiang, D. Malkhi, K. Nayak, and L. Ren, "Strengthened fault tolerance in Byzantine fault tolerant replication," in *ICDCS*. IEEE, 2021, pp. 205–215.

[16] Twitter user @dannyryan. (2023). [Online]. Available: https://twitter.com/dannyryan/status/1664018289021063168

[17] ——. (2023). [Online]. Available: https://twitter.com/dannyryan/status/1688644951230267392

[18] Reddit user u/Ethical-trade. (2023). [Online]. Available: https://www.reddit.com/r/ethfinance/comments/13wcu4w/comment/jmbhv0z/

[19] Twitter user @superphiz. (2023). [Online]. Available: https://twitter.com/superphiz/status/1663980785693753344

[20] D. Ryan. (2023) The risks of LSD. [Online]. Available: https://github.com/djrtwo/writing/blob/main/docs/2022-05-30_the-risks-of-lsd.md

[21] B. Alpern and F. B. Schneider, "Defining liveness," *Inf. Process. Lett.*, vol. 21, no. 4, pp. 181–185, 1985.

[22] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous BFT protocols," in *CCS*. ACM, 2020, pp. 803–818.

[23] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-NG: Fast asynchronous BFT consensus with throughput-oblivious latency," in *CCS*. ACM, 2022, pp. 1187–1201.

[24] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks," in *NSDI*. USENIX Association, 2022, pp. 493–512.

[25] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*. ACM, 2016, pp. 31–42.

[26] H. Zhang and S. Duan, "PACE: Fully parallelizable BFT from reproposable Byzantine agreement," in *CCS*. ACM, 2022, pp. 3151–3164.

[27] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus," in *EuroSys*. ACM, 2022, pp. 34–50.

[28] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *CCS*. ACM, 2022, pp. 2705–2718.

[29] S. Duan, H. Zhang, X. Sui, B. Huang, C. Mu, G. Di, and X. Wang, "Dashing and Star: Byzantine fault tolerance with weak certificates," Cryptology ePrint Archive, Paper 2022/625, 2022. [Online]. Available: https://eprint.iacr.org/2022/625

[30] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC*. ACM, 2021, pp. 165–175.

[31] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and Casper," arXiv:2003.03052v3 [cs.CR], 2020. [Online]. Available: http://arxiv.org/abs/2003.03052v3

[32] S. Cohen and D. Malkhi. (2020) What they did not teach you in Streamlet. [Online]. Available: https://malkhi.com/posts/2020/12/what-they-didnt-teach-you-in-streamlet/

[33] F. D'Amato. (2022) Casper-FFG as a full protocol and its relationship with Streamlet. [Online]. Available: https://ethresear.ch/t/casper-ffg-as-a-full-protocol-and-its-relationship-with-streamlet/13803

[34] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *SP*. IEEE, 2021, pp. 446–465.

[35] ——, "Two more attacks on proof-of-stake GHOST/Ethereum," in *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, ser. ConsensusDay '22. ACM, 11 2022.

[36] C. Schwarz-Schilling, J. Neu, B. Monnot, A. Asgaonkar, E. N. Tas, and D. Tse, "Three attacks on proof-of-stake Ethereum," in *Financial Cryptography*, ser. LNCS, vol. 13411. Springer, 2022, pp. 560–576.

[37] Ethereum Foundation. (2023). [Online]. Available: https://github.com/ethereum/consensus-specs/blob/cc4c810b8f12a18f3ecfb8f1f969d91b3440eb24/specs/phase0/

[38] S. Agrawal. (2023) Patronum: Ethereum RPC proxy that verifies RPC responses against given trusted block hashes. [Online]. Available: https://github.com/lightclients/patronum

[39] S. Agrawal, J. Neu, E. N. Tas, and D. Zindros, "Proofs of proof-of-stake with sublinear complexity," in *AFT*, ser. LIPIcs, vol. 282. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 14:1–14:24.

[40] (2023) Eth beacon node API. [Online]. Available: https://ethereum.github.io/beacon-APIs/

[41] (2023) Client diversity. [Online]. Available: https://clientdiversity.org

[42] M. Nijkerk. (2023) Ethereum briefly stopped finalizing transactions. What happened? [Online]. Available: https://www.coindesk.com/tech/2023/05/17/ethereums-loss-of-finality-what-happened/

[43] N. Das, T. Tsao, P. V. Loon, Potuz, K. Kirkham, and J. He. (2023) Post-mortem report: Ethereum mainnet finality (05/11/2023). [Online]. Available: https://offchain.medium.com/post-mortem-report-ethereum-mainnet-finality-05-11-2023-95e271dfd8b2

[44] J. Li and D. Mazières, "Beyond one-third faulty replicas in Byzantine fault tolerant systems," in *NSDI*. USENIX, 2007.

[45] D. Kane, A. Fackler, A. Gagol, and D. Straszak, "Highway: Efficient consensus with flexible finality," arXiv:2101.02159v2 [cs.DC], 2021. [Online]. Available: http://arxiv.org/abs/2101.02159v2

[46] X. Dai, L. Huang, J. Xiao, Z. Zhang, X. Xie, and H. Jin, "Trebiz: Byzantine fault tolerance with Byzantine merchants," in *ACSAC*. ACM, 2022, pp. 923–935.

[47] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riché, "Upright cluster services," in *SOSP*. ACM, 2009, pp. 277–290.

[48] V. Zikas, S. Hauser, and U. M. Maurer, "Realistic failures in secure multi-party computation," in *TCC*, ser. LNCS, vol. 5444. Springer, 2009, pp. 274–293.

[49] K. Eldefrawy, J. Loss, and B. Terner, "How Byzantine is a send corruption?" in *ACNS*, ser. LNCS, vol. 13269. Springer, 2022, pp. 684–704.

[50] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "XFT: Practical fault tolerance beyond crashes," in *OSDI*. USENIX Association, 2016, pp. 485–500.

[51] E. Blum, J. Katz, and J. Loss, "Tardigrade: An atomic broadcast protocol for arbitrary network conditions," in *ASIACRYPT (2)*, ser. LNCS, vol. 13091. Springer, 2021, pp. 547–572.

[52] ——, "Synchronous consensus with optimal asynchronous fallback guarantees," in *TCC (1)*, ser. LNCS, vol. 11891. Springer, 2019, pp. 131–150.

[53] Y. Guo, R. Pass, and E. Shi, "Synchronous, with a chance of partition tolerance," in *CRYPTO (1)*, ser. LNCS, vol. 11692. Springer, 2019, pp. 499–529.

[54] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "GearBox: Optimal-size shard committees by leveraging the safety-liveness dichotomy," in *CCS*. ACM, 2022, pp. 683–696.

[55] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *EUROCRYPT (2)*, ser. LNCS, vol. 10821. Springer, 2018, pp. 3–33.

[56] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync HotStuff: Simple and practical synchronous state machine replication," in *SP*. IEEE, 2020, pp. 106–118.

[57] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong, "Zyzzyva: Speculative Byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, pp. 7:1–7:39, 2009.

[58] H. Howard, A. Charapko, and R. Mortier, "Fast flexible Paxos: Relaxing quorum intersection for fast Paxos," in *ICDCN*. ACM, 2021, pp. 186–190.

[59] The Diem Team. (2021) DiemBFT v4: State machine replication in the Diem blockchain. [Online]. Available: https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf

[60] D. Malkhi and M. K. Reiter, "Byzantine quorum systems," *Distributed Comput.*, vol. 11, no. 4, pp. 203–213, 1998.

[61] C. Cachin and B. Tackmann, "Asymmetric distributed trust," in *OPODIS*, ser. LIPIcs, vol. 153. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 7:1–7:16.

[62] I. Damgård, Y. Desmedt, M. Fitzi, and J. B. Nielsen, "Secure protocols with asymmetric trust," in *ASIACRYPT*, ser. LNCS, vol. 4833. Springer, 2007, pp. 357–375.

[63] C. Cachin, G. Losa, and L. Zanolini, "Quorum systems in permissionless networks," in *OPODIS*, ser. LIPIcs, vol. 253. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 17:1–17:22.

[64] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb, "Fast and secure global payments with Stellar," in *SOSP*. ACM, 2019, pp. 80–96.

[65] M. Gabbay and G. Losa, "Semitopology: A new topological model of heterogeneous consensus," arXiv:2303.09287v2 [cs.LO], 2023. [Online]. Available: http://arxiv.org/abs/2303.09287v2

[66] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *SOSP*. ACM, 2007, pp. 175–188.

[67] A. Haeberlen and P. Kuznetsov, "The fault detection problem," in *OPODIS*, ser. LNCS, vol. 5923. Springer, 2009, pp. 99–114.

[68] P. Civit, S. Gilbert, and V. Gramoli, "Polygraph: Accountable Byzantine agreement," in *ICDCS*. IEEE, 2021, pp. 403–413.

[69] A. Ranchal-Pedrosa and V. Gramoli, "ZLB: A blockchain to tolerate colluding majorities," arXiv:2007.10541v3 [cs.DC], 2020. [Online]. Available: http://arxiv.org/abs/2007.10541v3

[70] J. Neu, E. N. Tas, and D. Tse, "Snap-and-chat protocols: System aspects," arXiv:2010.10447v1 [cs.CR], 2020. [Online]. Available: http://arxiv.org/abs/2010.10447v1

[71] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "BFT protocol forensics," in *CCS*. ACM, 2021, pp. 1722–1743.

[72] J. Neu, E. N. Tas, and D. Tse, "Short paper: Accountable safety implies finality," in *Financial Cryptography*, 2024.

[73] Ethereum Foundation. (2023) Ethereum proof-of-stake attack and defense. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/

[74] V. Buterin. (2019) Responding to 51% attacks in Casper FFG. [Online]. Available: https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363

Appendix A. OFlex-Streamlet Security Proof Details

For completeness, we provide a pseudocode of Streamlet in Alg. 3. The pseudocode of OFlex-Streamlet in Alg. 2 highlights its differences with respect to Streamlet.

Proof of Lem. 3. This proof follows techniques from the proof of [11, Lem. 2] and is recapped here for completeness. Denote the heights of the blocks A, B, C as $\ell, \ell+1, \ell+2$ respectively. Denote the epochs of A, B, C as $e, e+1, e+2$ respectively.

Algorithm 3 Streamlet protocol $\mathbf{\Pi}$ [11] (cf. Sec. 4.1)

```
▷ Replica-side logic $\Pi$
on INIT()
    $\mathcal{B}, \mathcal{V} \leftarrow \{B_0\}, \{\}$   ▷ Background task: receive blocks and votes
    into $\mathcal{B}$ and $\mathcal{V}$, respectively, subject to the canonical validation: retain
    only messages with valid signatures; retain only blocks produced by
    respective epoch leader; add messages only once hash pointers can
    be resolved in $\mathcal{B}$; add messages only when their epoch has come
for each epoch $e = 1, 2, 3, ...$
    ▷ Propose (done by epoch leader at the start of the epoch)
    $B' \leftarrow$ tip of any one longest notarized chain in $(\mathcal{B}, \mathcal{V})$
    $h \leftarrow \mathrm{Hash}(B')$
    $\mathsf{txs} \leftarrow$ transactions not present in chain of $B'$
    Sign and broadcast block $(h, e, \mathsf{txs})$
    ▷ Vote (done by all replicas once during the epoch)
    $B \leftarrow$ first block from epoch $e$ in $\mathcal{B}$ signed by epoch leader
    $B' \leftarrow$ parent block of $B$ in $\mathcal{B}$
    if $B'$ is tip of any longest notarized chain in $(\mathcal{B}, \mathcal{V})$
        $h \leftarrow \mathrm{Hash}(B)$
        Sign and broadcast vote $h$

▷ Client-side confirmation rule $\mathcal{C}$
on INIT()
    $\mathcal{B}, \mathcal{V} \leftarrow \{B_0\}, \{\}$   ▷ Background task: receive blocks and votes
    into $\mathcal{B}$ and $\mathcal{V}$, respectively, subject to the canonical validation: retain
    only messages with valid signatures; retain only blocks produced by
    respective epoch leader; add messages only once hash pointers can
    be resolved in $\mathcal{B}$; add messages only when their epoch has come
▷ Confirmation
if $(\mathcal{B}, \mathcal{V})$ contains a notarized chain with three adjacent blocks
    $A, B, C$ from consecutive epochs
    Choose $A, B, C$ as such blocks with maximum height
    $\mathsf{LOG} \leftarrow$ sequence of transactions as ordered in chain of $B$
```

Assume for the sake of contradiction that some block $F \neq B$ with epoch e', at the same height as B, is also notarized in the view of some honest replica. We look at three cases.

First, if $e' \in \{e, e+1, e+2\}$, then in one of these three epochs, at least $n/3+1$ replicas must have voted for two different blocks. This cannot happen if $f \leq n/3$.

Second, let $e' < e$. Since both blocks A and F are notarized, at least $n/3+1$ replicas must have voted for both blocks. Since $f \leq n/3$, at least one of these replicas must be honest. Note that this replica must have voted for F by the end of epoch e' and thus before the beginning of epoch e. This implies that the replica must have observed a notarized chain of length ℓ before the beginning of epoch e (by observing F's notarized parent chain, which is a prerequisite for voting). However, in epoch e, the honest replica voted for block A which has length ℓ. This is a contradiction because the parent chain of A is not one of the longest notarized chains seen by the replica since it had already seen a notarized chain of length ℓ.

Third, let $e' > e+2$. In this case, since both blocks C and F are notarized, at least $n/3+1$ replicas must have voted for both blocks. Since $f \leq n/3$, at least one of these replicas must be honest. Note that this replica must have voted for C by the end of epoch $e+2$ and thus before the beginning of epoch e'. This implies that the replica must have observed a notarized chain of length $\ell+1$ before the beginning of epoch e' (C's notarized parent chain). However, in epoch e', the honest replica voted for block F which has length $\ell+1$. This is a contradiction because the parent chain of F is not one of the longest notarized chains seen by the replica since it had already seen a notarized chain of length $\ell+1$. □

Lemma 6 (cf. [11, Fact 3]). *Suppose that $f < n/3$ and that after GST, there are two epochs e and $e+1$ both with honest leaders denoted L_e and L_{e+1} respectively, and suppose that L_e and L_{e+1} propose blocks B_e and B_{e+1} at heights ℓ_0 and ℓ_1 respectively, it must be that $\ell_1 \geq \ell_0 + 1$.*

Proof. The proof is similar to that of [11, Fact 3], but additionally uses the fact that the voting rule based on the perma-lock is never invoked in the regime of interest for liveness, *i.e.*, $f < n/3$ (Lem. 4). All we need to prove is that the longest notarized chain of L_{e+1} at the beginning of epoch $e+1$ is of length at least ℓ_0. Since L_e proposes a block whose parent chain L_e saw notarized at the beginning of epoch e, due to synchrony, all honest replicas see this notarized chain Δ time into epoch e. Then, every honest replica will vote for B_e unless by time Δ into epoch e, it had already observed a conflicting notarized chain of length ℓ_0 (otherwise, B_e's parent chain is one of the longest notarized chains see by the honest replica, and by Lem. 4, B_e is also a descendant of the replica's perma-lock). If all honest replicas vote for B_e, then since $f \leq n/3$, all honest replicas see the chain ending in B_e (of length ℓ_0) notarized by the beginning of epoch $e+1$. If not, then at least one honest replica had already observed a notarized chain of length ℓ_0 which all honest replicas observe by the beginning of epoch $e+1$. This completes the proof. □

Lemma 7 (cf. [11, Lem. 5]). *Suppose $f < n/3$. After GST, suppose there are three consecutive epochs e, $e+1$, $e+2$ all with honest leaders denoted L_e, L_{e+1}, and L_{e+2}, then the following holds (below we use B to denote the block proposed by L_{e+2} during epoch $e+2$):*

1) *by the beginning of epoch $e+3$, every replica/client will observe a notarized chain ending at B and $n-f$ votes for B (and B had not received $n-f$ votes before the beginning of epoch e);*
2) *furthermore, no conflicting block $F \neq B$ with the same height as B will ever get notarized in the view of any honest replica/client.*

Proof. This proof is identical to that of [11, Lem. 5] and is repeated for completeness. Let ℓ_0, ℓ_1, ℓ_2 be the heights of the blocks proposed by L_e, L_{e+1}, L_{e+2} respectively. By Lem. 6, $\ell_2 > \ell_1 > \ell_0$. Recall that B is the block proposed by L_{e+2} in epoch $e+2$, whose length is ℓ_2.

We now argue that by the beginning of epoch $e+3$, no honest replica voted for a conflicting $F \neq B$ at the same height as B. No honest replica will have voted for a block $F \neq B$ at height ℓ_2 in epochs $e, e+1$ or $e+2$ because the leaders L_e and L_{e+1} proposed blocks at heights ℓ_0 and ℓ_1 respectively, which are different from ℓ_2, and the leader L_{e+2} proposed the block $B \neq F$. If an honest replica had voted for such a block F before epoch e started, at that time this replica must have observed a notarized parent chain of length $\ell_2 - 1$. Due to synchrony after GST, this notarized parent chain of length $\ell_2 - 1$ must have been observed by all honest replicas by the beginning of epoch $e+1$. Therefore, L_{e+1} must propose a block at length at least ℓ_2. Thus we have reached a contradiction.

Since by the beginning of epoch $e+3$, no honest replica has signed any $F \neq B$ at length ℓ_2, at this time there cannot be a notarization for any $F \neq B$ at length ℓ_2. Moreover, L_{e+2}'s proposal and the notarized parent chain that triggered the proposal will be observed by all honest replicas at the beginning of Δ time into epoch $e+2$. Therefore all honest replicas will vote for B by Δ time into epoch $e+2$ (since B's parent is indeed one of the longest notarized chains seen by any replica). Thus by the beginning of epoch $e+3$, all honest replicas will have seen a notarization for B. Thus, no honest replica will ever sign any conflicting $F \neq B$ at length ℓ_2 after the start of epoch $e+3$ either; and any conflicting $F \neq B$ at length ℓ_2 cannot ever gain notarization. □

Proof of Lem. 5. Due to Lem. 7 (1), client k observes the blocks proposed by $L_{e+2}, ..., L_{e+5}$, henceforth denoted $B_2, ..., B_5$, to have received $n - t_k^{\mathrm{L}} \geq q_k$ votes.

Further, the blocks B_2, B_3, B_4, B_5 must form a chain. This is because due to Lem. 7 (2), the leader of epoch $e+3$ observed the chain ending in B_2 as notarized by the beginning of epoch $e+3$ and did not observe any other notarized chains at the same length as B_2. Therefore, the honest leader of epoch $e+3$ must propose block B_3 with the chain ending in B_2 as the parent chain. The same argument holds for blocks B_3, B_4, B_5. Thus, client k, based on its confirmation rule, confirms the block B_3 by the beginning of epoch $e+5$. □

Appendix B. Impossibility Result for Strongly-Consistent Flexible Consensus

In this section, we prove the impossibility result referred to in Sec. 7.1: if two clients k, k' choose optimal resilience pairs, that is, $2t_k^{\mathrm{L}} + t_k^{\mathrm{S}} = n-1$ (and similarly for k'), then consistency between the clients k and k' is impossible if $f > (t_k^{\mathrm{S}} + t_{k'}^{\mathrm{S}})/2$. Observe that due to the optimality of the resilience pairs, this is equivalent to showing that consistency is impossible if $f > (n-1-2t_k^{\mathrm{L}})/2 + (n-1-2t_{k'}^{\mathrm{L}})/2$, *i.e.*, for $f \geq n - t_k^{\mathrm{L}} - t_{k'}^{\mathrm{L}}$. We prove this in Thm. 3. This proof is a generalization of the partially-synchronous resilience trade-off ($2t_k^{\mathrm{L}} + t_{k'}^{\mathrm{S}} < n$) proven in [6], [13].

Theorem 3. *In a partially synchronous network, there is no consensus protocol in which for all clients k, k' with $0 \leq t_k^{\mathrm{L}}, t_{k'}^{\mathrm{L}} \leq n$:*

- ***Liveness:*** *For every* tx *input to all honest replicas, eventually, for all clients k with $f \leq t_k^{\mathrm{L}}$,* $\mathsf{tx} \in \mathsf{LOG}_k$.
- ***Safety:*** *For all clients k, k' with $f \leq n - t_k^{\mathrm{L}} - t_{k'}^{\mathrm{L}}$ for all times τ, τ',* LOG_k^{τ} *and* $\mathsf{LOG}_{k'}^{\tau'}$ *are consistent.*

Proof. Suppose that there is a consensus protocol $\mathbf{\Pi}$ which provides the above mentioned liveness property. We will now prove that there is an execution of the protocol in which the safety property is violated. Suppose that there are two clients k and k'. Let P, Q, R be disjoint subsets of the n replicas such that $|P| = t_k^{\mathrm{L}}$, $|Q| = t_{k'}^{\mathrm{L}}$, and $|R| = n - t_k^{\mathrm{L}} - t_{k'}^{\mathrm{L}}$. Consider the following three worlds:

World 1: A high-entropy transaction tx_1 is sent to all replicas. No other transactions are sent. Replicas in the subsets P and R are honest. Replicas in the subset Q are adversary and do not communicate with the honest replicas and clients k, k'. Since $f \leq t_{k'}^{\mathrm{L}}$ in this world, due to liveness, client k' confirms tx_1, *i.e.*, $\mathsf{tx} \in \mathsf{LOG}_{k'}^{\tau'}$ at some time τ'.

World 2: A high-entropy transaction $\mathsf{tx}_2 \neq \mathsf{tx}_1$ is sent to all replicas. No other transactions are sent. Replicas in the subsets Q and R are honest. Replicas in the subset P are adversary and do not communicate with the honest replicas and clients k, k'. Since $f \leq t_k^{\mathrm{L}}$ in this world, due to liveness, client k confirms tx_2, *i.e.*, $\mathsf{tx}_2 \in \mathsf{LOG}_k^{\tau}$ at some time τ.

World 3: Transaction tx_1 is sent to replicas in P and tx_2 is sent to replicas in Q and both transactions are sent to replicas in R. Replicas in the subsets P and Q are honest. Replicas in R are adversary. The adversary chooses $\mathsf{GST} = \max\{\tau, \tau'\}$ (recall from Sec. 2 that GST is chosen by the adversary and unknown to honest replicas) and until GST, replicas in P and Q cannot communicate with each other. The adversary replicas in R perform a 'split-brain' attack. One brain interacts with P and the client k' as if it only received input tx_1 and this brain does not communicate with replicas in Q and the client k. The other brain interacts with Q and the client k as if it only received input tx_2 and this brain does not communicate with replicas in P and the client k.

For client k', worlds 1 and 3 are indistinguishable, so $\mathsf{tx}_1 \in \mathsf{LOG}_{k'}^{\tau'}$ in world 3. For client k, worlds 2 and 3 are indistinguishable, so $\mathsf{tx}_2 \in \mathsf{LOG}_k^{\tau}$ in world 3. Since tx_1 is high-entropy, $\mathsf{tx}_1 \notin \mathsf{LOG}_k^{\tau}$ because no replicas saw tx_1 in world 2. Similarly, $\mathsf{tx}_2 \notin \mathsf{LOG}_{k'}^{\tau'}$. Thus, LOG_k^{τ} and $\mathsf{LOG}_{k'}^{\tau'}$ are inconsistent. Thus, in world 3, where $f \leq n - t_k^{\mathrm{L}} - t_{k'}^{\mathrm{L}}$, there is a safety violation for clients k and k'. This shows that there cannot be any protocol with the above safety and liveness properties. □