Better Safe than Sorry: Recovering after Adversarial Majority

Srivatsan Sridhar, Dionysis Zindros, and David Tse

Stanford University
{svatsan,dionyziz,dntse}@stanford.edu

Abstract. The security of blockchain protocols is a combination of two properties: safety and liveness. It is well known that no blockchain protocol can provide *both* to sleepy (intermittently online) clients under adversarial majority. However, safety is more critical in that a single safety violation can cause users to lose money. At the same time, liveness must not be lost forever. We show that, in a synchronous network, it is possible to maintain *safety* for all clients even during adversarial majority, *and recover liveness* after honest majority is restored. Our solution takes the form of a *recovery gadget* that can be applied to any protocol with certificates (such as HotStuff, Streamlet, Tendermint, and their variants).

Keywords: Blockchain · Consensus · Recovery.

1 Introduction

Eve the Evil Adversary is at it again. She has somehow managed to capture the *majority* of the stake on our favorite proof-of-stake blockchain. She's about to double-spend: In one transaction, she sends multiplujillion dollars' worth of native tokens to one exchange owned by Alice. In another transaction, she sends the same amount to a different exchange, owned by Bob, spending the exact same money. Using her adversarial majority of voting power, she causes the chain to split into two forks. The first fork causes Alice to confirm the first transaction. The second fork causes Bob to confirm the second transaction. Before the exchanges realize what has transpired, Eve has withdrawn the money from both. As soon as the news hits, the whole community stops transacting: They convene at the bar to analyze what happened and which of the two forks to follow. They slash the misbehaving adversary, but a fight erupts between Alice and Bob: Neither of the two conflicting fork choices can make *both* of them happy.

Here's a simple idea to avoid such a devastating scenario: Alice and Bob re-broadcast any chains they receive from the network, and so if conflicts occur, they are observed by both exchanges. In case of conflict, they each *freeze* their operation and await human intervention. If such a deterring strategy is carefully followed, all honest participants, even those who joined late, always confirm ledgers consistent with one another, retaining safety even under adversarial majority. This simple trick achieves something remarkable: Our protocol

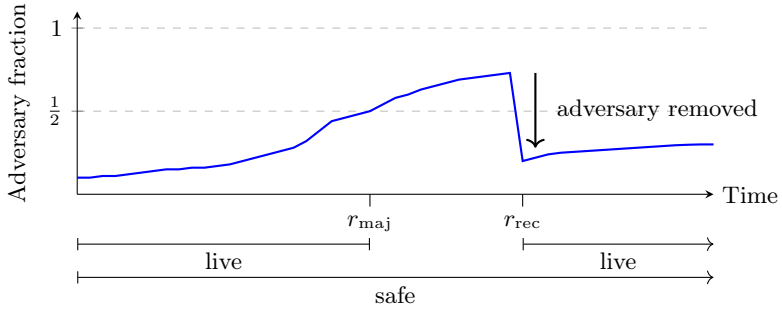
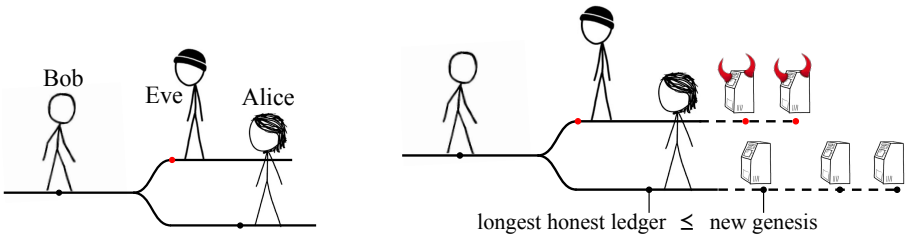


Fig. 1: The timeline of recovery from adversarial majority. Safety is required at all times and liveness must recover once the adversary is removed from the protocol by an external process and honest majority is restored.

lost liveness during adversarial majority but retains safety even up to a 99% adversary, as long as everyone (including clients like Alice and Bob) gossips evidence of conflict and slightly delays confirming ledgers. Retaining safety without liveness makes sense: *safety is more important than liveness*.

Even though we have achieved safety up to a 99% adversary, the money is useless if it can never be spent again. We want liveness to eventually recover. To achieve this, the community who met at the bar uses evidence of misbehavior or other means to exclude the adversary from the protocol so that honest majority is restored (Fig. 1). Still, a challenge remains: Regrettably, Alice and Bob may each freeze their ledger at a different length (Fig. 2a). To unfreeze with human intervention, the community must make a choice of *which ledger to extend*. If they extend the shorter one confirmed by Bob, then Alice, who confirmed a longer ledger, will be unhappy, as she will lose money. They must therefore extend the longer one. However, even though Alice knows in her heart that she's being truthful, the other honest parties may see conflicting evidence. In particular, Eve may claim that *her* view of the ledger is the correct one, and honest parties who are behind cannot tell the difference (Fig. 2a). The situation is frustrating: There *is* a right choice to be made (the longest honest ledger), but this choice is unknowable, even if we try to resolve the conflict socially (see Sec. 3.2).

It was previously believed [7] that such deadlocks are insurmountable and that an arbitrary choice must be made during those perilous circumstances. In this work, we challenge this folklore belief by introducing a new *gadget* which transforms existing protocols to allow for recovery after honest majority heals. The resulting protocol is safe and live under honest majority. When the adversary gains majority (at the moment r_{maj}, see timeline in Fig. 1) and attempts to perform an infraction, our protocol freezes to preserve safety. When the community restores honest majority (for example, by slashing or through a social mechanism), our protocol recovers in a way that preserves safety. At the moment of recovery after honest majority heals (r_{rec} in Fig. 1), which must be truthfully announced as external advice, the protocol computes a *new genesis* block eval-



(a) Freezing protocol (b) Recoverable protocol

Fig. 2: (a) By freezing the ledger, honest parties report ledgers that are prefixes of one another. However, Eve may confuse Bob, who holds a shorter ledger, as to what is its valid honest extension, *i.e.*, the one confirmed by Alice. (b) A recoverable protocol makes a distinction between *confirmed* ledgers (solid lines) and *bookmarked* ledgers (dashed lines). The longest honestly confirmed ledger is a prefix of the shortest bookmarked ledger, and all of them are consistent with each other. The new genesis block can be computed by a majority vote between the bookmarked ledgers held by the validators (illustrated as machines).

uated based on past evidence. The new genesis block is guaranteed to extend the longest ledger confirmed by any honest party, even if the adversary presents confusing evidence. Beyond healing the population and announcing the moment of recovery, the computation of the new genesis block is automatic and internal to the protocol.

Crucial to this result is our network model. We make a distinction between clients and validators: *Validators* form a set of known online participants taking actions, such as voting, in the protocol. *Clients* are unknown and free to join and leave the network at any time (sleepy clients)[1]. Contrary to previous work, which only allowed validators to gossip messages to all parties, we also allow clients to do so. In other words, any message received by a client will soon be received by all online parties. This is a more realistic depiction of currently deployed gossip networks comprised of both clients and validators. The client gossip allows us to bypass previous impossibility results (see Related Work).

Our contributions. In summary, we make the following contributions:

1. *A freezing, always-safe protocol.* Our first, extremely simple, protocol is one that is always *safe*, but *live* only up to a 50% adversary. It involves waiting before confirming a ledger, gossiping evidence of conflict, and freezing on observing evidence of conflict. Despite its straightforward design, it achieves something never seen in the literature before: an always-safe protocol which supports sleepy clients. In the analysis, we note two modeling intricacies: (a) *safety* and *liveness* resilience bounds do not need to be conflated; and (b) in real gossip networks, clients, too, can gossip messages.

[1] Joining and leaving is similar to the sleepy model [32], but with the marked distinction between *sleepy clients* and validators.

2. *An always-safe recoverable protocol.* Our second protocol achieves safety always and liveness before adversarial majority, and after the population has healed from adversarial majority, recovers liveness in a safe manner (as shown in Fig. 1). Remarkably, the protocol can support sleepy clients who inherit similar liveness and safety guarantees whenever they are awake. A new modeling aspect is the introduction of r_{rec}, a moment of recovery, in which the protocol receives trusted external advice that the population has healed, after which recovery can commence.

Construction overview. Our recoverable protocol constitutes a *recovery gadget* built in a black-box fashion on top of any existing distributed ledger protocol (Fig. 3) that provides certificates of confirmation and is secure under honest majority (such as Sync-Hotstuff [1] and Sync-Streamlet [10]). The *internal* protocol Π outputs ledgers that are consumed by the gadget. The gadget *delays* these ledgers twice before reporting them to the external user: Once before they are considered *bookmarked*; and, secondly, before they are considered *confirmed*. Transactions are only considered confirmed when they are part of the confirmed ledger. Any conflicting ledgers that appear in the internal protocol at any point in time are used as a signal to freeze the protocol, and this evidence of infraction is gossiped to the rest of the network to help everyone safely freeze shortly thereafter too. This simple freezing idea ensures all *bookmarked* and *confirmed* ledgers of different honest parties are mutually consistent, ensuring safety even under dishonest majority. The relationship between *bookmarked* and *confirmed* ledgers is that the longest confirmed ledger of clients (solid line in Fig. 2b) is behind the shortest bookmarked ledger of validators (dashed line in Fig. 2b). When the moment for recovery arrives, and the population of validators (shown as machines in Fig. 2b) has been healed of dishonest majority (r_{rec} in Fig. 1), the validators begin a voting phase in which they sign and broadcast their bookmarked ledgers. Clients listen for votes containing each validator's bookmarked ledger and take the prefix of them that is vouched by a majority. This guarantees, due to honest majority, to lead to a ledger ("new genesis", left-most honest machine in Fig. 2b) that contains the longest previously confirmed ledger. Subsequently, the gadget reboots the internal consensus protocol using the ledger discovered in this manner as a new starting point, safely recovering liveness.

Paper structure. Our simple "freezing" idea is presented in Sec. 3.1 where we show that, with *client gossip*, it achieves safety against up to a 99% adversary. We give a detailed description of the scenario of Fig. 2b in Sec. 3.2 which motivates the steps required to design our full recoverable protocol in Sec. 3.3. We prove our protocol achieves the desired properties in Sec. 4. In Sec. 5, we discuss extensions to proof-of-stake and partial synchrony.

Related work. The question of achieving resilience against majority adversaries is not new. Lamport *et al.* [23] in 1982, and Dolev and Strong [16] in 1983 already proposed a broadcast protocol achieving both safety and liveness with resilience up to 100%. Repeated executions make a consensus protocol [6, 21, 11, 20]. Unfortunately, their protocol requires clients to be always online. It is well-known that with clients who may join and leave, it is impossible to achieve *both* safety

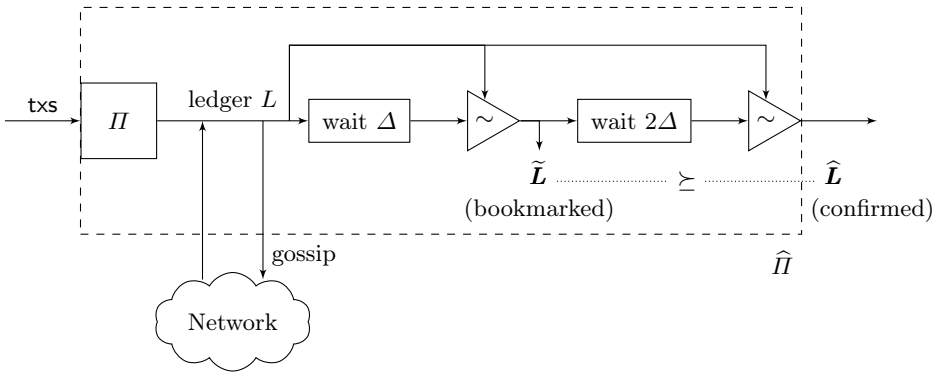


Fig. 3: A simplified block diagram of the recovery gadget. The internal protocol Π is any protocol that is safe and live under honest majority. Ledgers output by Π or obtained from the network are gossiped to the rest of the network. The component $\triangleright\!\sim$ remembers the set of all ledgers it ever received at the input port on its top. On receiving a ledger L at the input port on its left, this block outputs L if no conflicting ledgers were ever received. On waiting and checking for conflicts once, a ledger is considered *bookmarked*. On waiting longer and checking conflicts again, it is considered *confirmed*. Clearly, the bookmarked ledger extends the confirmed ledger.

and liveness with more than 50% resilience [34, 32, 31]. Once safety and liveness are decoupled, [29, 26] prove that safety resilience of 99% would drive liveness resilience to 0%. But the results of [29, 26] critically rely on the fact that clients cannot gossip messages they receive, an unrealistic view of real gossip networks. By utilizing client gossip, our protocol achieves 99% safety resilience and 50% liveness resilience. In our model of client gossip, although a client waking up receives messages sent while it was asleep, it cannot discern the round at which they were sent. This means our model is weaker than a model with always-online clients. All the above results, and our work, are for a synchronous network. In a partially synchronous network, client gossip does not help because gossiped messages may not be delivered in time. However, we discuss in Sec. 5 how our gadget preserves 33% security under partial synchrony while recovering from a 99% adversary under synchrony. The different network models and the safety and liveness resilience achievable and impossible in each model are summarized in a table in App. B. The technique of waiting and confirming on absence of conflict is also used by *validators* in some protocols [1, 10] with 50% resilience, but client gossip is required to push safety to 99%, and even further changes are needed to allow recovery.

Situations of temporary adversarial majority and healing have been explored in both proof-of-work [2] and proof-of-stake [3]. In these treatments, safety and liveness are both temporarily lost and recovered later, which can, regrettably, cause a loss of funds. Some works [33, 37] use accountability [19, 36, 30, 18, 13] to automatically detect some (but not all) kinds of adversaries and remove them

from the set of participants. These works, too, concede safety under adversarial majority. In contrast, we assume that the adversary is removed, automatically or manually, and propose the first protocol that is always safe and recovers liveness.

2 Model

Time. Time proceeds in discrete *rounds* indexed $r = 0, 1, 2, \ldots$. All parties have synchronized local clocks.
Parties. There are two kinds of parties: *validators* and *clients*. At round 0, a set $\mathcal{N}$ of validators awaken and remain awake until they are killed by the environment, whereas clients may awaken at any round, stay awake for an arbitrary number of rounds, and then sleep forever[2]. We denote r_k^{wak} the round in which client k awakens (for validators, $r^{\text{wak}} = 0$). The number of validators $n = |\mathcal{N}|$ is known to everyone, whereas the number of clients is unknown.
PKI. Each validator has a public and private key. The public keys of all validators are known by all parties. All messages sent by a validator are signed using the validator's private key.
Distributed ledger protocol. Validators and clients run a distributed ledger protocol Π. The roles of a validator and client are different. At each round, a validator reads inputs (called *transactions*) from the environment and collaborates with other validators with the aim of placing these transactions into a total order. A client does not receive transactions from the environment, but interacts with the validators and other clients, and at every round, confirms (outputs) a sequence of transactions called the *ledger*. At round 0, the protocol is initialized with the validator set $\mathcal{N}$ and a genesis ledger (which may or may not be empty). All ledgers output by the protocol must extend the genesis ledger.
Network. Validators and clients can send messages to each other. The network is synchronous, *i.e.*, a message sent by an honest party at round r is delivered to honest party p latest by round $\max\{r, r_p^{\text{wak}}\} + \Delta$. Note that this means when a client wakes up at round $r_p^{\text{wak}} > 0$, then it soon receives all messages that were sent to the network before it awoke[3], but the adversary may also inject some of its own messages (similar to the model in [32]). However, different validators or clients may receive a message at different rounds.

The following aspects of the model are new to this work:
Client gossip. In our network model described above, every honest validator and client re-broadcasts every message it receives to the rest of the network. In deviation from previous literature, we even allow *clients* to gossip messages they receive, and validators to possibly act on such messages. On the one hand, this is a more accurate depiction of reality in which communication is facilitated by a non-eclipsed gossip network comprised of both clients and validators. On the other hand, this seemingly insignificant change in the network model enables us

[2] A client that sleeps for a while and awakens again later is treated as if the client sleeps forever and a new client is awakened in its place.
[3] In practice, this can be achieved by having online parties send all important past messages, including equivocations, to a newly-joining client.

to circumvent impossibility results [29, 26].

Time-varying adversarial corruption. Since our work deals with recovering from adversarial majority, we model an adversary whose corruption level varies over time. We denote by $f(r)$ the fraction of validators that are corrupted by the adversary at round r. As seen in Fig. 1, there are two important rounds chosen by the adversary, adaptively: r_{maj} (the *adversarial majority* round) and $r_{\text{rec}} > r_{\text{maj}}$ (the *recovery round*). Before r_{maj}, the adversary maintains $f(r) < \frac{1}{2}$, while during $[r_{\text{maj}}, r_{\text{rec}})$, the adversary may corrupt $f(r) \geq \frac{1}{2}$. Honest parties do not know r_{maj}.

Only at round r_{rec}, the environment is allowed to kill validators to ensure that $f(r_{\text{rec}}) < \frac{1}{2}$, which we describe in the paragraph below. After r_{rec}, the adversary may corrupt more validators, but it is assumed that $f(r) < \frac{1}{2}$ for $r \geq r_{\text{rec}}$. A validator or client, once corrupted, remains corrupted forever, so $f(r)$ is non-decreasing in r, except at round r_{rec}. Similar to the eventual synchrony model [17] in which a single transition from asynchrony to synchrony (GST) is a proxy for alternating periods of asynchrony and synchrony, we use r_{maj} and r_{rec} as a proxy for alternating period of honest majority, adversarial majority, and recovery.

Since, in our model, clients can influence the execution (through client gossip), we also allow the adversary to corrupt any number of clients. The adversary has access to the internal state of all corrupted parties, including private keys.

Healing honest majority. At round r_{rec}, the environment must kill a number of validators such that less than $\frac{1}{2}$ of the *remaining* validators are corrupted. This causes the number of validators n to be updated to a new number of validators $n' \leq n$. In practice, this cleanup may take place by internal *slashing* [7] by the protocol (c.f., *tombstoning* [35]) or by a social consensus process in which adversarial validators are removed by the community. This is akin to classic models of external reconfiguration [38].

At round r_{rec}, the environment sends a special message $\langle \text{recover}, \mathcal{N}' \rangle$ to all parties announcing that recovery must now commence. This message can only be sent at round r_{rec} and includes the public keys of the set $\mathcal{N}'$ of the n' surviving validators. This allows the honest parties to update their PKI. This special message is also sent to all clients that wake up after r_{rec}. An external update of the PKI is, in fact, a software update that can be realized through a "hard fork" [5]. Thus, we follow previous literature [12], which modeled coordination of software updates by a special message sent by the environment announcing the code of the updated protocol and the moment at which the update takes effect. In practice, agreement on such an announcement can be achieved by a community vote external to the protocol [40].

Notation. We use $A \preceq B$ to denote that sequence A is a (not necessarily strict) prefix of the sequence B. We use $A \sim B$ as a shorthand for $A \preceq B \vee B \preceq A$. The ledger confirmed by party p at round r is denoted $\widehat{\boldsymbol{L}}_p^r$.

Security. We define the following properties for the protocol:

Definition 1 (Safety). *A distributed ledger protocol Π is* safe *in a set of rounds I if for all rounds $r, s \in I$ and all honest clients p, q awake at rounds r, s*

respectively, $\widehat{\boldsymbol{L}}_p^r \sim \widehat{\boldsymbol{L}}_q^s$.

Definition 2 (Liveness). *A distributed ledger protocol* Π *is* live *with latency* u *in a set of rounds* I *if for all rounds* $r \in I$*, if a transaction* tx *was received by all honest validators before round* $r - u$*, then for all honest clients* p *with* $r_p^{\text{wak}} < r - u$*,* $\mathsf{tx} \in \widehat{\boldsymbol{L}}_p^r$.

In this work, we develop a protocol that safely recovers from adversarial majority as defined below.

Definition 3. *A distributed ledger protocol* Π *is said to* safely recover from adversarial majority *over an execution of length* R *if for some finite* u, u_{rec}*,* Π *is safe in rounds* $[0, R]$ *and live with latency* u *in rounds* $[0, r_{\text{maj}}) \cup (r_{\text{rec}} + u_{\text{rec}}, R]$.

3 Protocol

In this section, we describe the recovery gadget protocol. As a warmup, we start with the freezing gadget which preserves safety under adversarial majority, but does not permit recovery after honest majority heals. The full recovery gadget builds on this idea and uses the same key components as the freezing gadget.

These gadgets are pieces of code that are meant to be run as an add-on to a distributed ledger protocol Π that provides safety and liveness under honest majority. We will call Π the *internal protocol*. The gadget then specifies actions that clients and validators perform using the output of Π (see Fig. 4). The gadget, when combined with Π, also forms a distributed ledger protocol (validators input transactions and clients confirm a ledger), that we call the *freezing protocol* $\widehat{\Pi}$ (Fig. 4), that provides safety even under adversarial majority. In other words, our constructions are by computational reduction[4]. We will denote the internal protocol's output ledger as $\boldsymbol{L}$ and the freezing protocol's as $\widehat{\boldsymbol{L}}$.

Both the freezing gadget and recovery gadget apply to protocols that satisfy a property called *certifiability* [30, 24]. Several PBFT-style protocols such as Hot-Stuff [39, 25], Streamlet [10], Tendermint [4], Casper [8], and their synchronous variants such as Sync HotStuff [1] and Sync-Streamlet [10, Sec. 4] have this property. In these protocols, clients confirm a ledger on seeing a certain number of blocks with quorum certificates, which forms a *witness*. Any other client, on seeing this witness, irrespective of other messages that it may have seen, considers this ledger to be confirmed. Such protocols have a stronger notion of safety: a minority adversary cannot create witnesses certifying two conflicting ledgers. In our gadgets, clients use witnesses to let each other know about conflicting ledgers that could potentially be confirmed so that they freeze instead of confirming. In what follows, we consider certifiable protocols that are secure under synchrony and honest majority, *e.g.*, Sync-HotStuff [1] and Sync-Streamlet [10, Sec. 4].[5]

[4] Formally speaking, because protocols are executed by multiple parties within an execution, these reductions are more complicated. They can be modelled as a sub-protocol in the form of an Interactive Turing Machine Instance in the UC setting.

[5] These protocols can be made certifiable by having validators broadcast a signature on their committed/finalized ledgers [26, Sec. 4.2].

Definition 4 (Certifiable protocol). *A distributed ledger protocol Π accompanied by a computable functionality $\mathcal{W}$ (the* witness producer*) and a computable deterministic non-interactive function $\mathcal{C}$ (the* witness consumer*), is called* certifiable*. When a client p invokes $\mathcal{W}()$ at round r, it produces a witness w such that $\mathcal{C}(w) = \boldsymbol{L}_p^r$.*

Definition 5 (Certifiable safety). *A certifiable protocol Π accompanied by $\mathcal{W}$ and $\mathcal{C}$ is certifiably safe in a set of rounds I if Π is safe in I, and moreover, if at any round $r \in I$, the adversary outputs a witness w such that $\mathcal{C}(w) = L$, then for all clients q, for all rounds $s \in I$, $L \sim \boldsymbol{L}_q^s$.*

In the following, we will consider any certifiable protocol that is certifiably safe and live under honest majority (Def. 7), in particular safe and live until r_{maj}, as the internal protocol.

3.1 Always Safe: The *Freezing* Gadget

We begin by introducing our freezing gadget. Whereas the construction is simple, the result of an always safe protocol is remarkable: Different honest parties can never reach conflicting conclusions.

Algorithm 1 Freezing Gadget (run by clients)

on INIT($\mathcal{N}, L_{\text{genesis}}$)
  $P \leftarrow$ new $\Pi(\mathcal{N}, L_{\text{genesis}})$ ▷ *instantiate a new Π client*
  $\mathcal{S} \leftarrow \emptyset$ ▷ *set of valid ledgers seen so far*
  $\widehat{\boldsymbol{L}} \leftarrow L_{\text{genesis}}$ ▷ *confirmed (output) ledger of the combined protocol $\widehat{\Pi}$*
on w output by $P.\mathcal{W}()$ once per round or w received from network
  $L \leftarrow \mathcal{C}(w)$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{L\}$
  gossip(w)
  wait(Δ) ▷ *meanwhile, continue processing other events*
  if $L \not\preceq \widehat{\boldsymbol{L}} \wedge \forall L' \in \mathcal{S}\colon L \sim L'$ ▷ *ledger has grown, no conflicting ledgers*
    $\widehat{\boldsymbol{L}} \leftarrow L$

The freezing gadget is described in Alg. 1 and is illustrated as a block diagram in Fig. 4. This gadget, when applied to a certifiable distributed ledger protocol Π (the internal protocol), produces a distributed ledger protocol $\widehat{\Pi}$ (the freezing protocol). The freezing gadget is only run by clients. A client for the freezing protocol $\widehat{\Pi}$ internally runs a client for the internal protocol Π (Π in Fig. 4, Alg. 1 l. 2). All parties are connected to the network (as modeled in Sec. 2).

Each client uses the $\mathcal{W}()$ functionality of the certifiable protocol to periodically output a witness w. It may also receive witnesses in the form of messages sent by honest or adversarial parties in the network. The client extracts a ledger L from the witness (Alg. 1 l. 6), adds L to its set of seen ledgers $\mathcal{S}$ and gossips the

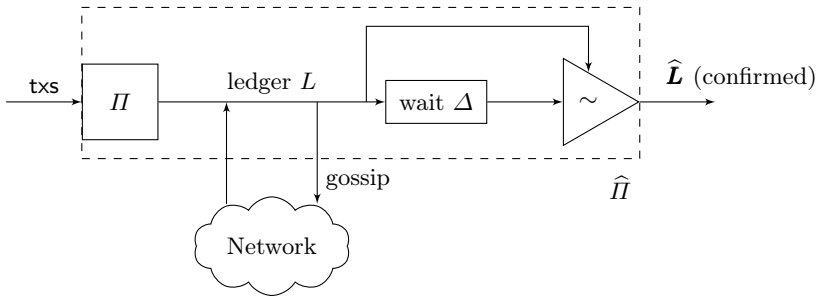


Fig. 4: The freezing gadget. The internal protocol Π is any certifiable distributed ledger protocol. On seeing a ledger L (formally, seeing a witness w such that $\mathcal{C}(w) = L$) output by Π or from the network, the client gossips L (formally, w) to the rest of the network, and then waits for Δ rounds. The *conflict resolution* component $\sim$ remembers the set $\mathcal{S}$ of all ledgers it ever received at the input port on its top. On receiving L at the input port on its left, this component outputs L if there were no conflicting ledgers in $\mathcal{S}$ (see Alg. 1 ll. 10 and 11). The ledger confirmed by the freezing protocol $\widehat{\Pi}$ is denoted $\widehat{\boldsymbol{L}}$.

witness (Alg. 1 l. 8). The client then waits for Δ rounds, during which it continues to process other received witnesses in the same manner (Alg. 1 l. 9). At the end of the wait, the client confirms the ledger L if it has seen no conflicting ledgers and if L is longer than its previously confirmed ledger. (Alg. 1 ll. 10 and 11). Notice that when one client sees conflicting ledgers L, L', after Δ rounds, all clients have added both L and L' to their respective $\mathcal{S}$ sets, and will thereafter not confirm any new ledger, *i.e.*, they *freeze*. However, crucially, they continue to gossip messages and witnesses they see after they have frozen.

The freezing protocol provides safety and liveness until round r_{maj} and retains safety after r_{maj}. We prove these properties below. The freezing protocol's safety comes entirely from the freezing gadget. See Fig. 5 for a visual summary of the proof. Liveness during honest majority is a consequence of liveness and safety of the internal protocol. Liveness of the internal protocol ensures that new transactions are included in its output. Safety of the internal protocol ensures that honest clients do not see witnesses for conflicting ledgers, hence they eventually confirm all ledgers output by Π.

Lemma 1 (Safety). *For any set of rounds I, $\widehat{\Pi}$ is safe in I.*

Proof. See Fig. 5 for reference. Towards contradiction, let r be the smallest round such that for some $s \geq r$, and some honest clients p, q, $\widehat{\boldsymbol{L}}_p^r \not\sim \widehat{\boldsymbol{L}}_q^s$. For shorthand, let $L = \widehat{\boldsymbol{L}}_p^r$. Then, at round $r - \Delta$, client p must have seen a witness w such that $\mathcal{C}(w) = L$. Client p also gossiped w at round $r - \Delta$, which means that before the end of round r, client q must have seen w. Thus, client q added L to its set $\mathcal{S}$ before the end of round r. However, since client q confirmed $\widehat{\boldsymbol{L}}_q^s \not\sim L$ at round $s \geq r$, this is a contradiction to the freezing (Alg. 1 l. 10).

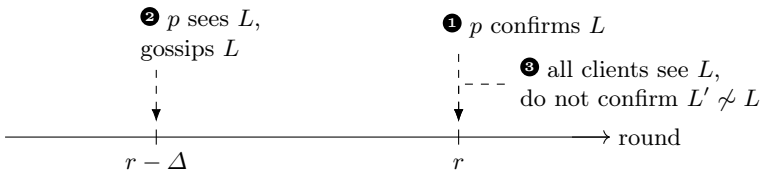


Fig. 5: Illustration for the freezing protocol's safety which is maintained during adversarial majority (Lem. 1). ❶ Suppose that at round r, a client p confirms a ledger L. ❷ The client must have seen L (formally, a witness w) either from the internal protocol Π or from the network latest by round $r - \Delta$, at which point it must have gossipped L (formally, w). ❸ Thus, by round r, all clients must have seen L and thereafter will never confirm a ledger that conflicts with L.

Lemma 2 (Liveness). *If Π is certifiably safe and live with latency u_Π in rounds $[0, r_{\text{maj}})$, then $\widehat{\Pi}$ is live with latency $u_\Pi + \Delta$ in rounds $[0, r_{\text{maj}})$.*

Proof. Let $u = u_\Pi + \Delta$. Let $r < r_{\text{maj}}$ be any arbitrary round. Suppose that a transaction tx is received by all honest validators before round $r - u$. Consider an honest client p that wakes up before $r - u$, *i.e.*, $r_p^{\text{wak}} < r - u$. Due to liveness of Π, at round $s = r - u + u_\Pi$, $\mathsf{tx} \in \boldsymbol{L}_p^s$ (the ledger output by the internal protocol Π). At round s, client p runs $w \leftarrow \mathcal{W}()$ and adds $L = \mathcal{C}(w)$ to its set $\mathcal{S}$(Alg. 1 ll. 5 and 6). Recall from Def. 4 that $L = \boldsymbol{L}_p^s$. Due to certifiable safety, the set $\mathcal{S}$ of client p, at all rounds before r_{maj}, contains only ledgers that are consistent with L. Therefore, at round $s + \Delta = r$, $\widehat{\boldsymbol{L}}_p^r \succeq L \ni \mathsf{tx}$ (due to Alg. 1 ll. 10 and 11).

3.2 Towards Recovery

Given that all clients have frozen, the protocol must recover liveness after r_{rec}. One way to do this is for the new validator set after r_{rec} to restart the protocol from a "new genesis". However, to maintain safety for all clients, it is required that the new genesis contains a ledger that extends the previously confirmed ledgers of all clients. In the scenario described in Fig. 2a, although clients froze to maintain safety, the protocol cannot recover because honest validators are unable to decide on such a ledger. A timeline of events leading up to this scenario is shown in Fig. 6. Due to network delay, different clients see messages at different rounds and in different orders. As a result, while one client, Alice, confirmed a longer ledger before freezing, another client, Bob, may have confirmed only a prefix of Alice's ledger before freezing (as in Fig. 2a).

Knowing that the freezing protocol provides safety against up to 100% adversary, Alice knows that no other honest client would have confirmed any ledger that conflicts with the one she confirmed. However, Bob (whose ledger lags behind) sees two conflicting ledgers, L and L', and does not know which of them may have been confirmed by honest clients. Bob could attempt to find out by asking all other clients which ledger they confirmed. However, there might even be *adversarial clients* who claim that they confirmed L' (Eve in Fig. 2a), while

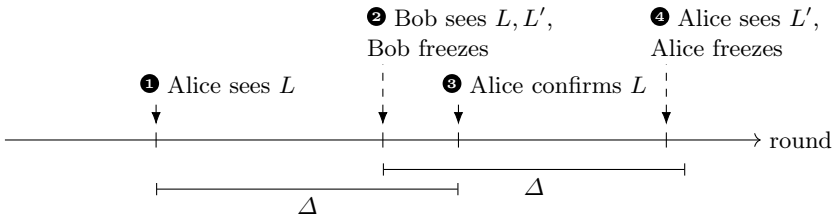


Fig. 6: Timeline of a scenario in which the freezing protocol results in two clients (Alice and Bob) freezing and the last ledger they confirmed are of different lengths. ❶ First, client Alice sees a witness for a ledger L (in short, we say it "sees L"). ❷ Client Bob also sees L within Δ rounds. In addition, Bob sees another ledger L' which is inconsistent with L. As a result, Bob does not confirm L. ❸ Meanwhile, Alice confirms L on waiting Δ rounds after seeing L. ❹ Eventually, Alice also sees L' (due to client gossip) and Alice freezes. The result is that Alice froze after confirming L but Bob froze before confirming L, so they have ledgers that are consistent but of different lengths (as in Fig. 2a).

honest clients like Alice confirmed L. Bob, based on his view, cannot tell which client, Eve or Alice, is lying. Since there may be an arbitrary number of adversarial clients, Bob cannot use a majority vote to decide which ledger was confirmed by honest clients. One might consider running a majority vote on the set of validators, of whom we know that a majority are honest after round r_{rec}. However, it might be the case that all validators have a view similar Bob's, *i.e.*, they cannot tell which ledger was confirmed by honest clients. Thus, even if validators were to run an interactive protocol to reach a consensus on which ledger was confirmed by honest clients, they would not be able to do so. Not knowing which of the two ledgers to extend with new transactions, validators are unable to recover the protocol.

3.3 The *Recovery* Gadget

The recovery gadget (Alg. 2) has two parts. Part 1 is a protocol that is run at all times and is very similar to the freezing gadget, except that both clients and validators run the recovery gadget in different ways. Part 1 is designed to provide a property in addition to safety during adversarial majority that allows honest validators to know which ledgers honest clients could have potentially confirmed. Part 2 is a protocol that is performed only during recovery (which is announced by the environment at round r_{rec}), in which validators collaborate to decide on a new genesis from which to restart the protocol, and clients restart their protocol based on the validators' decision.

In Part 1 (Fig. 3), the code that validators run is identical to that of the freezing gadget, except that validators don't consider the output ledger as confirmed (recall, only clients confirm ledgers). Instead, they consider a ledger as 'bookmarked' (denoted $\widetilde{\boldsymbol{L}}$) when it is output by the freezing gadget (Fig. 3, Alg. 2 l. 2). The bookmarked ledgers are internal to the recovery gadget and

Algorithm 2 Recovery Gadget

▷ *Code for validators*
▷ *Part 1: Same as freezing gadget (Alg. 1 ll. 1 to 11), except replace* $\widehat{\boldsymbol{L}}$ *by* $\widetilde{\boldsymbol{L}}$
▷ *Part 2 (Recovery):*
on receiving $\langle \text{recover}, \mathcal{N}' \rangle$ from the environment ▷ *received at round* r_{rec}
  $\mathcal{S}_{\text{rec}} \leftarrow \emptyset$ ▷ *set of messages delivered during recovery*
  BROADCAST($\langle \text{bookmark}, \widetilde{\boldsymbol{L}} \rangle$)
  wait(u_{BC}) ▷ *wait to deliver bookmarks broadcast by validators in* $\mathcal{N}'$*, see l. 12*
  $\mathcal{S}_{\text{rec}} \leftarrow \{L \in \mathcal{S}_{\text{rec}} \colon |\{L' \in \mathcal{S}_{\text{rec}} \colon L' \succeq L\}| > |\mathcal{N}'|/2\}$
  $L_{\text{rec}} \leftarrow \arg\max_{L \in \mathcal{S}_{\text{rec}}} |L|$ ▷ *longest prefix of ledgers bookmarked by a majority*
  gossip($\langle \text{genesis}, L_{\text{rec}} \rangle$) ▷ *declare the new genesis for clients joining after* r_{rec}
  INIT($\mathcal{N}', L_{\text{rec}}$) ▷ *restart with new validator set and new genesis*
on DELIVER($\langle \text{bookmark}, L \rangle$) broadcast by $V \in \mathcal{N}'$ ▷ *At most one value per* V
  $\mathcal{S}_{\text{rec}} \leftarrow \mathcal{S}_{\text{rec}} \cup \{L\}$

▷ *Code for clients*
▷ *Part 1: Same as freezing gadget (Alg. 1 ll. 1 to 11), except replace* Δ *by* 3Δ
▷ *Part 2 (Recovery):*
on receiving $\langle \text{recover}, \mathcal{N}' \rangle$ from the environment ▷ *received at* $\max\{r_{\text{rec}}, r^{\text{wak}}\}$
  recovering $\leftarrow$ true
  lock($\widehat{\boldsymbol{L}}$) ▷ *forbid future updates to* $\widehat{\boldsymbol{L}}$
  $\mathcal{S}_{\text{gen}} = \{\ \}$
on receiving $\langle \text{genesis}, L \rangle$ from $V \in \mathcal{N}'$ and recovering = true
  $\mathcal{S}_{\text{gen}}[L] = \mathcal{S}_{\text{gen}}[L] + 1$
  if $|\mathcal{S}_{\text{gen}}[L]| > |\mathcal{N}'|/2$
    recovering $\leftarrow$ false
    $L_{\text{rec}} \leftarrow L$
    unlock($\widehat{\boldsymbol{L}}$) ▷ *permit updates to* $\widehat{\boldsymbol{L}}$
    INIT($\mathcal{N}', L_{\text{rec}}$) ▷ *restart with new validator set and new genesis*

their role is to assist validators during recovery. The bookmarked ledgers of different validators are consistent before r_{rec} (due to the freezing gadget's safety). But the bookmarked ledgers after recovery may not remain consistent with those before recovery. However, this doesn't matter since bookmarked ledgers are not confirmed by any party.

The clients' code during Part 1 is also identical to the freezing gadget, except that the client waits 3Δ before confirming a ledger (Alg. 2 l. 15), longer than the Δ that validators wait before bookmarking. This is done in order to provide a property called *follow-the-leader*. Intuitively, by waiting longer to confirm, a client ensures that when it confirms a ledger, all honest validators have already bookmarked it. Thus, at the time of recovery, every honest validator knows that it is safe to restart the protocol from the bookmarked ledger in its view because it is an extension of all clients' confirmed ledgers. Thus, the longest common prefix of all honest validators' bookmarks is a safe new genesis (see Fig. 2b). Recall that $\widehat{\boldsymbol{L}}$ denotes a client's confirmed ledger, and $\widetilde{\boldsymbol{L}}$ a validator's bookmarked ledger.

Definition 6 (Follow-the-leader). *A distributed ledger protocol* Π *has the*

follow-the-leader property if for all rounds r, clients p, and validators v, $\widehat{\boldsymbol{L}}_p^r \preceq \widetilde{\boldsymbol{L}}_v^r$.

A validator begins running Part 2 of the recovery gadget when it receives a "recover" message from the environment at round r_{rec}, specifying the new validator set $\mathcal{N}'$. At this point, each validator broadcasts their own bookmarked ledger (Alg. 2 l. 6). Upon receiving these bookmarks, each validator decides the new genesis L_{rec} as the longest prefix of ledgers bookmarked by a majority of the new validator set (ll. 8 and 9). Due to honest majority among the new validator set, L_{rec} must extend the longest prefix of all honest validators' bookmarks. Finally, the validator gossips a "genesis" message, which is a vote on the new genesis. Clients, on receiving the "recover" message from the environment, at round r_{rec}, or upon waking after r_{rec}, freeze their ledgers if they haven't done so already (l. 19). Clients then wait for "genesis" messages and set their new genesis to be one that is included in the "genesis" messages from a majority of $\mathcal{N}'$. Both clients and validators restart the protocol with the new genesis and validator set (ll. 11 and 27).

During this part, we need to ensure that all honest validators compute the same genesis and that they restart the protocol at the same time. This can be easily achieved by using any solution to the Byzantine generals problem [23] that ensures the following properties under honest majority:

1. If an honest validator BROADCASTs a bookmark (l. 6), all validators DELIVER (l. 12) it at the end of u_{BC} rounds.
2. At the end of u_{BC} rounds, all honest validators have DELIVERed the same set of bookmarks.
3. Each honest validator DELIVERs at most one bookmark per validator.

Solutions to achieve these properties under synchrony are given in [16, 23]. These properties allow all validators to restart their protocol at the same round and with the same genesis.

4 Analysis

We prove that if Π is certifiably safe and live (Defs. 2 and 5) under honest majority, the protocol $\widehat{\Pi}$ resulting from running the recovery gadget (Alg. 2) on $\widehat{\Pi}$ safely recovers from adversarial majority (Def. 3), *i.e.*, it is always safe, live before r_{maj}, and live soon after r_{rec}.

Toward proving safety, we first show that the bookmarked ledgers of all validators at all rounds before r_{rec} are consistent with each other.

Lemma 3. *For all rounds $r_1, r_2 < r_{\text{rec}}$, honest validators v_1, v_2, $\widetilde{\boldsymbol{L}}_{v_1}^{r_1} \sim \widetilde{\boldsymbol{L}}_{v_2}^{r_2}$.*

Proof. Follows from Lem. 1 because the validators' code for bookmarking ledgers is the same as the code for confirming ledgers in the freezing gadget (see Alg. 2 l. 2).

Second, we prove that the follow-the-leader property (Def. 6) holds until r_{rec}.

Lemma 4. *For all $r < r_{\text{rec}}$, honest validators v and honest clients p, $\widehat{\boldsymbol{L}}_p^r \preceq \widetilde{\boldsymbol{L}}_v^r$.*

Proof. Consider an honest client p at let $L = \boldsymbol{L}_p^r$. Then, p saw a witness w such that $\mathcal{C}(w) = L$ by round $r - 3\Delta$. Client p then gossiped w so all validators saw w by round $r - 2\Delta$. Moreover, since p confirmed L, we know that p did not see w' such that $\mathcal{C}(w') = L'$ and $L' \not\sim L$ until round r. This means that no validator saw w' until round $r - \Delta$, because otherwise, the validator would have broadcast w' and client p would have seen w' before round r. Therefore, all validators bookmark L or a ledger extending it by round r.

As a corollary of Lems. 3 and 4, safety holds for all clients until round r_{rec}. Due to the follow-the-leader property, we prove that the new genesis computed by honest validators extends the ledgers confirmed by all honest clients before r_{rec}.

Lemma 5. *For all clients p and all round $r < r_{\text{rec}}$, $\widehat{\boldsymbol{L}}_p^r \preceq L_{\text{rec}}$.*

Proof. The properties of BROADCAST and DELIVER as described in Sec. 3.3 hold. Since all honest validators deliver the same set of bookmarks, each validator computes the same L_{rec}. Let $L_{\text{cp}} = \bigcap_{v \in \text{honest}} \widetilde{\boldsymbol{L}}_v^{r_{\text{rec}}-1}$ be the common prefix of all bookmark ledgers BROADCAST by honest validators. Since each honest validators v BROADCASTs some $\widetilde{\boldsymbol{L}}_v^{r_{\text{rec}}-1} \succeq L_{\text{cp}}$, due to honest majority among $\mathcal{N}'$, fewer than $|\mathcal{N}'|/2$ DELIVERed bookmarks contain any $\widetilde{\boldsymbol{L}} \not\sim L_{\text{cp}}$. Hence, $L_{\text{rec}} \sim L_{\text{cp}}$ (see Alg. 2 l. 8). Moreover, $L_{\text{rec}} \succeq L_{\text{cp}}$ because L_{rec} is the longest ledger whose extensions are bookmarked by $> |\mathcal{N}'|/2$ validators (Alg. 2 l. 9). Due to Lem. 4, for every client p, $\widehat{\boldsymbol{L}}_p^{r_{\text{rec}}-1} \preceq L_{\text{cp}}$. Since confirmed ledgers never shrink, for every round $r < r_{\text{rec}}$, $\widehat{\boldsymbol{L}}_p^r \preceq \widehat{\boldsymbol{L}}_p^{r_{\text{rec}}-1}$. This concludes the proof.

This concludes the proof that safety holds at all times (Thm. 1) since all ledgers confirmed after r_{rec} will extend all ledgers confirmed before r_{rec}.

Theorem 1. *For any set of rounds I, the recoverable protocol $\widehat{\Pi}$ is safe in I.*

Proof. Due to Lems. 3 and 4, for all $r, s < r_{\text{rec}}$ and clients p, q, $\widehat{\boldsymbol{L}}_p^r \sim \widehat{\boldsymbol{L}}_q^s$. Suppose clients p and q restart their protocol (Alg. 2 l. 27) at rounds $r_p, r_q \geq r_{\text{rec}}$ respectively. Since clients freeze their ledgers at r_{rec} (Alg. 2 l. 19), for all $r < r_p$, $\widehat{\boldsymbol{L}}_p^r \preceq \widehat{\boldsymbol{L}}_p^{r_{\text{rec}}-1}$. Thus, for all $r < r_p, s < r_q$, $\widehat{\boldsymbol{L}}_p^r \sim \widehat{\boldsymbol{L}}_q^s$. Due to honest majority in $\mathcal{N}'$ and since all honest validators in $\mathcal{N}'$ decide the same L_{rec}, clients p and q also decide the same L_{rec} in Alg. 2 l. 25. Then, $\widehat{\boldsymbol{L}}_p^r \sim \widehat{\boldsymbol{L}}_q^s$ for $r \geq r_p, s \geq r_q$ as well. Finally, for $r < r_p, s \geq r_q$, due to Lem. 5, $\widehat{\boldsymbol{L}}_p^r \preceq L_{\text{rec}} \preceq \widehat{\boldsymbol{L}}_q^s$, thus $\widehat{\boldsymbol{L}}_p^r \sim \widehat{\boldsymbol{L}}_q^s$.

Liveness before r_{maj} and after r_{rec} follows from the safety and liveness of Π during honest majority and the safety of the recovery process (proof in App. A).

5 Discussion

Proof-of-Stake. We described our gadgets in a permissioned setting for simplicity, but they can be extended to a proof-of-stake setting, where $\mathcal{N}$ is not a set

of validators but a stake distribution. Due to our closed-box treatment of the internal protocol Π, the gadget's working is unaffected by internal updates to the stake distribution by Π (Alg. 2 does not use $\mathcal{N}$ except during recovery). During the moment of recovery, analogous to the permissioned setting, the environment announces a new stake distribution $\mathcal{N}'$. In addition to proof-of-stake implementations of Hotstuff, Casper, and Tendermint, proof-of-stake longest chain protocols [22, 15, 14] can also be made certifiable by defining a new confirmation rule for clients [24]. Thus, our gadget applies to all these protocols.

Partial Synchrony. Consider partially-synchronous PBFT-style protocols (*e.g.*, Casper [8], Tendermint [4], HotStuff [39]). As long as the adversary corrupts less than one-third of the validators, these protocols are live during periods of synchrony and safe even during asynchrony. It is easy to see that applying our recovery gadget to such a protocol does not harm the above guarantees. When the adversary corrupts more than one-third validators, if the network is synchronous, our gadget maintains safety and allows for safe recovery of liveness after the adversary falls below 1/3. However, because our gadget critically uses synchrony, it does not maintain safety in cases where the network is asynchronous and the adversary exceeds 1/3 *at the same time*.

Acknowledgment

We thank Ertem Nusret Tas, Joachim Neu, Zeta Avarikioti for discussions related to our recovery model; Christian Cachin, Giulia Scaffino, and Orfeas Litos for discussions and reviews of early versions of this paper. This research was funded by a Research Hub Collaboration agreement with Input Output Global Inc.

References

1. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync hotstuff: Simple and practical synchronous state machine replication. In: SP. pp. 106–118. IEEE (2020)
2. Avarikioti, G., Käppeli, L., Wang, Y., Wattenhofer, R.: Bitcoin security under temporary dishonest majority. In: Financial Cryptography. LNCS, vol. 11598, pp. 466–483. Springer (2019)
3. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Consensus redux: Distributed ledgers in the face of adversarial supremacy. Cryptology ePrint Archive, Paper 2020/1021 (2020), https://eprint.iacr.org/2020/1021
4. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on bft consensus (2018)
5. Buterin, V.: 51 https://www.youtube.com/watch?v=8DHGOlIlMvc\&t=32m18s, invited talk at the Science of Blockchain Conference 2022
6. Buterin, V.: A guide to 99 https://vitalik.ca/general/2018/08/07/99_fault_tolerant.html
7. Buterin, V.: Responding to 51 https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363
8. Buterin, V., Griffith, V.: Casper the friendly finality gadget. CoRR **abs/1710.09437** (2017)

9. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI. pp. 173–186. USENIX Association (1999)
10. Chan, B.Y., Shi, E.: Streamlet: Textbook streamlined blockchains. In: AFT. pp. 1–11. ACM (2020)
11. Chan, T.H., Pass, R., Shi, E.: Sublinear-round byzantine agreement under corrupt majority. In: Public Key Cryptography (2). LNCS, vol. 12111, pp. 246–265. Springer (2020)
12. Ciampi, M., Karayannidis, N., Kiayias, A., Zindros, D.: Updatable blockchains. In: ESORICS (2). LNCS, vol. 12309, pp. 590–609. Springer (2020)
13. Civit, P., Gilbert, S., Gramoli, V.: Polygraph: Accountable byzantine agreement. In: ICDCS. pp. 403–413. IEEE (2021)
14. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: Financial Cryptography. LNCS, vol. 11598, pp. 23–41. Springer (2019)
15. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: EUROCRYPT (2). LNCS, vol. 10821, pp. 66–98. Springer (2018)
16. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
17. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
18. Haeberlen, A., Kouznetsov, P., Druschel, P.: Peerreview: practical accountability for distributed systems. In: SOSP. pp. 175–188. ACM (2007)
19. Haeberlen, A., Kuznetsov, P.: The fault detection problem. In: OPODIS. LNCS, vol. 5923, pp. 99–114. Springer (2009)
20. Hou, R., Yu, H.: Optimistic fast confirmation while tolerating malicious majority in blockchains. In: SP. pp. 2481–2498. IEEE (2023)
21. Hou, R., Yu, H., Saxena, P.: Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains. In: SP. pp. 1263–1280. IEEE (2022)
22. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO (1). LNCS, vol. 10401, pp. 357–388. Springer (2017)
23. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
24. Lewis-Pye, A., Roughgarden, T.: How does blockchain security dictate blockchain implementation? In: CCS. pp. 1006–1019. ACM (2021)
25. Malkhi, D., Nayak, K.: Extended abstract: Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive, Paper 2023/397 (2023), https://eprint.iacr.org/2023/397
26. Momose, A., Ren, L.: Multi-threshold byzantine fault tolerance. In: CCS. pp. 1686–1699. ACM (2021)
27. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
28. Neu, J., Sridhar, S., Yang, L., Tse, D.: Optimal flexible consensus and its application to ethereum (2023)
29. Neu, J., Tas, E.N., Tse, D.: The availability-accountability dilemma and its resolution via accountability gadgets (2021)
30. Neu, J., Tas, E.N., Tse, D.: The availability-accountability dilemma and its resolution via accountability gadgets. In: Financial Cryptography. LNCS, vol. 13411, pp. 541–559. Springer (2022)

31. Pass, R., Shi, E.: Rethinking large-scale consensus. In: CSF. pp. 115–129. IEEE Computer Society (2017)
32. Pass, R., Shi, E.: The sleepy model of consensus. In: ASIACRYPT (2). LNCS, vol. 10625, pp. 380–409. Springer (2017)
33. Ranchal-Pedrosa, A., Gramoli, V.: Zlb, a blockchain tolerating colluding majorities (2023)
34. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comput. Surv. **22**(4), 299–319 (1990)
35. SDK, C.: Staking tombstone, https://docs.cosmos.network/v0.45/modules/slashing/07_tombstone.html
36. Sheng, P., Wang, G., Nayak, K., Kannan, S., Viswanath, P.: BFT protocol forensics. In: CCS. pp. 1722–1743. ACM (2021)
37. de Souza, L.F., Kuznetsov, P., Rieutord, T., Tucci Piergiovanni, S.: Accountability and reconfiguration: Self-healing lattice agreement. In: OPODIS. LIPIcs, vol. 217, pp. 25:1–25:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
38. Spiegelman, A., Keidar, I., Malkhi, D.: Dynamic reconfiguration: Abstraction and optimal asynchronous solution. In: DISC. LIPIcs, vol. 91, pp. 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
39. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: PODC. pp. 347–356. ACM (2019)
40. Zhang, B., Oliynykov, R., Balogun, H.: A treasury system for cryptocurrencies: Enabling better collaborative intelligence. In: NDSS. The Internet Society (2019)

A Proof Details

Since the recoverable protocol involves initializing a new internal protocol after recovery, we define what it means for the internal protocol to be certifiably safe and live during honest majority. This is the property satisfied by existing certifiable protocols such as Sync-Hotstuff [1] and Sync-Streamlet [10].

Definition 7. *A certifiable protocol Π is* certifiably safe and live with latency u under honest majority *if when Π is initialized at round r with validator set $\mathcal{N}$, Π is certifiably safe and live with latency u in $[r, r^*)$ where $r^* \geq r$ is the first round in which at least $|\mathcal{N}|/2$ validators in $\mathcal{N}$ are corrupted.*

Theorem 2. *If Π is certifiably safe and live with latency u_Π under honest majority, then with $u = u_\Pi + 3\Delta$ and $u_{\text{rec}} = u_\Pi + u_{\text{BC}} + 4\Delta$, for any R, the recoverable protocol $\widehat{\Pi}$ is live with latency u in $[0, r_{\text{maj}}) \cup (r_{\text{rec}} + u_{\text{rec}}, R]$.*

Proof. Consider a round r such that $r < r_{\text{maj}}$ or $r > r_{\text{rec}} + u_{\text{rec}}$. Suppose that all honest validators receive a transaction tx at round $r' \leq r - u$. Consider a client p that awakens at round $r_p^{\text{wak}} \leq r - u$.

- Case 1: $r < r_{\text{maj}}$. The recoverable client protocol is identical to the freezing protocol before round r_{rec}, except with a 3Δ wait instead of Δ (Alg. 2 l. 15). Thus, using Lem. 2, $\widehat{\Pi}$ is live with latency $u_\Pi + 3\Delta$ in $[0, r_{\text{maj}})$.

- Case 2: $r > r_{\text{rec}} + u_{\text{rec}}$, $r' < r_{\text{maj}} - u$. Due to Case 1 and Lem. 5, $\mathsf{tx} \in L_{\text{rec}}$. By round $r_{\text{rec}} + u_{\text{BC}}$, all honest validators send a "genesis" message (Alg. 2 l. 10) and by round $\max\{r_{\text{rec}} + u_{\text{BC}}, r_p^{\text{wak}}\} + \Delta < r$, client p receives all honest "genesis" messages and thus knows L_{rec} (Alg. 2 l. 25). Subsequently, client p restarts its protocol and thus, $\widehat{\boldsymbol{L}}_p^r \succeq L_{\text{rec}} \ni \mathsf{tx}$.
- Case 3: $r > r_{\text{rec}} + u_{\text{rec}}$, $r' \geq r_{\text{rec}} + u_{\text{BC}}$. Note that the instance of Π started at $r_{\text{rec}} + u_{\text{BC}}$ is live after $r_{\text{rec}} + u_{\text{BC}}$ because of honest majority. By round $\max\{r_{\text{rec}} + u_{\text{BC}}, r_p^{\text{wak}}\} + \Delta$, all clients have restarted the protocol, and thus following the argument from Case 1, for $r > r_{\text{rec}} + (u_{\text{BC}} + \Delta) + u_\Pi + 3\Delta$, $\mathsf{tx} \in \widehat{\boldsymbol{L}}_p^r$.
- Case 4: $r > r_{\text{rec}} + u_{\text{rec}}$, $r' \in [r_{\text{maj}} - u, r_{\text{rec}} + u_{\text{BC}})$. A transaction sent at such a round r' may not be confirmed by any client or bookmarked by any validator because Π may not be live during this period. To ensure such transactions are eventually confirmed, validators must carry over pending transactions that were input before $r_{\text{rec}} + u_{\text{BC}}$ and consider them as inputs provided to the new instance of Π at $r_{\text{rec}} + u_{\text{BC}}$. Then, following Case 3, $\mathsf{tx} \in \widehat{\boldsymbol{L}}_p^r$.

B Comparison with Related Work

See Tab. 1.

Table 1: Safety and liveness resiliences achievable and impossible in commonly studied models. The safety resilience t^{S} denotes the maximum adversary fraction up to which the protocol is safe. Similarly, the liveness resilience t^{L} is the maximum adversary fraction up to which the protocol is live. For each model, we state the pareto-optimal set of resiliences and refer to protocols achieving these resiliences as well as works proving that higher resiliences are impossible. From top to bottom, the assumptions in the model weaken and the safety and liveness resiliences decrease. We also refer to protocols that recover liveness after a temporary period when the adversary exceeded t^{L}. Note that for rows in which $t^{\text{S}} < 1$, safety may be lost during recovery.

Model	**Safety and liveness resilience**	**Protocols**	**Impossibility**	**Recovery**
Always-online clients	$t^{\text{L}} = t^{\text{S}} = 1$	[16, 21, 11, 6]	No need	Trivial
Synchrony with client gossip	$t^{\text{S}} = 1, t^{\text{L}} = \frac{1}{2}$	This work (Sec. 3.1)	Future work	This work (Sec. 3.3)
Classic synchrony (no client gossip)	$t^{\text{L}} + t^{\text{S}} \leq 1$ (*e.g.*, $t^{\text{S}} = t^{\text{L}} = \frac{1}{2}$)	[27, 22, 15, 14, 1, 10]	[29, 26]	[2, 3]
Partial synchrony	$2t^{\text{L}} + t^{\text{S}} \leq 1$ (*e.g.*, $t^{\text{S}} = t^{\text{L}} = \frac{1}{3}$)	[9, 39, 10, 4, 8, 28]	[17, 29, 28]	[33]