Consensus Under Adversary Majority Done Right

Srivatsan Sridhar[1], Ertem Nusret Tas[1], Joachim Neu[2], Dionysis Zindros[1,3], and David Tse[1]

[1] Stanford University
{svatsan,nusret,dntse}@stanford.edu
[2] a16z Crypto Research
jneu@a16z.com
[3] Common Prefix
dionyziz@commonprefix.com

Abstract. A spectre is haunting consensus protocols—the spectre of adversary majority. The literature is inconclusive, with possibilities and impossibilities running abound. Dolev and Strong in 1983 showed an early possibility for up to 99% adversaries. Yet, we have known impossibility results for adversaries above 1/2 in synchrony, and above 1/3 in partial synchrony. What gives? It is high time that we pinpoint the culprit of this confusion: the critical role of the modeling details of *clients*. Are the clients *sleepy* or *always-on*? Are they *silent* or *communicating*? Can validators be *sleepy* too? We systematize models for consensus across four dimensions (sleepy/always-on clients, silent/communicating clients, sleepy/always-on validators, and synchrony/partial-synchrony), some of which are new, and tightly characterize the achievable safety and liveness resiliences with matching possibilities and impossibilities for each of the sixteen models. To this end, we unify folklore and earlier results, and fill gaps left in the literature with new protocols and impossibility theorems.

1 Introduction

The field of *Byzantine consensus* presents a seemingly contradictory landscape of claims regarding the *resilience* of protocols—that is, regarding what fraction of parties a protocol can tolerate to deviate from the protocol, while preserving the consensus security-properties safety and liveness. The oft-cited Dolev–Strong protocol [18], along with recent works such as [25,24], famously tolerates up to 99% adversary parties. This stands in stark contrast to the perhaps equally-oft-cited "51% attack" that renders many blockchains based on Nakamoto's longest-chain consensus protocol [17,14][4] insecure as soon as more than half of the parties

SS and ENT contributed equally and are listed alphabetically. A part of Sec. 3.2 appeared in an earlier preprint [54] by SS, DZ, and DT.

[4] By Nakamoto consensus, we here mean longest-chain proof-of-stake (PoS) variants [17,14], not the proof-of-work (PoW) protocol of Bitcoin [43,21]. Some readers may remark that PoS Nakamoto is special because it supports the sleepy model [50]. We'll get to that, but for now treat it as a protocol for synchrony, which it is, too.

are Byzantine. One naturally asks: Why bother with protocols that break under 51% attacks when there are protocols that purportedly tolerate 99% adversary?

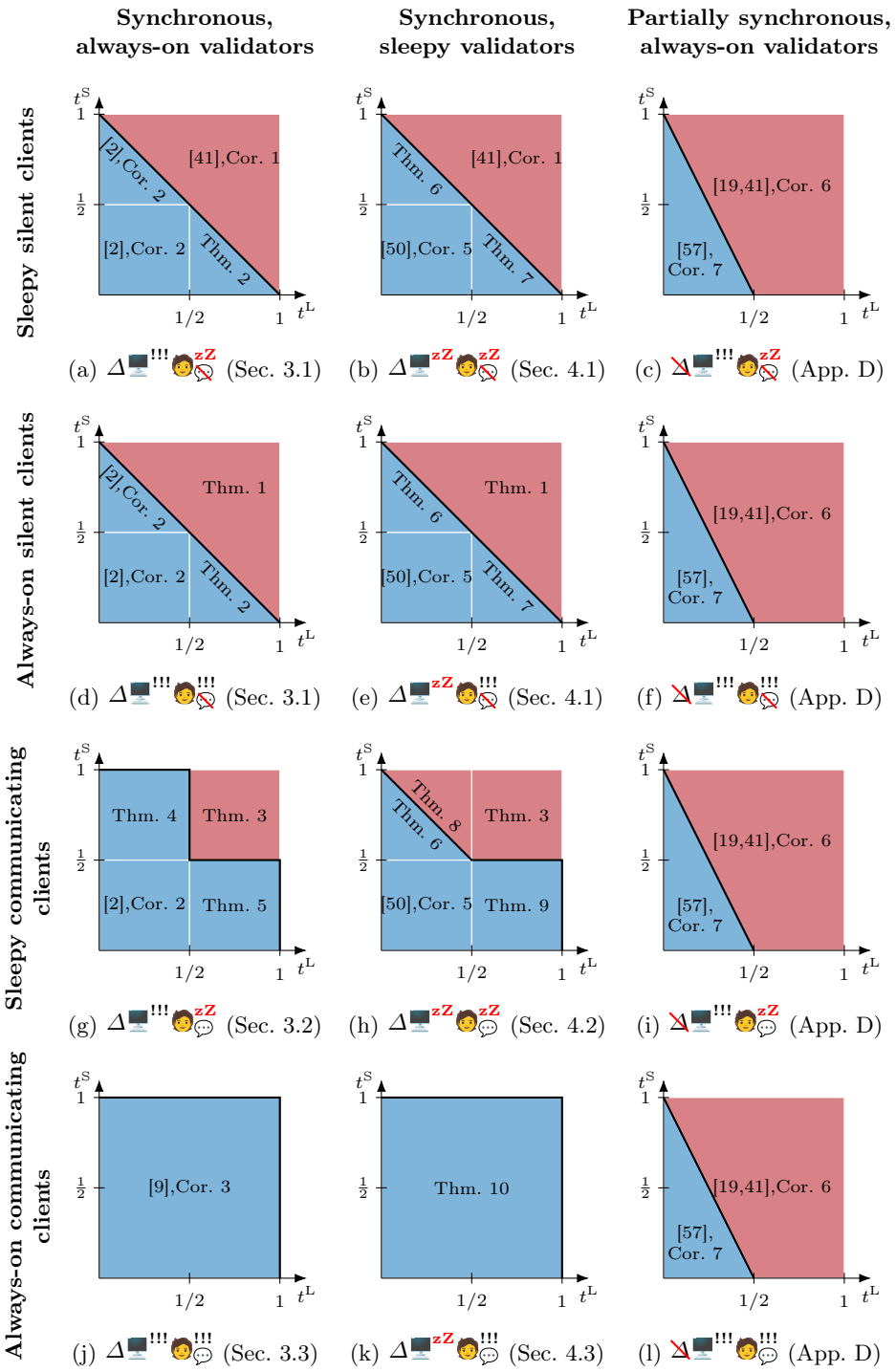
The Crucial Role of Clients. The security guarantees of Dolev–Strong and Nakamoto protocols are both given under synchrony assumptions, so the network model is not the culprit. Rather, it is *client assumptions*. Blockchains comprise not only *validators*——active participants in the consensus protocol, such as stakeholders in a proof-of-stake (PoS) blockchain——but also *clients*. While they do not actively contribute to consensus, they monitor the chain for payments and ship merchandise in response. Two specific characteristics of clients are relevant: (1) *Sleepiness:* Clients may only follow the chain intermittently (*e.g.*, a merchant during business hours), or may turn to a chain only long after its inception. We then call this the *sleepy* client model, in analogy to sleepy validators in [50]. In contrast, in the *always-on* client model, we expect clients to follow the chain continuously, such as in the case of block explorers or wallet providers. (2) *Interactivity:* In the *silent* client model, clients may be constrained to only *listen* to messages from validators. In contrast, in the *communicating* client model, they may be able to relay messages to validators or other clients, for instance, through a system-wide gossip protocol. Consensus is easier when clients are always-on rather than sleepy, and communicating rather than silent.

Nakamoto consensus makes only the weakest client assumptions, *sleepy silent* clients, but achieves only 49% resilience—which is optimal for that model [52,50,49]. In contrast, the Dolev–Strong 99%-resilience holds under the assumption of *always-on communicating* clients, i.e. the strongest client assumptions [9]. (The original Dolev–Strong work was developed in a model with only validators and no clients.) What about the intermediate client assumptions, *sleepy communicating* clients or *always-on silent* clients? What about if the validators themselves can also be sleepy instead of being always-on as in Nakamoto consensus [17,14]? And what about if the network is partially synchronous instead of synchronous?

Our Contributions. The main contribution of this paper is a full characterization of the achievable security in all such scenarios. The results are summarized in Fig. 1 in terms of tight achievable *safety* and *liveness resiliences* under each scenario. *Safety resilience* of a protocol is the maximum fraction of adversary validators such that safety is guaranteed, and *liveness resilience* of a protocol is the maximum fraction of adversary validators such that liveness is guaranteed [41,44]. Traditional *resilience* of a protocol, the maximum fraction of adversary validators such that it is *both safe and live*, is the minimum of the protocol's safety and liveness resiliences. Separate safety and liveness resiliences provide a meaningfully more fine-grained measure of a protocol's security since the impact of safety loss and liveness loss to a client is often different.

Fig. 2 shows the relationship of all the scenarios we considered in this paper.

Synchronous network with always-on validators: The first column of Fig. 1 shows the results in the synchronous network model. Fig. 1j shows that one can achieve 99% resilence when clients are always-on and communicating, i.e., the Dolev–Strong client model. Fig. 1a shows that one can achieve 49% resilence when clients are sleepy and silent, i.e., the Nakamoto client model. Fig. 1d



(a) Δ (Sec. 3.1) (b) Δ (Sec. 4.1) (c) $\not\Delta$ (App. D)

(d) Δ (Sec. 3.1) (e) Δ (Sec. 4.1) (f) $\not\Delta$ (App. D)

(g) Δ (Sec. 3.2) (h) Δ (Sec. 4.2) (i) $\not\Delta$ (App. D)

(j) Δ (Sec. 3.3) (k) Δ (Sec. 4.3) (l) $\not\Delta$ (App. D)

Fig. 1: Tight achievable (■) and impossible (■) safety resilience t^S and liveness resilience t^L bounds for different models (cf. Fig. 2), each with four aspects: *Network delay:* synchrony Δ vs. partial synchrony $\not\Delta$; *Validator sleepiness:* always-on validators $^{!!!}$ vs. sleepy validators ^{zZ}; *Client sleepiness:* always-on clients $^{!!!}$ vs. sleepy clients ^{zZ}; and *Client interactivity:* communicating $_{💬}$ vs. silent $_{🚫}$. Citations with corollaries, or theorems, indicate previously known, or new results, respectively.

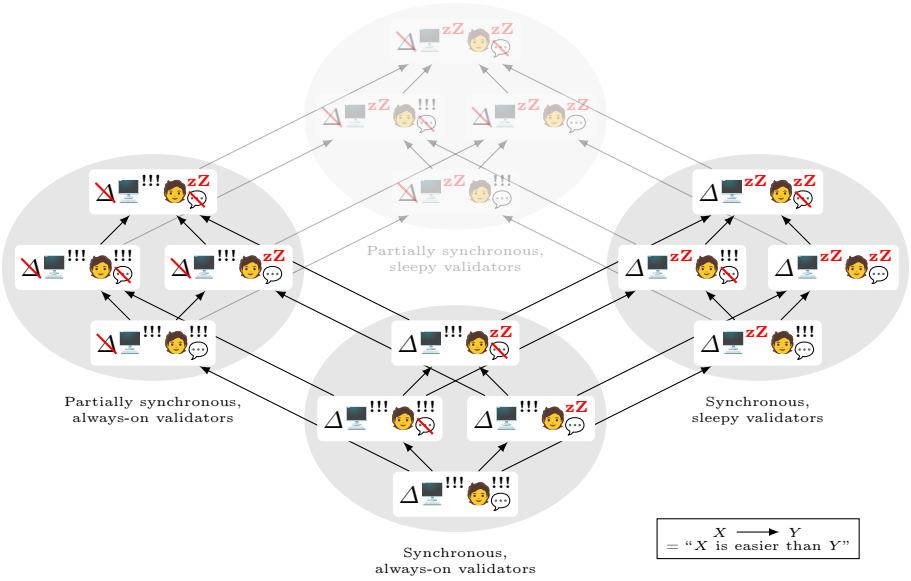


Fig. 2: Hasse diagram illustrating the relative difficulty of consensus in all the different models we study (proof: Lem. 1). Each small white box indicates a different model (see Fig. 1 for icon legend). Models are grouped in a shaded circle when they share validator model and network model but the client model differs. Group ($\not\Delta$, $\not$zZ) is faded out because consensus is impossible [46,22,31].

and Fig. 1g are the two intermediate client settings; our impossibility results show that the achievable resilences in both settings do not improve over the Nakamoto client model, i.e., 49% resilience. However, the similarity ends when one looks at safety and liveness resiliences separately. In particular, we show a new protocol for sleepy communicating clients that can achieve 99% safety resilience and 49% liveness resilience *simultaneously* (Fig. 1g, Thm. 4), a resilience pair that is impossible for sleepy silent clients (Fig. 1a), and strictly dominates classical protocols like Nakamoto's that achieve only 49% safety resilience and 49% liveness resilience. What is more, we show another protocol for sleepy communicating clients that achieves 49% safety and 99% liveness resilience (Fig. 1g, Thm. 5). On the other hand, we show that silent clients do not benefit from being always-on rather than sleepy in synchronous networks even when one considers safety and liveness resiliences separately (Figs. 1a and 1d, Thms. 1 and 2).

Synchonous network with sleepy validators: The concept of "sleepy" parties was previously introduced in the sleepy model [50] where it pertains to validators rather than clients. That model features—in addition to adversary Byzantine faults—relatively-benign mobile crash-faults. Each validator is either honest, crashed, or adversary, and liveness must hold even when many validators are crashed [46,47]. We show that under synchrony with silent or with always-on

communicating clients, validator sleepiness does not affect achievable safety–liveness pairs (Figs. 1a, 1b, 1d, 1e, 1j and 1k). Validator sleepiness *does* affect, though, what's achievable for sleepy communicating clients (Figs. 1g and 1h). Specifically, while 49% safety and 99% liveness resilience remain simultaneously achievable (Fig. 1h, Thm. 9), that is not the case for 99% safety and 49% liveness resilience (Fig. 1h, Thms. 6 and 8).

Partially synchronous network: Finally, the astute reader will note that—whether validators are sleepy or always-on—our discussion above regarding Dolev–Strong and Nakamoto consensus assumes *synchrony*, where the protocol is parameterized by an upper bound Δ on the delay of message propagation among honest parties. In contrast, PBFT-style protocols [11,10,57,53,16] are designed for *partial synchrony*, where such a delay bound is guaranteed to hold only *eventually*, after an initial period of unknown duration with arbitrary network delay. Network delay constitutes the fourth and last aspect of our models (Fig. 2). Interestingly, the safety–liveness resilience pairs achievable under partial synchrony do not depend on client sleepiness or client interactivity (Figs. 1c, 1f, 1i and 1l—with sleepy validators, safety under partial synchrony is impossible [46,22,31]; cf. Fig. 2). This "robustness" of the partially synchronous model to different client assumptions is perhaps why clients have so far often been an afterthought in the distributed-systems literature.

2 Model

We focus on the most relevant model aspects. Other standard aspects: App. A.

Parties. A fixed set $\mathcal{N}$ of parties called *validators* is known to all parties. Define $n = |\mathcal{N}|$. Each validator has a secret key with which they may sign their messages, and all parties know the public keys of all validators. We assume a permissioned setting [32,8,50] with a fixed and known set of validators, and defer the proof-of-stake setting to Sec. 6. Unlike validators, the number and identities of the *clients* is not known to all parties, and clients do not have public keys.

Adversary. At the beginning of the execution, before any randomness is drawn, the PPT adversary $\mathcal{A}$ controls f adversary validators (a.k.a. Byzantine faults). $\mathcal{A}$ may also corrupt any number of clients. All our theorems hold for any number of adversary clients. We discuss adaptive corruption in Sec. 6.

Validator sleepiness (cf. [50]). At every round r, a subset $\mathsf{awake}_r \subseteq \mathcal{N}$ of validators are awake while the rest are *asleep*. When asleep, validators behave like temporarily crashed nodes: they do not run computation or send messages. Whenever a validator is awake, it knows the current round (*i.e.*, it wakes up with a synchronized clock). In the *always-on validators* model, all validators are always awake, $(\forall r\colon \mathsf{awake}_r = \mathcal{N})$. In the *sleepy validators* model, $\mathcal{A}$ selects awake_r, and adversary validators are always awake.

Client Models. We classify clients along two orthogonal criteria. *Interactivity*: *Communicating* clients may send messages to other parties, *silent* clients do not. *Sleepiness*: *Always-on* clients are always awake while *sleepy* clients may

be put to sleep by $\mathcal{A}$. When asleep, clients do not perform any computation, send messages, or output new logs. This results in the four client models shown in Fig. 2. For example, the *sleepy communicating* model means that all clients are sleepy and communicating. Communicating clients are new in this work and more accurately depict blockchain implementations in which communication is facilitated by a non-eclipsed gossip network comprised of both clients and validators. Since clients' messages cannot be authenticated due to their lack of PKI, really the best they can do is relay messages received from validators to other clients. Yet, communicating clients circumvent impossibility results for silent clients (Fig. 1).

Network delay models. We consider two standard network models. In the *synchronous* model, there is a known constant Δ such that if an honest party sends a message at round r, then every honest party receives the message by round $r + \Delta$.[5] The partially synchronous model is described in App. A.

In both models, messages are delivered to asleep parties, but they can only process them after awakening. In practice, equivalent behavior can be achieved by having the awakening party query online parties who reply with the 'important' past messages (*e.g.*, 'initial block download' [5]). Thus, although sleepy parties receive all the same messages as always-on parties, they are less powerful since they cannot record the time of message receipt.

SMR. At the start of each round, each awake party may receive some transactions as input. At the end of every round r, each awake honest client k outputs a log (sequence of transactions) $\boldsymbol{L}_k^r$. For a client k asleep at round r, let $\boldsymbol{L}_k^r = \boldsymbol{L}_k^{r-1}$. For all clients k, $\boldsymbol{L}_k^0 = L_{\text{genesis}}$. We use $A \preceq B$ to denote that log A is a (not necessarily strict) prefix of the log B. We use $A \sim B$ ('A is consistent with B') as a shorthand for $A \preceq B \vee B \preceq A$.

Definition 1 (Safety). *An SMR protocol Π is* safe *iff for all rounds r, s and all honest clients k, k', $\boldsymbol{L}_k^r \sim \boldsymbol{L}_{k'}^s$.*

Definition 2 (Liveness). *An SMR protocol Π is* live *with latency u iff for all rounds r, if a transaction tx was received by an awake honest validator or communicating client before round $r - u$, then for all honest clients p awake during rounds $[r - u, r]$, $\mathsf{tx} \in \boldsymbol{L}_p^r$.*[6]

Definition 3 (Resilience). *For always-on validators, a family of SMR protocols $\Pi(n)$ achieves* safety resilience $t^{\mathrm{S}} \in [0, 1]$ *and* liveness resilience $t^{\mathrm{L}} \in [0, 1]$ *if for all n,*[7] *$\Pi(n)$ is safe with overwhelming probability over executions with $f \leq t^{\mathrm{S}} n$ and live with overwhelming probability over executions with $f \leq t^{\mathrm{L}} n$. For sleepy validators, denote the adversary fraction $\frac{f}{\min_r \mathsf{awake}_r}$ by β. Then, a protocol Π achieves safety resilience $t^{\mathrm{S}} \in [0, 1]$ and liveness resilience $t^{\mathrm{L}} \in [0, 1]$*

[5] Gossip networks have been shown to maintain connectivity, and thus synchrony, even under adversary majority [13,34,35].

[6] Clients may not output new logs for a few rounds after awakening. We use a single parameter u for the maximum of such delay and the protocol's latency.

[7] The number of parties is constrained to be polynomial in the security parameter.

if Π is safe with overwhelming probability over executions with $\beta \leq t^{\mathrm{S}}$ and live with overwhelming probability over executions with $\beta \leq t^{\mathrm{L}}$.

3 Synchrony with Always-On Validators

3.1 Sleepy Silent, Always-On Silent Clients (Fig. 1a, Fig. 1d)

We group the protocols and impossibility results for sleepy silent and always-on silent clients in this section because the results are the same for both (Figs. 1a and 1d). Due to Lem. 1, we prove impossibility results for the easier always-on silent client model and show protocols for the harder sleepy silent client model.

Impossibility for Always-On Silent Clients

Theorem 1. *In a synchronous network with always-on validators and always-on silent clients, no protocol can achieve resiliences $(t^{\mathrm{L}}, t^{\mathrm{S}})$ such that $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$.*

Thm. 1 is due to a split-brain attack. Suppose a protocol has resilience $(t^{\mathrm{L}}, t^{\mathrm{S}})$ such that $t^{\mathrm{L}} + t^{\mathrm{S}} = 1$. Then, the protocol must remain live given $f = t^{\mathrm{L}}n$ adversary validators and safe given $f = (1 - t^{\mathrm{L}})n$ adversary validators. Then, consider a set of $(1 - t^{\mathrm{L}})n$ adversary validators that emulate in their heads two apparently honest executions with two different transactions.[8] These validators can ensure that two clients, each hearing only one of the emulated executions, output different logs. Thus, the protocol cannot ensure safety under $(1 - t^{\mathrm{L}})n = t^{\mathrm{S}}n$ adversary validators, which is a contradiction. Note that the success of the split-brain attack crucially requires the clients to remain isolated, *i.e.*, to be silent. The full proof is in App. B.1.1. For sleepy silent clients, Cor. 1 follows from Thm. 1 and Lem. 1, and a similar proof is also in [41,45].

Corollary 1. *In a synchronous network with always-on validators and sleepy silent clients, no protocol can achieve $(t^{\mathrm{L}}, t^{\mathrm{S}})$ such that $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$.*

Achievability for Sleepy Silent Clients (Safety-Favoring)

Corollary 2. *In a synchronous network with always-on validators and sleepy silent clients, for all $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ and $t^{\mathrm{L}} < 1/2$, Sync HotStuff [2] with a quorum size of $q \in (t^{\mathrm{S}}n, (1 - t^{\mathrm{L}})n]$ achieves $(t^{\mathrm{L}}, t^{\mathrm{S}})$.*

Cor. 2 follows from [2, Theorems 3 and 4], by replacing the quorum sizes by $q \in (n/2, n]$. A similar construction and its security proof can be found in [41]. Other protocols such as Sync Streamlet [12] can also be adapted with quorums $q \in (n/2, n]$ to achieve the same result. Due to Lem. 1, the protocol achieves the same resiliences in a synchronous network with always-on silent clients.

Achievability for Sleepy Silent Clients (Liveness-Favoring) We next describe a family Π^{q}_{live} of protocols (Alg. 1, Fig. 3) parameterized by the integers

[8] Since each execution requires a polynomial amount of computation, a polynomial-time adversary can emulate both these executions.

Algorithm 1 Liveness-favoring SMR protocol Π^q_{live} for sleepy silent clients

```
1  ▷ Code for validator v
2  on INIT(𝒩, L_genesis)
3      P_int ← new Π_int(𝒩, L_genesis)                    ▷ instantiate a new Π_int validator
4  on receiving transaction tx or ⟨tx⟩_v' for some v' ∈ 𝒩
5      gossip(⟨tx⟩_v)                                     ▷ send tx and signature on tx to all parties
6      P_int.input(tx)                                    ▷ input tx to the internal protocol

7  ▷ Client code
8  on INIT(𝒩, L_genesis)
9      P_int ← new Π_int(𝒩, L_genesis)                    ▷ instantiate a new Π_int client
10     Q ← ∅                                              ▷ liveness queue: txs seen so far
11     L ← L_genesis                                      ▷ output log of the combined protocol Π^q_live
12 on {⟨tx⟩_v}_{v∈V} such that V ⊆ 𝒩, |V| ≥ q at round r
13     Q.enqueue((tx, r))          ▷ add tx to the liveness queue on receiving at least q signatures
14 on every round r
15     L_int ← output by P_int at round r
16     for (tx, r') ∈ Q such that r' ≤ r − u_int and tx ∉ L_int
17         L_int ← L_int||tx
18     L ← L_int                                          ▷ output log
```

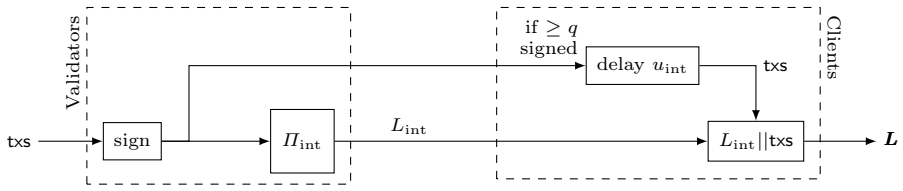


Fig. 3: A family of protocols that achieves any resilience $t^{\text{L}} + t^{\text{S}} < 1$ and $t^{\text{S}} < t^{\text{L}}$ for sleepy silent or always-on silent clients (lower right triangle of Figs. 1a and 1d). The internal protocol Π_{int} is any SMR protocol achieving all resilience pairs $t^{\text{S}} < 1/2, t^{\text{L}} < 1/2$ for sleepy silent clients (*e.g.* Sync HotStuff [2]). On receiving transaction tx, validators sign it and broadcast the signature before processing it as an input to Π_{int}. A client, on receiving transaction tx signed by $q > t^{\text{S}}n$ validators, and after waiting u_{int} rounds (where u_{int} is the maximum latency of Π_{int}), if tx is not included in the log L_{int} output by the client from Π, concatenates tx to L_{int} to output the final confirmed log $\boldsymbol{L}$.

$q \in [0, n/2]$, one for each resilience pair satisfying $t^{\text{L}} + t^{\text{S}} < 1$ and $t^{\text{L}} \geq t^{\text{S}}$. The protocol Π^q_{live} consists of an *internal protocol* Π_{int} and a *liveness queue*. The internal protocol can be any SMR protocol that achieves all $t^{\text{S}} < 1/2, t^{\text{L}} < 1/2$ under synchrony (*e.g.*, Sync HotStuff [2]).

Each honest validator v participates in the internal protocol. Upon receiving a transaction tx for the first time, v signs tx and sends tx and its signature to all parties (Alg. 1 l. 5). Each client locally maintains a liveness queue. If a transaction tx gathers q or more signatures, it is added to the queue (Alg. 1 l. 13). Each client also maintains an *internal log* L_{int} output from the internal protocol (see Fig. 3). To output its log at a round r, a client k appends transactions added to the liveness queue at rounds $r' \leq r - u_{\text{int}}$ (where u_{int} is the internal protocol's

latency) to its internal log at round r, discarding duplicates (Alg. 1 l. 17). The augmented internal log is then output as the log at round r.

Theorem 2. *In a synchronous network with always-on validators and sleepy silent clients, for all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *with* $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ *and* $t^{\mathrm{L}} \geq 1/2$*, the protocol* Π^{q}_{live} *with* $q \in (t^{\mathrm{S}}n, (1 - t^{\mathrm{L}})n]$ *achieves* $(t^{\mathrm{L}}, t^{\mathrm{S}})$.

When $f \leq t^{\mathrm{L}}n$ validators are adversary, all transactions input to an honest validator gather q signatures and enter the liveness queues and eventually enter the output log, ensuring liveness with resilience t^{L}. When $f \leq t^{\mathrm{S}}n$ (which implies $f \leq t^{\mathrm{L}}n$, since $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ and $t^{\mathrm{L}} \geq 1/2$), the internal protocol is safe *and* live, and adversary validators cannot produce q signatures without an honest validator. Therefore, any transaction tx added to the liveness queue must be known to an honest validator and processed by the internal protocol. By the internal protocol's liveness, tx enters the internal log within u_{int} rounds. Thus, no transaction is ever appended to the internal log. Safety then follows from the internal protocol's safety. The full proof is in App. B.1.2

3.2 Sleepy Communicating Clients (Fig. 1g)

Impossibility for Sleepy Communicating Clients

Theorem 3. *In a synchronous network with always-on validators and sleepy communicating clients, no protocol can achieve resiliences* $(t^{\mathrm{L}}, t^{\mathrm{S}}) \in [1/2, 1] \times [1/2, 1]$.

Suppose a protocol can achieve $t^{\mathrm{L}} = t^{\mathrm{S}} = 1/2$. Let P and Q be two disjoint sets of $n/2$ validators and k_1, k_2 be two clients. Consider two worlds where the (P, k_1) and (Q, k_2) are adversary respectively. In both worlds, the adversary parties initially do not communicate with honest parties. By liveness, in world $i \in \{1, 2\}$, client k_i awake since the start outputs transaction tx_i by round u after hearing from the honest validators. In both worlds, a client k_3 awakes after round u and hears from *all parties* including the adversary ones. By liveness, k_3 also outputs tx_i in world i. However, the two worlds are indistinguishable for k_3 because $P - Q$ and $k_1 - k_2$ exchange their roles in the two worlds, implying its log is the same and must contain both tx_1 and tx_2 in both worlds, leading to a safety violation in at least one world. The full proof is in App. B.2.1.

Achievability for Sleepy Communicating Clients (Safety-Favoring) This protocol achieves $(t^{\mathrm{L}}, t^{\mathrm{S}})$ for all $t^{\mathrm{L}} < 1/2$ and $t^{\mathrm{S}} = 1$. In particular, it is *always* safe. It uses a *freezing gadget* applied to an internal SMR protocol Π_{int} that is *certifiable* [47,30] (cf. public verifiability [41]). Quorum-based protocols such as HotStuff [57,38], Streamlet [12], Tendermint [7], Casper [10], and their synchronous variants such as Sync HotStuff [2] and Sync-Streamlet [12, Sec. 4][9] are

[9] The synchronous variants can be made certifiable by having validators broadcast a signature on their 'committed'/'finalized' logs [41, Sec. 4.2].

Algorithm 2 Freezing protocol for sleepy communicating clients

```
1  ▷ Code for client
2  on INIT(𝒩, L_genesis)
3      P_int ← new Π_int(𝒩, L_genesis)                ▷ instantiate a new Π_int client
4      𝒮 ← ∅                                          ▷ set of valid logs seen so far
5      L ← L_genesis                                  ▷ output log of the combined protocol Π_frz
6  on certificate C output by P_int.𝒲() once per round or C received from network
7      L_int ← 𝒞(C)                                   ▷ extract log from certificate
8      𝒮 ← 𝒮 ∪ {L_int}                                ▷ add L_int to set of logs seen so far
9      gossip(C)                                      ▷ send the transcript to all parties
10     wait(Δ)                                        ▷ meanwhile, continue processing other events
11     if L_int ⋠ L and ∀L' ∈ 𝒮: L_int ∼ L'           ▷ log has grown, no conflicting logs
12         L ← L_int
```

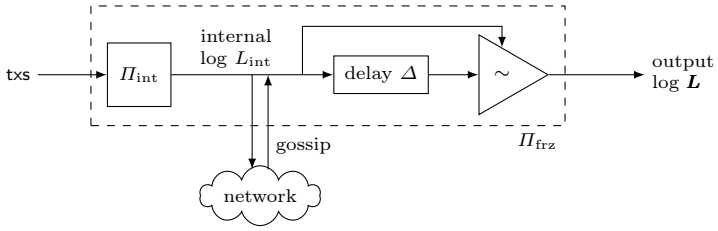


Fig. 4: The freezing protocol Π_{frz} that achieves $t^{\text{L}} < 1/2, t^{\text{S}} = 1$ for sleepy communicating clients. The internal protocol Π_{int} is any certifiable SMR protocol that can achieve all $t^{\text{S}}, t^{\text{L}} < 1/2$. On seeing a log L_{int} from Π_{int} or the network, the client gossips L_{int} (formally, the corresponding certificate), and waits for Δ rounds. The *conflict resolution* component $\boxed{\sim}$ remembers the set $\mathcal{S}$ of all logs 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 logs in $\mathcal{S}$ (see Alg. 2 ll. 11 and 12).

certifiable. In these protocols, clients output a log on receiving enough quorum certificates which form a *certificate* that other clients can verify *non-interactively*. Certifiable safety means that adversaries controlling $\leq t^{\text{S}}$ validators cannot forge certificates certifying two conflicting logs.

Definition 4 (Certifiable protocol). *An SMR protocol Π is certifiable if there exists a computable functionality $\mathcal{W}$ (the* certificate producer*) and a computable deterministic non-interactive function $\mathcal{C}$ (the* certificate consumer*) such that when a client p invokes $\mathcal{W}()$ at round r, it produces a certificate C such that $\mathcal{C}(C) = \boldsymbol{L}_p^r$. A certifiable protocol Π is certifiably safe if Π is safe, and moreover, if at any round r, the adversary outputs a certificate C such that $\mathcal{C}(C) = L$, then for all clients q, for all rounds s, $L \sim \boldsymbol{L}_q^s$. A certifiable protocol Π achieves certifiable safety resilience t^{S} if Π is certifiably safe with overwhelming probability over executions with $f \leq t^{\text{S}} n$.*

The protocol is described in Alg. 2 and is illustrated as a block diagram in Fig. 4. Each client runs a client for the internal protocol Π_{int}, (Π_{int} in Fig. 4, Alg. 2 l. 3) periodically outputs a certificate C for the internal log L_{int}, and

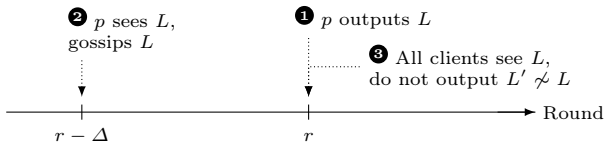


Fig. 5: Illustration for the freezing protocol's safety which is maintained during adversary majority (Thm. 4). ❶ Suppose that at round r, a client p outputs a log L. ❷ The client must have seen L (and its certificate C) either from the internal protocol Π or from the network latest by round $r - \Delta$, at which point it must have sent L (and C) to all other clients. ❸ Thus, by round r, all clients must have seen L and thereafter will never output a log that conflicts with L.

gossips it to the network. It similarly processes certificates received from other clients. After waiting Δ rounds (Alg. 2 l. 10), the client outputs the log L_{int} iff it has seen no conflicting logs (Alg. 2 ll. 11 and 12).

Applying this gadget to an internal protocol Π_{int} with certifiable safety and liveness resilience $t^{\text{L}}_{\text{int}} < 1/2, t^{\text{S}}_{\text{int}} < 1/2$ (*e.g.*, Sync HotStuff [2]) results in a protocol Π_{frz} (Fig. 4) with resilience $t^{\text{L}} < 1/2, t^{\text{S}} = 1$. Safety of Π_{frz} is ensured by the freezing gadget. See Fig. 5 for a visual safety proof. Liveness of Π_{frz} comes from the internal protocol's liveness and certifiable safety, which guarantee that new transactions are included in the internal log and no conflicting certificates are seen. The full proof is in App. B.2.2.

Theorem 4. *In a synchronous network with always-on validators and sleepy communicating clients, for all* $(t^{\text{L}}, t^{\text{S}})$ *with* $t^{\text{L}} < 1/2$ *and* $t^{\text{S}} \leq 1$, Π_{frz} *with Sync HotStuff as its internal protocol achieves* $(t^{\text{L}}, t^{\text{S}})$.

Achievability for Sleepy Communicating Clients (Liveness-Favoring) The protocol Π^{*}_{live} achieves any $(t^{\text{L}}, t^{\text{S}})$ with $t^{\text{L}} = 1, t^{\text{S}} < 1/2$ under always-on validators. The protocol is very similar to Π^{q}_{live} the liveness-favoring protocol for sleepy silent clients (Sec. 3.1) but simpler, so we only describe the key difference. Unlike silent clients in Π^{q}_{live} (Sec. 3.1), *communicating* clients add transactions to their liveness queue as soon as they receive them. This is because while silent clients require signatures from $q > t^{\text{S}}n$ validators to infer that at least one honest validator received the transaction, communicating clients can do so by gossiping the transaction themselves. The full protocol and security proof are in App. B.2.3.

Theorem 5. *In a synchronous network with always-on validators and sleepy communicating clients, for all* $(t^{\text{L}}, t^{\text{S}})$ *with* $t^{\text{L}} \leq 1, t^{\text{S}} < 1/2$, Π^{*}_{live} *with Sync HotStuff as its internal protocol achieves* $(t^{\text{L}}, t^{\text{S}})$.

3.3 Always-On Communicating Clients (Fig. 1j)

Achievability for Always-On Communicating Clients Dolev and Strong [18], and Lamport, Shostak, and Pease [27] presented protocols for the Byzantine

Generals problem when all but one validator are adversary. In their problem setting, there are no clients. Instead, a designated leader ('general') broadcasts a value and all honest validators ('lieutenants') must agree on a common value, which matches the leader's value if the leader is honest. However, the Dolev-Strong protocol can be extended to support always-on communicating clients; the client follows the same rules a validator uses to output a value.

An SMR protocol can be created by having each validator propose a block as the leader in an instance of the Dolev-Strong protocol, allowing always-on communicating clients to agree on a unique block (possibly a default empty block) per leader. The client's log is formed by concatenating these blocks in a set order, repeating the process to grow the log. This approach is presented in [9,25], with optimizations in [24]. We recap the protocol from [9] in Alg. 5 and prove its security in App. B.3.1.

Corollary 3. *In a synchronous network with always-on validators and always-on communicating clients, for all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *with* $t^{\mathrm{L}}, t^{\mathrm{S}} < 1$*, Alg. 5 achieves* $(t^{\mathrm{L}}, t^{\mathrm{S}})$*.*

We note that no protocol can achieve $t^{\mathrm{L}} = t^{\mathrm{S}} = 1$ (cf. App. C.3.2).

4 Synchrony with Sleepy Validators

4.1 Sleepy Silent, Always-On Silent Clients (Fig. 1b,Fig. 1e)

Impossibility for Always-On Silent Clients This follows from Thm. 1 and Lem. 1.

Corollary 4. *In a synchronous network with sleepy validators and always-on silent clients, no protocol can achieve resiliences* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *such that* $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$*.*

Achievability for Sleepy Silent Clients (Equal Resiliences) This follows from Sleepy Consensus [50, Theorem 1] and Goldfish [15, Theorem 2].

Corollary 5. *In a synchronous network with sleepy validators and sleepy silent clients, for all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *with* $t^{\mathrm{L}} < 1/2, t^{\mathrm{S}} < 1/2$*, the Sleepy Consensus protocol [50] and Goldfish [15] achieve* $(t^{\mathrm{L}}, t^{\mathrm{S}})$*.*

Achievability for Sleepy Silent Clients (Safety-Favoring) To achieve any resilience $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ and $t^{\mathrm{L}} < t^{\mathrm{S}}$, we modify the Goldfish protocol [15]. In a nutshell, in Goldfish, voters select a block to vote for by walking down a tree of blocks, and at each fork, selecting the subtree with the largest number of votes from the previous slot. Our key modification is to instead select the subtree with at least ϕ fraction of the total votes from the previous slot, where $\phi \in (t^{\mathrm{S}}, 1 - t^{\mathrm{L}}]$. The details of this protocol are in App. C.1.1.

Theorem 6. *In a synchronous network with sleepy validators and sleepy silent clients, for all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *with* $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ *and* $t^{\mathrm{L}} < 1/2$*, Goldfish modified with* $\phi \in (t^{\mathrm{S}}, 1 - t^{\mathrm{L}}]$ *achieves* $(t^{\mathrm{L}}, t^{\mathrm{S}})$*.*

Achievability for Sleepy Silent Clients (Liveness-favoring) For any resilience $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}}+t^{\mathrm{S}} < 1$ and $t^{\mathrm{L}} \geq t^{\mathrm{S}}$, we show a protocol $\Pi^{\phi}_{\mathrm{live}}$ achieving $(t^{\mathrm{L}}, t^{\mathrm{S}})$. The protocol is very similar to Π^{q}_{live}, the liveness-favoring protocol for sleepy silent clients under always-on validators (Sec. 3.1). The key difference is that to add a transaction to the liveness queue, clients require a fraction ϕ of validators to sign the transaction (unlike a fixed number q in Π^{q}_{live}). More precisely, at every round $r = \ell\Delta$ for some $\ell \in \mathbb{Z}$, each client calculates the number T_{tx} of validators that signed a transaction tx. It also calculates the number $T_{\ell-1}$ of unique validators that have either sent a 'heartbeat' message for round $(\ell-1)\Delta$ or a signature on some transaction in the past. Then, if $T_{\mathsf{tx}}/T_{\ell-1} \geq \phi$, the client adds tx to its liveness queue. The full protocol and security proof are in App. C.1.2.

Theorem 7. *In a synchronous network with sleepy validators and sleepy silent clients, for all $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$ and $t^{\mathrm{L}} \geq 1/2$, the protocol $\Pi^{\phi}_{\mathrm{live}}$ with $\phi \in (t^{\mathrm{S}}, 1-t^{\mathrm{L}}]$ achieves $(t^{\mathrm{L}}, t^{\mathrm{S}})$.*

4.2 Sleepy Communicating Clients (Fig. 1h)

Impossibility for Sleepy Communicating Clients

Theorem 8. *In a synchronous network with sleepy validators and sleepy communicating clients, no protocol can achieve $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}}+t^{\mathrm{S}} \geq 1$ and $t^{\mathrm{S}} \geq 1/2$.*

The proof is similar to Thm. 3. Suppose a protocol can achieve $t^{\mathrm{L}} = 25\%, t^{\mathrm{S}} = 75\%$. Let P and Q be two disjoint sets of $0.75n$ and $0.25n$ validators respectively, and k_1, k_2 be two clients awake since the start. Consider two worlds, 1 and 2, where the (P, k_1) and (Q, k_2) are adversary respectively. In both worlds, the adversary parties initially do not communicate with honest parties. Note that liveness must hold in world 2 because only 25% validators are adversary, and in world 1 because the 75% adversary validators appear indistinguishable from sleepy honest validators. Thus, in each world, client k_i outputs transaction tx_i in its log. In both worlds, a client k_3 awakes after round u and hears from *all parties* including the adversary ones. However, the two worlds are indistinguishable for k_3 because $P - Q$ and $k_1 - k_2$ exchange their roles, implying that its log must be the same in both worlds, and it contains tx_2 as world 2 has liveness, leading to a safety violation in at least one world. The full proof is in App. C.2.1.

Achievability for Sleepy Communicating Clients (Liveness-Favoring) The protocol Π^{*}_{live} in Sec. 3.2 does not rely on the validators for its liveness, and its safety only requires safety and liveness of the internal protocol in a closed-box manner. Therefore, the same protocol, when instantiated with an internal protocol for sleepy validators (*e.g.*, Sleepy Consensus [50]), achieves the following.

Theorem 9. *In a synchronous network with sleepy validators and sleepy communicating clients, for all $(t^{\mathrm{L}}, t^{\mathrm{S}})$ with $t^{\mathrm{L}} \leq 1, t^{\mathrm{S}} < 1/2$, Π^{*}_{live} (Sec. 3.2) with the Sleepy Consensus protocol [50] as its internal protocol achieves $(t^{\mathrm{L}}, t^{\mathrm{S}})$.*

Proof. Follows from [50, Theorem 1] and Lems. 4 and 5. □

4.3 Always-on Communicating Clients (Fig. 1k)

Achievability for Always-on Communicating Clients We show that the SMR protocol based on Dolev-Strong (Sec. 3.3, Alg. 5) achieves any $t^{\mathrm{L}} < 1, t^{\mathrm{S}} < 1$ even under sleepy validators. Under sleepy validators with always-on communicating clients, when a majority of the awake validators are adversary, the clients must output safe and live logs even though the validators themselves may not agree on a log (since validators are sleepy and communicating, the impossibility in Fig. 1h applies to them). Thus, the challenge is to design the validator's code to behave correctly even without knowing what happened while it was sleeping.

This challenge resolves itself due to the following observations. First, while the SMR protocol (Alg. 5) runs instances of Dolev-Strong one after the other, each instance does not depend on the previous instances. Second, within an instance, since the Dolev-Strong protocol (Alg. 4) guarantees agreement and validity when all but one validator are adversary under always-on validators (Lem. 6), it does so even when only one validator is honest and awake throughout the instance (all honest validators who sleep could be considered adversary). Moreover, we don't even require the same honest validator to be awake throughout the instance but only require that for each round during the instance, *some* honest validator is awake. Finally, since each validator (including the leader) signs only one message per instance, it may sleep after it does so without affecting the protocol's remaining execution. Thus, sleepy validators can faithfully run Alg. 4. We explain these surprising observations and prove security in App. C.3.1.

Theorem 10. *In a synchronous network with sleepy validators and always-on communicating clients, for all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *such that* $t^{\mathrm{L}}, t^{\mathrm{S}} < 1$*, Alg. 5 achieves* $(t^{\mathrm{L}}, t^{\mathrm{S}})$.

5 Related Work

No clients. Much of the classic SMR literature [2,1,33,12,50,14,17,26,21] did not explicitly consider clients. Let's call this the 'no clients' model. In this model, validators (a.k.a. 'replicas' or 'nodes') output logs, and in any protocol with resilience $t^{\mathrm{L}} + t^{\mathrm{S}} < 1$, sleepy silent clients may learn the log by querying a quorum of $t^{\mathrm{S}}n + 1$ validators [2,1,33] (see Fig. 1a). However, always-on and/or communicating clients may use other means to learn the log. For example, always-on communicating clients can run the same 'confirmation logic' that validators use. This makes the 'no clients' model equivalent to always-on communicating clients (Fig. 1j). However, under sleepy validators, the 'no clients' model is equivalent to sleepy communicating clients (Fig. 1h), possibly weaker than always-on communicating clients. Reliable broadcast and Byzantine agreement, typically defined without clients, can also apply to different client types (*e.g.*, Def. 10).

Sleepy Silent Clients. Sleepy clients have been called sleepy [50], late-spawning [49,55,14], and lazy [29]. It has been proven that no protocol with sleepy silent clients can

achieve both $t^{\mathrm{L}} \geq \frac{1}{2}$ and $t^{\mathrm{S}} \geq \frac{1}{2}$ [52,50,49]. When safety and liveness are decoupled, [45,41] prove $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$ is impossible. But their proofs require a stronger notion of security, *certifiability*, *i.e.*, the protocol produces *non-interactively* verifiable certificates (cf. Def. 4, [30]). Certifiable protocols are also secure for sleepy silent clients, but the converse is not true (*e.g.*, Nakamoto consensus [43,50,14,17] supports sleepy silent clients, but lack certificates since clients must check for longer chains). Thus, the impossibility results of [45,41] apply to certifiable protocols, but not necessarily to sleepy silent clients. We prove (in Thm. 1) that $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$ is impossible for both sleepy silent and always-on silent clients.

Sleepy communicating clients. The idea of sleepy clients gossiping out-of-band to detect liveness or safety violations isn't new [40], although we are the first to formalize the sleepy communicating client model and apply it to SMR. Validators in some protocols [2,12] with 1/2 resilience also wait Δ to detect conflicts, but achieving $t^{\mathrm{S}} = 99\%$ requires communicating clients. Concurrent work [28] proposes a variant of Bitcoin [43] that is claimed to achieve resilience $t^{\mathrm{L}} < 1/2, t^{\mathrm{S}} = 1$ in the *sleepy validators* model with no clients, equivalent to sleepy validators with sleepy communicating clients. However, in App. E, we show a concrete attack and prove that no protocol can achieve these resiliences under sleepy validators with *sleepy* clients (see also Fig. 1h and Sec. 4.2).[10]

Sleepy validators. Protocols achieving $t^{\mathrm{L}} < 1/2, t^{\mathrm{S}} < 1/2$ for sleepy validators appear in [50,37,15,23,42,51,36]. With sleepy validators and 'no clients', [50,49] prove that no protocol can simultaneously achieve $t^{\mathrm{S}} \geq 1/2, t^{\mathrm{L}} \geq 1/2$.

6 Discussion

Proof-of-Stake. The protocols in Sec. 3.1, Sec. 3.2, Sec. 3.2, Sec. 4.1, Sec. 4.2 are independent of the internal protocol's validator selection mechanism, and can be applied to proof-of-stake. The protocol in Sec. 3.1 is based on Sync HotStuff and Sec. 4.1 on Goldfish [15], and similar protocols are deployed on proof-of-stake Ethereum. The protocol in Sec. 3.3 supports proof-of-stake [9], but the one in Sec. 4.3 doesn't as validators lack knowledge of the log and stake distribution.

Preserving Security under Partial Synchrony. Partially-synchronous PBFT-style protocols (*e.g.*, Casper [10], Tendermint [7], HotStuff [57] or HotStuff2 [38]) maintain safety under asynchrony with resilience $t^{\mathrm{L}} < 1/3, t^{\mathrm{S}} < 1/3$. Applying our freezing gadget (Sec. 3.2) to such a protocol preserves this safety under asynchrony, while making it always safe under synchrony. Safety is only compromised if *both* the network is asynchronous, and the adversary exceeds 1/3.

Adaptive Corruption. Our impossibility results only use static corruption, but all protocols inherit security under adaptive corruption from Sync HotStuff variants [2,41], Dolev-Strong [18], and Goldfish [15].

[10] The model we use assumes a known set of validators of which f are corrupted while the model in [28] instead uses proof-of-work and assumes limited adversary hashing power. Although these models are incomparable, our impossibility proof (Thm. 8) applies to that model as well (see detailed discussion in App. E).

Heterogeneous Clients. Heterogeneous clients are described in [39,56,44]. In the Dolev-Strong-based SMR protocol (Sec. 3.3), always-on communicating clients may output the resulting log achieving any $t^{\mathrm{L}}, t^{\mathrm{S}} < 1$, silent clients may query $\frac{n}{2} + 1$ validators achieving $t^{\mathrm{L}}, t^{\mathrm{S}} < 1/2$, and sleepy communicating clients may further use the freezing gadget (Sec. 3.2) achieving $t^{\mathrm{L}} < 1/2, t^{\mathrm{S}} = 1$ or the liveness gadget (Sec. 3.2) achieving $t^{\mathrm{L}} = 1, t^{\mathrm{S}} < 1/2$ simultaneously.

Liveness-favoring Protocols. Optimistic rollups can benefit from layer 1 blockchains that prioritize liveness since the liveness of the layer 1 is necessary for the timely inclusion of fraud proofs. Although layer 1's safety is also often needed for the rollup's safety, rollups that settle on a different chain than the one they use for ordering the blocks [4] would need only liveness from the settlement chain.

Acknowledgments. SS, ENT, and DT are funded by a Research Hub Collaboration agreement between Stanford University and Input Output Global Inc. ENT is supported by the Stanford Center for Blockchain Research. We thank Zeta Avarikioti, Christian Cachin, Jacob Leshno, Orfeas Stefanos Thyfronitis Litos, Tim Roughgarden, Giulia Scaffino, Elaine Shi, and Roger Wattenhofer for fruitful discussions and feedback.

References

1. Abraham, I., Devadas, S., Nayak, K., Ren, L.: Brief announcement: Practical synchronous byzantine consensus. In: DISC. LIPIcs, vol. 91, pp. 41:1–41:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
2. 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)
3. Abraham, I., Nayak, K.: Dolev-Strong authenticated broadcast (2019), https://decentralizedthoughts.github.io/2019-12-22-dolev-strong/, last accessed: Oct 11, 2024
4. Aditi, Adler, J., Al-Bassam, M.: Quantum gravity bridge: Secure off-chain data availability for Ethereum L2s with Celestia (2022), https://blog.celestia.org/celestiums/, last accessed: Oct 11, 2024
5. Bitcoin Project: Bitcoin developer guide – P2P network – initial block download – headers-first (2020), https://web.archive.org/web/20230314181737/https://developer.bitcoin.org/devguide/p2p_network.html#headers-first
6. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: TCC (1). LNCS, vol. 11891, pp. 131–150. Springer (2019)
7. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938v3 [cs.DC] (2018), http://arxiv.org/abs/1807.04938v3
8. Budish, E., Lewis-Pye, A., Roughgarden, T.: The economic limits of permissionless consensus. arXiv:2405.09173v2 [cs.DC] (2024), http://arxiv.org/abs/2405.09173v2
9. Buterin, V.: A guide to 99% fault tolerant consensus (2018), https://vitalik.eth.limo/general/2018/08/07/99_fault_tolerant.html, last accessed: Oct 11, 2024
10. Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv:1710.09437v4 [cs.CR] (2017), http://arxiv.org/abs/1710.09437v4
11. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)

12. Chan, B.Y., Shi, E.: Streamlet: Textbook streamlined blockchains. In: AFT. pp. 1–11. ACM (2020)
13. Coretti, S., Kiayias, A., Moore, C., Russell, A.: The generals' scuttlebutt: Byzantine-resilient gossip protocols. In: CCS. pp. 595–608. ACM (2022)
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. D'Amato, F., Neu, J., Tas, E.N., Tse, D.: Goldfish: No more attacks on Ethereum?! In: Financial Cryptography (2024), https://eprint.iacr.org/2022/1171
16. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In: EuroSys. pp. 34–50. ACM (2022)
17. 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)
18. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
19. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
20. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distributed Comput. **1**(1), 26–39 (1986)
21. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. J. ACM **71**(4), 25:1–25:49 (2024)
22. Gilbert, S., Lynch, N.A.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002)
23. Goyal, V., Li, H., Raizes, J.: Instant block confirmation in the sleepy model. In: Financial Cryptography (2). LNCS, vol. 12675, pp. 65–83. Springer (2021)
24. Hou, R., Yu, H.: Optimistic fast confirmation while tolerating malicious majority in blockchains. In: SP. pp. 2481–2498. IEEE (2023)
25. Hou, R., Yu, H., Saxena, P.: Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains. In: SP. pp. 1263–1280. IEEE (2022)
26. 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)
27. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
28. Leshno, J.D., Shi, E., Pass, R.: On the viability of open-source financial rails: Economic security of permissionless consensus. arXiv:2409.08951v1 [cs.GT] (2024), http://arxiv.org/abs/2409.08951v1
29. Lewis-Pye, A.: Consensus in 50 pages (rough draft) (2022), https://lewis-pye.com/wp-content/uploads/2023/01/consensus-in-50-pages7-1.pdf, last accessed: Oct 11, 2024
30. Lewis-Pye, A., Roughgarden, T.: How does blockchain security dictate blockchain implementation? In: CCS. pp. 1006–1019. ACM (2021)
31. Lewis-Pye, A., Roughgarden, T.: Byzantine generals in the permissionless setting. In: FC (1). LNCS, vol. 13950, pp. 21–37. Springer (2023)
32. Lewis-Pye, A., Roughgarden, T.: Permissionless consensus. arXiv:2304.14701v5 [cs.DC] (2023), http://arxiv.org/abs/2304.14701v5
33. Liu, S., Viotti, P., Cachin, C., Quéma, V., Vukolic, M.: XFT: practical fault tolerance beyond crashes. In: OSDI. pp. 485–500. USENIX Association (2016)

34. Liu-Zhang, C., Matt, C., Maurer, U., Rito, G., Thomsen, S.E.: Practical provably secure flooding for blockchains. In: ASIACRYPT (1). LNCS, vol. 13791, pp. 774–805. Springer (2022)
35. Liu-Zhang, C., Matt, C., Thomsen, S.E.: Asymptotically optimal message dissemination with applications to blockchains. In: EUROCRYPT (3). LNCS, vol. 14653, pp. 64–95. Springer (2024)
36. Losa, G., Gafni, E.: Consensus in the unknown-participation message-adversary model. arXiv:2301.04817v2 [cs.DC] (2023), http://arxiv.org/abs/2301.04817v2
37. Malkhi, D., Momose, A., Ren, L.: Towards practical sleepy BFT. In: CCS. pp. 490–503. ACM (2023)
38. 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
39. Malkhi, D., Nayak, K., Ren, L.: Flexible byzantine fault tolerance. In: CCS. pp. 1041–1053. ACM (2019)
40. Mazières, D., Shasha, D.E.: Building secure file systems out of byzantine storage. In: PODC. pp. 108–117. ACM (2002)
41. Momose, A., Ren, L.: Multi-threshold byzantine fault tolerance. In: CCS. pp. 1686–1699. ACM (2021)
42. Momose, A., Ren, L.: Constant latency in sleepy consensus. In: CCS. pp. 2295–2308. ACM (2022)
43. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), https://bitcoin.org/bitcoin.pdf, last accessed: Oct 11, 2024
44. Neu, J., Sridhar, S., Yang, L., Tse, D.: Optimal flexible consensus and its application to Ethereum. In: SP. pp. 3885–3903. IEEE (2024)
45. Neu, J., Tas, E.N., Tse, D.: The availability-accountability dilemma and its resolution via accountability gadgets. arXiv:2105.06075v1 [cs.CR] (2021), http://arxiv.org/abs/2105.06075v1
46. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: SP. pp. 446–465. IEEE (2021)
47. 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)
48. Okun, M.: On the round complexity of byzantine agreement without initial set-up. Inf. Comput. **207**(12), 1351–1368 (2009)
49. Pass, R., Shi, E.: Rethinking large-scale consensus. In: CSF. pp. 115–129. IEEE Computer Society (2017)
50. Pass, R., Shi, E.: The sleepy model of consensus. In: ASIACRYPT (2). LNCS, vol. 10625, pp. 380–409. Springer (2017)
51. Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: EUROCRYPT (2). LNCS, vol. 10821, pp. 3–33. Springer (2018)
52. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comput. Surv. **22**(4), 299–319 (1990)
53. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: DAG BFT protocols made practical. In: CCS. pp. 2705–2718. ACM (2022)
54. Sridhar, S., Zindros, D., Tse, D.: Better safe than sorry: Recovering after adversarial majority. Cryptology ePrint Archive, Paper 2023/1556 (2023), https://eprint.iacr.org/2023/1556
55. Tas, E.N., Tse, D., Gai, F., Kannan, S., Maddah-Ali, M.A., Yu, F.: Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In: SP. pp. 126–145. IEEE (2023)

56. Xiang, Z., Malkhi, D., Nayak, K., Ren, L.: Strengthened fault tolerance in byzantine fault tolerant replication. In: ICDCS. pp. 205–215. IEEE (2021)
57. 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)

A Model

In this section, we describe the model and notation used in the rest of the paper. A state-machine replication (SMR) consensus protocol is a distributed protocol run among two types of parties: *validators* and *clients*. Validators take inputs called *transactions* and enable clients to agree on a sequence of confirmed transactions called the *log*.

Time. Time proceeds in logical units called *rounds* indexed by $r = 0, 1, 2, \ldots$. We assume the maximum clock offset between any two parties is bounded (any bounded clock offset can be absorbed into the network delay bound [50]).

Cryptography. We assume an ideal random oracle as a common source of randomness for the protocols. We assume probabilistic polynomial-time (PPT) adversaries and the existence of collision-resistant hash functions and unforgeable digital signatures.

Validators. There is a fixed set $\mathcal{N}$ of parties called *validators* known to all parties. We denote the number of validators by $n = |\mathcal{N}|$. We assume a public key infrastructure (PKI): each validator has a public key and a secret key with which they may sign their messages, and all parties know the public keys of all validators. In this work, we adopt the permissioned setting [32,8,50] where the set of validators is fixed and known to all parties. Our impossibility results also apply to the weaker proof-of-stake (in the literature also called "quasi-permissionless" [32,8]) setting where the set of validators is known but may change over time. We discuss in Sec. 6 how our protocols can be adapted to the proof-of-stake setting.

Clients. Unlike validators, the number and identities of the clients is not known to all parties, and clients do not have public keys. Thus, messages sent by clients cannot be authenticated by other parties and the adversary can impersonate an honest client (cf. 'identity theft' [48]).

Adversary. For simplicity, we assume a *static* adversary, so that at the beginning of the execution, before any randomness is drawn, the PPT adversary $\mathcal{A}$ corrupts a set of validators $\mathsf{corrupt} \subseteq \mathcal{N}$. These validators are called *adversary* and are controlled by $\mathcal{A}$ (also known as Byzantine faults). We also use *adversary validator* to denote a corrupted validator. Parties that are not adversary are called *honest*. The number of adversary validators is denoted by $f = |\mathsf{corrupt}|$. Since our model allows clients to influence the execution by sending messages, we also allow $\mathcal{A}$ to corrupt any number of clients. While a protocol's security may be conditioned on the number of adversary validators f, it must be independent of the number of adversary clients because the adversary may impersonate even

honest clients due to the lack of authentication. The adversary has access to the internal state of all adversary parties, including private keys. Modeling static corruption makes our impossibility results stronger. Protocols that we build as closed-box transformations of an existing protocol inherit the latter's security under adaptive corruption (details in Sec. 6).

Validator models. We use the sleepy model of consensus [50] to model intermittently crashed honest validators. At every round r, a subset $\mathsf{awake}_r \subseteq \mathcal{N}$ of validators are awake while the rest are *asleep*. When asleep, validators behave like temporarily crashed nodes: they do not run computation or send messages. Whenever a validator is awake, it knows the current round (*i.e.*, it wakes up with a synchronized clock). Based on the sleepiness of validators, we define two validator models:

- **Always-on validators**: All validators are always awake, $(\forall r\colon \mathsf{awake}_r = \mathcal{N})$.
- **Sleepy validators**: At the start of each round r, $\mathcal{A}$ selects awake_r. Adversary validators are always awake. Other parties do not know whether a validator is awake or asleep.

Client Models. We classify clients along two orthogonal criteria:

- **Communicating** clients may send messages to other parties, **silent** clients do not.
- **Always-on** clients are always awake while **sleepy** clients may be put to sleep by $\mathcal{A}$. When asleep, clients do not perform any computation, send messages, or output new logs.

This results in four client models shown in Fig. 2. For example, the *sleepy communicating* model means that all clients are sleepy and communicating.

Communicating clients are newly introduced in this work. In deviation from previous literature, we allow clients to send messages and other parties to act on such messages. However, due to the lack of a PKI for clients, their messages cannot be authenticated, so really the best they can do is relay messages received from validators to other clients. On the one hand, this is a more accurate depiction of blockchain implementations 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 to circumvent impossibility results [45,41], thus solving SMR under higher adversary resilience than with silent clients (Fig. 1).

Network delay models. Parties can send messages to each other and call a functionality gossip(.) to send a message to all other parties. We consider two standard models of network communication: the synchronous model and the partially (eventually) synchronous model.

- **Synchronous model:** There is a known constant Δ such that if an honest party sends a message at round r, then every honest party receives the message

by round $r + \Delta$. Within this bound, the adversary chooses when each honest party receives each message.[11]

– **Partially synchronous model:** there exists a round GST (unknown to honest parties) and a known constant Δ such that if an honest party sends a message at round r, then every honest party receives the message by round $\max\{r, \mathsf{GST}\} + \Delta$. The adversary chooses GST adaptively and decides when each honest party receives each message within these bounds.

The adversary may also inject some of its own messages, and a party does not know the time at which other parties received the messages. Note that in both models, messages are delivered to the inbox of asleep parties but they can process the messages only after awakening (like the model in [50]). In practice, equivalent behavior can be achieved by having the awakening party query online parties who reply with the 'important' past messages the awakening party then processes (*e.g.*, 'initial block download' [5]), with the weak extra assumption that at least one (potentially different each round) honest party stays awake between consecutive rounds to relay the messages. Thus, while a sleepy party still receives all messages an always-on party receives, sleepy parties are strictly less powerful because they cannot record when they received a message.

Notation. We use $A \preceq B$ to denote that sequence A is a (not necessarily strict) prefix of the sequence B. We use $A \sim B$ ('A is consistent with B') as a shorthand for $A \preceq B \vee B \preceq A$.

Definition of SMR. At the start of each round, each awake party may receive some transactions as input. At the end of every round r, each awake honest client k outputs a log (sequence of transactions) $\boldsymbol{L}_k^r$. For a client k asleep at round r, let $\boldsymbol{L}_k^r = \boldsymbol{L}_k^{r-1}$. For all clients k, $\boldsymbol{L}_k^0 = L_{\text{genesis}}$. We define the following properties for an SMR protocol:

Definition 5 (Safety). *An SMR protocol Π is* safe *iff for all rounds r, s and all honest clients k, k', $\boldsymbol{L}_k^r \sim \boldsymbol{L}_{k'}^s$.*

Definition 6 (Liveness). *An SMR protocol Π is* live *with latency u iff for all rounds r ($r \geq \mathsf{GST}$ for partial synchrony), if a transaction tx was received by an awake honest validator or communicating client before round $r - u$, then for all honest clients p awake during rounds $[r - u, r]$, $\mathsf{tx} \in \boldsymbol{L}_p^r$.*[12]

Definition 7 (Resilience). *For always-on validators, a family of SMR protocols $\Pi(n)$ achieves* safety resilience $t^{\mathrm{S}} \in [0, 1]$ *and* liveness resilience $t^{\mathrm{L}} \in [0, 1]$ *if for all n,*[13] *$\Pi(n)$ is safe with overwhelming probability over executions with $f \leq t^{\mathrm{S}} n$ and live with overwhelming probability over executions with $f \leq t^{\mathrm{L}} n$. For sleepy validators, denote the adversary fraction $\frac{f}{\min_r \mathsf{awake}_r}$ by β. Then, a*

[11] Gossip networks have been shown to maintain connectivity, and thus synchrony, even under adversary majority [13,34,35].

[12] Clients may not output new logs for a few rounds after awakening. We use a single parameter u for the maximum of such delay and the protocol's latency.

[13] The number of parties is constrained to be polynomial in the security parameter.

protocol Π achieves safety resilience $t^{\mathrm{S}} \in [0,1]$ and liveness resilience $t^{\mathrm{L}} \in [0,1]$ if Π is safe with overwhelming probability over executions with $\beta \leq t^{\mathrm{S}}$ and live with overwhelming probability over executions with $\beta \leq t^{\mathrm{L}}$.

Note that it is trivial to get protocols with $t^{\mathrm{S}} = 1, t^{\mathrm{L}} = 0$ (never output) and $t^{\mathrm{L}} = 1, t^{\mathrm{S}} = 0$ (output everything in any order), so we don't consider these edge cases going forward.

Hierarchy of Models. We group the models defined above into four categories as shown in Fig. 2. Within each category, one model is more powerful than the other. For example, communicating clients are more powerful than silent clients because communicating clients can simulate silent clients by staying silent. Thus, any protocol designed for silent clients will work equally well for communicating clients. Therefore, solving SMR for communicating clients is at least as easy as solving SMR for silent clients. This is formalized in the lemma below.

Lemma 1 ((cf. Fig. 2)). *Define the four pairs of models: validator activity models $(A_0, A_1) = (\text{🖥}^{!!!}, \text{🖥}^{zZ})$, network delay models $(B_0, B_1) = (\Delta, \not{\Delta})$, client communication models $(C_0, C_1) = (\text{🧑}_{\text{💬}}, \text{🧑}_{\not{\text{💬}}})$, client activity models $(D_0, D_1) = (\text{🧑}^{!!!}, \text{🧑}^{zZ})$. For all $i_1, j_1, k_1, l_1 \in \{0,1\}$ and $i_2 \leq i_1, j_2 \leq j_1, k_2 \leq k_1, l_2 \leq l_1$, if some SMR protocol Π achieves resilience $(t_1^{\mathrm{L}}, t_1^{\mathrm{S}})$ in the model $(A_{i_1}, B_{j_1}, C_{k_1}, D_{l_1})$ and $(t_2^{\mathrm{L}}, t_2^{\mathrm{S}})$ in the model $(A_{i_2}, B_{j_2}, C_{k_2}, D_{l_2})$, then $t_2^{\mathrm{L}} \geq t_1^{\mathrm{L}}$ and $t_2^{\mathrm{S}} \geq t_1^{\mathrm{S}}$.*

Proof. It is sufficient to prove the statement in the four cases when from (i_1, j_1, k_1, l_1) to (i_2, j_2, k_2, l_2), exactly one index is decreased and the other three remain the same. We now prove these four cases.

If Π achieves $(t^{\mathrm{L}}, t^{\mathrm{S}})$ under sleepy validators, in particular, Π is safe (live) when n validators are all awake and up to $t^{\mathrm{S}}n$ ($t^{\mathrm{S}}n$) of them are adversary. If Π is safe (live) under partial synchrony, in particular, it is safe (live) with the same number of adversary validators when the adversary sets $\mathsf{GST} = 0$ which is equivalent to synchrony. If Π is safe (live) under silent clients, it is also safe (live) when the communicating clients do not send any messages. If Π is safe (live) under sleepy clients, in particular, it is safe (live) when no client sleeps. □

The above lemma implies that any resilience pair achievable in a 'harder' model in Fig. 2 is also achievable in an 'easier' model. Conversely, any impossibility result proven for an 'easier' model also holds for a 'harder' model.

B Proofs for Synchrony with Always-On Validators

B.1 Sleepy Silent Clients, Always-On Silent Clients

B.1.1 Impossibility for Always-On Silent Clients

Proof of Thm. 1. Proof is by contradiction. Suppose there exists a protocol with safety resilience t^{S} and liveness resilience t^{L} such that $t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$. Then, there are numbers f^S and f^L such that $f^S + f^L = n$, and the protocol is safe and

live in the presence of f^S and f^L adversary validators respectively. Let P and Q be disjoint sets of $f^S = n - f^L$ and f^L validators respectively. Consider the following three worlds:

World 1: Validators in P are honest, and those in Q are adversary. There is a single client k_1. The environment inputs a single transaction tx_1 to the validators in P at time 0. The adversary validators do not communicate with those in P and ignore their messages. Towards k_1, they simulate the behavior of honest validators that have not received any transaction from the environment and that cannot communicate with those in P. Due to liveness under f^L adversary validators, k_1 outputs tx_1 (and no other transaction from the environment) as part of its log by time u.

World 2: Validators in P are honest, and those in Q are adversary. There is a single client k_2. The environment inputs a single transaction tx_2 to the validators in P at time 0. The adversary validators do not communicate with those in P and ignore their messages. Towards k_2, they simulate the behavior of honest validators that have not received any transaction from the environment and that cannot communicate with those in P. By liveness, k_2 outputs tx_2 (and no other transaction from the environment) as part of its log by time u.

World 3: World 3 is a hybrid world. Validators in Q are honest, and those in P are adversary. The environment inputs the transactions tx_1 and tx_2 to the validators in P at time 0. Validators in P simulate the execution in world 1 towards k_1 and the execution in world 2 towards k_2 via a *split-brain attack*. They do not communicate with the validators in Q and ignore their messages. Since the worlds 1 and 3 are indistinguishable in k_1's view, it outputs tx_1 (but not tx_2) as part of its log by time u. Since the worlds 2 and 3 are indistinguishable in k_2's view, it outputs tx_2 (but not tx_1) as part of its log by time u. However, this implies a safety violation in the presence of f^S adversary validators, which is a contradiction. □

B.1.2 Achievability for Sleepy Silent Clients (Liveness-Favoring)

Proof of Thm. 2. Let f be the number of adversary validators.

Liveness: Suppose $f/n \leq t^{\mathrm{L}}$, *i.e.*, $f \leq n - q$, and consider a transaction tx input to an honest validator at some round r. Now, tx gathers signatures from all honest validators by round $r + \Delta$, and all clients observe these signatures by round $r + 2\Delta$. Then, since there are q or more honest validators, all honest clients add tx to their liveness queues by round $r + 2\Delta$. Every transaction added to the liveness queue of a client at some round r' is output as part of its log by round $r' + u_{\mathrm{int}}$. Therefore, tx is output as part of all honest clients' logs by round $r + u_{\mathrm{int}} + 2\Delta$, implying that Π^q_{live} satisfies liveness with latency $u_{\mathrm{int}} + 2\Delta$ and resilience $n - q$.

Safety: Suppose $f/n \leq t^{\mathrm{S}}$, *i.e.*, $f < q$. Then, the internal protocol is safe and live with latency u_{int} as $q \leq n/2$. Any transaction tx added to the liveness queue of an honest client k at some round r must have been signed by q validators before round r, one of which is honest. Thus, tx would be input to the internal

protocol Π_{int} by round r, and by liveness, output as part of k's internal log L_{int} by round $r + u_{\text{int}}$. Since k attempts to append tx to L_{int} for the first time at round $r + u_{\text{int}}$, and tx appears as part of L_{int} by round $r + u_{\text{int}}$, tx is not added to the tip of L_{int}. By the same logic, if $f < q$, no transaction added to the liveness queue of an honest client is appended to the tip of its internal log, implying that each honest client outputs its internal log as it is. Finally, safety follows from the safety of the internal protocol. □

B.2 Sleepy Communicating Clients

B.2.1 Impossibility for Sleepy Communicating Clients

Proof of Thm. 3. Proof is by contradiction. Suppose there exists a protocol with resiliences t^{S} and t^{L} such that $t^{\text{S}}, t^{\text{L}} \geq n/2$. Then, the protocol is safe and live in the presence of $f = \lceil n/2 \rceil$ adversary validators. Let P, Q and R denote disjoint sets of $n - f$, $n - f$ and $2f - n$ validators respectively. Consider the following four worlds:

World 1: Validators in P are honest, and those in Q and R have crashed. There is a single client k_1. The environment inputs a single transaction tx_1 to the validators in P at time 0. Since $|Q \cup R| \leq f$, by liveness, k_1 outputs tx_1 (and no other transaction from the environment) as part of its log by time u.

World 2: Validators in Q are honest, and those in P and R have crashed. There is a single client k_2. The environment inputs a single transaction tx_2 to the validators in Q at time 0. Since $|P \cup R| \leq f$, by liveness, k_2 outputs tx_2 (and no other transaction from the environment) as part of its log by time u.

World 3: Validators in P are honest, and those in $Q \cup R$ are adversary. Validators in R have crashed. There are two honest clients, k_1 and k_3, and the adversary simulates a client k_2. Client k_3 joins the protocol at round u. The environment inputs a single transaction tx_1 to the validators in P at time 0.

Client k_2 and the validators in Q do not communicate with the client k_1 and the validators in P. Thus, for k_1, world 3 is indistinguishable from world 1, and it outputs tx_1 (and no other transaction from the environment) as part of its log by time u. In the meanwhile, k_2 and the validators in Q start with transaction tx_2, and emulate the execution in world 2 until round u.

Once k_3 joins the protocol at round u, k_2 and the validators in Q emulate towards k_3 the behavior of the honest validators (in Q) and the client k_2 in world 4. In other words, they pretend like honest validators and an honest client who have been shunned by the validators in $P \cup R$ and client k_1. Since $|Q \cup R| \leq f$, by liveness, k_3 outputs tx_1 as part of its log by time $2u$.

World 4: Validators in Q are honest, and those in $P \cup R$ are adversary. Validators in R have crashed. There are two honest clients, k_2 and k_3, and the adversary simulates a client k_1. Client k_3 joins the protocol at round u. The environment inputs a single transaction tx_2 to the validators in Q at time 0.

Client k_1 and the validators in P do not communicate with the client k_2 and the validators in Q. Thus, for k_2, world 4 is indistinguishable from world 2, and it outputs tx_2 (and no other transaction from the environment) as part of its log

by time u. In the meanwhile, k_1 and the validators in P start with transaction tx_1, and emulate the execution in world 1 until round u.

Once k_3 joins the protocol at round u, k_1 and the validators in P emulate towards k_3 the behavior of the honest validators (in P) and the client k_1 in world 3. In other words, they pretend like honest validators and an honest client who have been shunned by the validators in $Q \cup R$ and client k_2. Since $|P \cup R| \leq f$, by liveness, k_3 outputs tx_2 as part of its log by time $2u$.

Finally, note that worlds 3 and 4 are indistinguishable by k_3 with overwhelming probability, since the validators and clients send the same messages in both worlds. Therefore, k_3 outputs the same log, containing tx_1 and tx_2, in both worlds by time $2u$. However, this implies a safety violation either in world 3, where k_1 outputs the log $[\mathsf{tx}_1]$ by round u, or in world 4, where k_2 outputs the log $[\mathsf{tx}_2]$ by round u. This is a contradiction as the protocol must have been safe in the presence of f adversary validators. □

B.2.2 Achievability for Sleepy Communicating Clients (Safety-Favoring)

Definition 8 (Certifiable protocol). *An SMR protocol Π is certifiable if there exists a computable functionality $\mathcal{W}$ (the* certificate producer*) and a computable deterministic non-interactive function $\mathcal{C}$ (the* certificate consumer*) such that when a client p invokes $\mathcal{W}()$ at round r, it produces a certificate C such that $\mathcal{C}(C) = \boldsymbol{L}_p^r$.*

Definition 9 (Certifiable safety). *A certifiable protocol Π is certifiably safe if Π is safe, and moreover, if at any round r, the adversary outputs a certificate C such that $\mathcal{C}(C) = L$, then for all clients q, for all rounds s, $L \sim \boldsymbol{L}_q^s$. A certifiable protocol Π achieves certifiable safety resilience t^{S} if Π is certifiably safe with overwhelming probability over executions with $f \leq t^{\mathrm{S}} n$.*

A more formal pseudocode of the protocol based on Def. 4 is in Alg. 2.

Lemma 2 (Safety). *Suppose the network is synchronous and the clients are sleepy and communicating. Then, Π_{frz} has safety resilience $t^{\mathrm{S}} = 1$.*

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, $\boldsymbol{L}_p^r \not\sim \boldsymbol{L}_q^s$. For shorthand, let $L = \boldsymbol{L}_p^r$. Then, at round $r - \Delta$, client p must have seen a certificate C such that $\mathcal{C}(C) = L$. Client p also gossiped C at round $r - \Delta$, which means that before the end of round r, client q must have seen C. Thus, client q added L to its set $\mathcal{S}$ before the end of round r. However, since client q output $\boldsymbol{L}_q^s \not\sim L$ at round $s \geq r$, this is a contradiction to the freezing (Alg. 2 l. 11). □

Lemma 3 (Liveness). *If Π_{int} has certifiable safety resilience $t_{\mathrm{int}}^{\mathrm{S}}$ and liveness resilience $t_{\mathrm{int}}^{\mathrm{L}}$, then Π_{frz} has liveness resilience $\min\{t_{\mathrm{int}}^{\mathrm{L}}, t_{\mathrm{int}}^{\mathrm{S}}\}$.*

Proof. Let $u = u_{\mathrm{int}} + \Delta$ where u_{int} is the latency of Π_{int}. Let $r < r_{\mathrm{maj}}$ be any arbitrary round. Suppose that a transaction tx is received by all honest validators

Algorithm 3 SMR protocol Π^*_{live} achieving $t^{\text{L}} = 1, t^{\text{S}} < 1/2$

```
1  on INIT(𝒩, L_genesis)
2      P ← new Π_int(𝒩, L_genesis)                    ▷ instantiate a new Π_int client
3      Q ← ∅                                          ▷ liveness queue: txs seen so far
4      L ← L_genesis                                  ▷ output log of the combined protocol Π*_live
5  on transaction tx from the network at round r
6      Q.enqueue((tx, r))                             ▷ add tx to the liveness queue
7      gossip(tx)
8  on L output by P at round r
9      for (tx, r') ∈ Q such that r' ≤ r − u_int − Δ and tx ∉ L
10         L ← L||tx
11     L ← L                                          ▷ output log
```

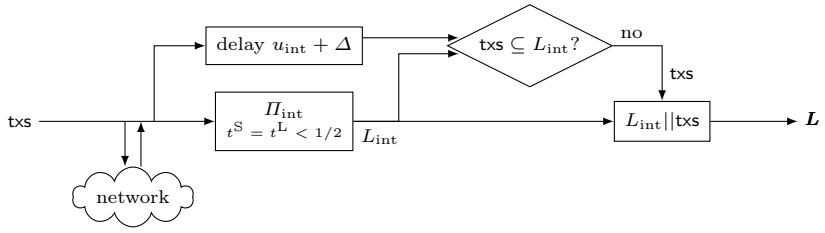


Fig. 6: A protocol that achieves $t^{\text{L}} = 1, t^{\text{S}} < 1/2$ for sleepy communicating clients. The internal protocol Π_{int} is any SMR protocol safe and live under honest majority, achieving any $t^{\text{S}} = t^{\text{L}} < 1/2$. On receiving any transaction tx, parties gossip the transaction to the network (all parties). Clients add tx to a local liveness queue. After $u_{\text{int}} + \Delta$ rounds (where u_{int} is the maximum latency of Π), if tx is not included in the log L_{int} output by the client from Π_{int}, the client appends tx to L_{int} to output the final confirmed log $\boldsymbol{L}$.

before round $r - u$. Consider an honest client p that wakes up before $r - u$. Due to liveness of Π_{int}, at round $s = r - u + u_{\text{int}}$, $\mathsf{tx} \in {L_{\text{int}}}^s_p$ (the log output by the internal protocol Π_{int}). At round s, client p runs $C \leftarrow \mathcal{W}()$ and adds $L = \mathcal{C}(C)$ to its set $\mathcal{S}$(Alg. 2 ll. 6 and 7). Recall from Def. 4 that $L = {L_{\text{int}}}^s_p$. Due to certifiable safety, the set $\mathcal{S}$ of client p, contains only logs that are consistent with L. Therefore, at round $s + \Delta = r$, $\boldsymbol{L}^r_p \succeq L \ni \mathsf{tx}$ (due to Alg. 2 ll. 11 and 12). □

Proof of Thm. 4. From Lems. 2 and 3. □

B.2.3 Achievability for Sleepy Communicating Clients (Liveness-Favoring)

Protocol pseudocode: Alg. 3, block diagram: Fig. 6.

Lemma 4. *Suppose the network is synchronous, and the clients are sleepy and communicating. Then, the protocol Π^*_{live} has liveness resilience $t^{\text{L}} = 1$.*

Proof. Consider a transaction tx received by an honest validator at round r. The validator gossips tx to all clients. All clients receive tx by round $r + \Delta$ and add it to their liveness queues. For any client k, by round $r' = r + \Delta + u_{\text{int}} + \Delta$, either

tx is in the internal log of k or k appends tx in its output log, therefore, $\mathsf{tx} \in \boldsymbol{L}_k^{r'}$. The protocol Π^*_{live} is thus live with resilience $t^{\text{L}} = 1$ and latency $u_{\text{int}} + 2\Delta$. □

Lemma 5. *Suppose the network is synchronous, and the clients are sleepy and communicating. If the internal protocol Π_{int} has resilience $(t^{\text{L}}, t^{\text{S}})$, then the protocol Π^*_{live} has safety resilience $\min\{t^{\text{L}}, t^{\text{S}}\}$.*

Proof. Suppose the number of adversary validators is $f \leq n \min\{t^{\text{L}}, t^{\text{S}}\}$. Thus, Π_{int} is safe and live. We will show that for every client k and round r, $\boldsymbol{L}_k^r = {L_{\text{int}}}_k^r$, *i.e.*, the output log is identical to the internal log. Then, Π^*_{live} is safe due to the safety of Π_{int}.

To show that $\boldsymbol{L}_k^r = {L_{\text{int}}}_k^r$, consider any transaction tx that client k adds to its liveness queue at some round r'. Since client k gossips tx, all honest validators receive tx by round $r' + \Delta$. Due to liveness of Π_{int}, $\mathsf{tx} \in {L_{\text{int}}}_k^{r'+\Delta+u_{\text{int}}}$. Therefore, client k does not append tx to $\boldsymbol{L}$ and simply drops it from the liveness queue. Since this holds for all transactions, $\boldsymbol{L}_k^r = {L_{\text{int}}}_k^r$ for all r. □

Proof of Thm. 5. From Lems. 4 and 5. □

B.3 Always-On Communicating Clients

B.3.1 Achievability for Always-On Communicating Clients We first define a variant of the Byzantine Generals problem in which clients (not validators) output values and then recap the Dolev-Strong protocol with always-on communicating clients.

Definition 10. *Let $\mathcal{V}$ be a predefined set of values and let $\perp \notin \mathcal{V}$ be a predefined default value. In the Byzantine Generals problem, a leader ℓ* BG-broadcasts *a value $v_\ell \in \mathcal{V}$ at a known start round R and each client k* BG-outputs *a value $v_k \in \mathcal{V} \cup \{\perp\}$ with the following properties:*

- *Termination: For some u, all clients output a value by round $R_\ell + u$.*
- *Agreement: For all honest clients k, k', $v_k = v_{k'}$.*
- *Validity: If ℓ is honest, then for all clients k, $v_k = v_\ell$.*

The Dolev-Strong protocol (with clients) achieving termination, agreement and validity when up to $n-1$ validators are adversary is shown in Alg. 4 (cf. [9,3]). Since we will use our Byzantine Generals protocol to build an SMR protocol over n validators, we consider the leader to be one of the validators, although, in general, the leader could be any party with a public key. In Alg. 4, $\langle m \rangle_p$ denotes messages m signed by party p and the protocol uses an instance identifier id to enable running multiple instances in the SMR protocol. As in the classic Dolev-Strong protocol [18,3], validators build a signature chain in which the leader signs its value, the second validator signs the leader's signed message, and so on (l. 9), and a signature chain is considered valid if it arrives within a timeout (ll. 8 and 13). The key differences from classic Dolev-Strong are that clients broadcast messages they receive as-is (without signing) to all parties and the timeouts are twice as long to accommodate for the round-trip delay between clients and validators ($2k\Delta$ in Alg. 4 v.s. $k\Delta$ in [18]).

Algorithm 4 Dolev–Strong protocol with clients

```
1  ▷ Each instance of the protocol is identified by (id, ℓ) where ℓ is the leader.
2  ▷ All parties know the starting round R_id for each id
3  ▷ Code for leader ℓ (who is also a validator)
4  on BG-broadcast(id, ℓ, v) at round R_id
5      Send ⟨id, ℓ, v⟩_ℓ to all parties
6  ▷ Code for validator i
7  on receiving m = ⟨⟨⟨id, ℓ, v⟩_{j_1}⟩_{j_2}...⟩_{j_k} where j_1, ..., j_k ≠ i are distinct validators and j_1 = ℓ
8      if current round ≤ R_id + 2kΔ
9          Send ⟨m⟩_i to all parties
10 ▷ Code for client
11 V_out ← ∅                                              ▷ Set of candidate output values
12 on receiving m = ⟨⟨⟨id, ℓ, v⟩_{j_1}⟩_{j_2}...⟩_{j_k} where j_1, ..., j_k are distinct validators and j_1 = ℓ
13     if current round ≤ R_id + (2k − 1)Δ
14         V_out ← V_out ∪ {v}
15         Send m to all parties
16 at the end of round R_id + (2n − 1)Δ
17     if |V_out| = 1
18         BG-output(id, ℓ, v) where V_out = {v}
19     else
20         BG-output(id, ℓ, ⊥)
```

Lemma 6. *For any $f < n$, Alg. 4 satisfies agreement and validity when f validators are adversary.*

Proof. Termination: All clients output a value after $(2n-1)\Delta$ rounds.

Validity: Suppose the leader ℓ is honest and BG-broadcasts value v_ℓ. By round Δ, all clients receive $\langle v \rangle_\ell$. Thus, the condition in l. 13 is true, all clients add v to their set of candidate output values V_{out}. Moreover, since the leader is honest and signatures are unforgeable, no party ever receives $\langle v' \rangle_\ell$ for $v' \neq v$. Therefore, $V_{\text{out}} = \{v\}$ and the client BG-outputs v at the end of round $(2n-1)\Delta$.

Agreement: Let's refer to a message of the form $\langle\langle\langle \mathsf{id}, \ell, v\rangle_{j_1}\rangle_{j_2}...\rangle_{j_k}$ where $j_1, ..., j_k$ are distinct validators and $j_1 = \ell$ as a k-signature chain on v. First, we show that if a client c adds a value v to V_{out}, then all other clients do so too. Since client c added v to V_{out}, for some $k \leq n$, it received a k-signature chain m on v by round $(2k-1)\Delta$. If at least one of the validators $j_1, ..., j_k$ who signed this message is honest, then due to l. 8, for some $k' \leq n$, this validator signed the k'-th signature in the chain by round $2(k'-1)\Delta$, so client c' receives a k'-signature chain by round $(2k'-1)\Delta$ and thus also adds v to V_{out}. If no signatory of m is honest, then $k \leq n-1$. In this case, client c sends m to all parties, so all validators receive m by round $2k\Delta$. Since the condition in l. 8 is satisfied, at least one honest validator (who has not yet signed by assumption) signs m, and so client c' receives a $(k+1)$-signature chain by round $(2k+1)\Delta \leq (2n-1)\Delta$. So, client c' also adds v to V_{out}. Agreement follows since we have established that all clients have the same set V_{out} at the end of round $(2n+1)\Delta$. □

Next, we build an SMR protocol using the Dolev-Strong protocol with clients (Alg. 4). A similar construction is already found in [9,25] but we recap it in Alg. 5 for completeness. Every $2n\Delta$ rounds (which Alg. 4 takes to terminate), each validator starts a new instance of the Dolev-Strong protocol (Alg. 4) as the leader. At the end of $2n\Delta$ rounds, clients agree on one block (possibly empty)

Algorithm 5 SMR protocol achieving any $t^{\mathrm{L}}, t^{\mathrm{S}} < 1$

```
1  R_id ← k · 2nΔ for all id = 1, 2, ...                    ▷ known to all parties
2  ▷ Code for validator i
3  at round R_id for id = 1, 2, ...
4      BG-BROADCAST(id, ℓ, B) where B is a block of transactions received so far
5  ▷ Code for client
6  L ← [ ], id ← 1
7  on BG-OUTPUT(id, ℓ, B_ℓ) for all ℓ ∈ 𝒩
8      for ℓ in enumerate(𝒩)                    ▷ all validators in a predetermined order
9          append(L, B_ℓ)          ▷ append B_ℓ to log; treat invalid B_ℓ or ⊥ as empty block
10     id ← id + 1
11     Output L
```

from each validator and add them to their logs in a predetermined ordering over the validators.

Proof of Cor. 3. Safety follows from the agreement property of the Dolev-Strong protocol with clients (Lem. 6). Assuming honest parties broadcast all transactions they receive, an honest validator will BG-broadcast a block containing all valid transactions sent to any honest party. By validity of the Dolev-Strong protocol with clients (Lem. 6), all clients will append this block to their logs. □

C Proofs for Synchrony with Sleepy Validators

C.1 Sleepy Silent Clients, Always-On Silent Clients

C.1.1 Achievability for Sleepy Silent Clients (Safety-Favoring) Recap of Goldfish (cf. [15]). Goldfish is an SMR protocol secure under synchrony and sleepy validators for any resilience $t^{\mathrm{S}} = t^{\mathrm{L}} < 1/2$. Goldfish divides time into slots of length 3Δ. Each slot t has a leader, which proposes a block, and a set of validators called voters that cast slot t votes, all selected via a verifiable random function (VRF). Each message contains a slot number and because of the VRF, all parties ignore messages from validators who were not a leader or voter for the claimed slot. Each validator and client maintains a *buffer* and a tree of blocks and votes called the *bvtree*. Upon receiving a valid message (block or vote), each validator echoes the message and adds it to its buffer, but does not add it to the bvtree immediately (*message buffering*). The crux of Goldfish is the mechanism through which messages are added to and removed from the bvtree as guided by two principles: *message buffering* and *vote expiry*.

At the beginning of each slot t, *i.e.*, time $3\Delta t$, the leader momentarily combines its buffer and bvtree without merging them permanently and runs the *GHOST-Eph* fork-choice rule on the combined bvtree using the slot $t-1$ votes. The rule outputs a canonical chain within the combined bvtree. It iteratively moves down the tree, starting at the genesis block, and, at each block B, observing the subtrees rooted at B's children. It then selects the child block with the largest number of slot $t-1$ votes by unique validators for the blocks in that tree. This process is repeated until reaching a leaf, which identifies the canonical

Algorithm 6 Modified GHOST-Eph fork-choice rule.

```
▷ The blocktree is denoted by 𝒯.
▷ Children(𝒯, B) returns the set of B's children within 𝒯.
▷ Votes(𝒯, B, t) returns the number of slot t votes by unique validators in the subtree defined by a block B within 𝒯.
function GHOST-Eph(𝒯, t)
    B ← B_0                                              ▷ Start fork-choice at genesis block
    while true
        ▷ Choose the subtree that has φ fraction of the slot t votes from unique validators.
        Finish ← true
        for B' ∈ Children(𝒯, B)
            ▷ The original Goldfish rule [15] would have selected the child with the heaviest subtree.
            if Votes(𝒯, B', t) ≥ φ · Votes(𝒯, B_0, t)
                B ← B'
                Finish ← false
        if Finish return B
```

chain.[14] Finally, the leader extends the tip of the identified canonical chain with a block, and proposes the new block along with the combined bvtree. Note that *vote expiry* is in action here, since the GHOST-Eph rule considers only the votes from the previous slot $t-1$, but not the earlier slots.

At time $3\Delta t + \Delta$, each slot t voter merges the bvtree proposed by the leader with its local bvtree. Subsequently, it votes for the tip of the canonical GHOST-Eph chain, again identified using only the slot $t-1$ votes on the merged bvtree. Note that the voter does not use its buffer at this stage, instead adding the leader's bvtree to its own before voting (vote buffering).

At time $3\Delta t+2\Delta$, each validator and client permanently adds the messages in its buffer into its bvtree. Each client finds the canonical GHOST-Eph chain, this time by running the *GHOST-Eph* fork-choice rule on its bvtree (after buffer is added) using the *slot t votes*. It outputs the sequence of blocks on this canonical chain from slots $t-\kappa$ or older as the log, where κ is a security parameter.

Our modification of Goldfish. To achieve any resilience $t^{\mathrm{S}} \leq \phi$ and $t^{\mathrm{L}} < 1-\phi$, we modify the GHOST-Eph fork-choice rule used by Goldfish as follows (Alg. 6). Within any iteration of the GHOST-Eph fork-choice, validators do not simply select a child block with the largest number of slot t votes (or slot $t-1$ votes, depending on which slot's votes are being considered). They instead inspect the number of slot t votes by unique validators on each tree rooted at the children of a block B (*i.e.*, the weight of the sub-trees) as well as the total number of slot t votes by unique validators observed so far (*i.e.*, the total weight). Then, if the fraction of the weight of one of the subtrees (rooted at a child of B) over the total weight is at least ϕ, the validator moves to that child and repeats this process. If none of the subtrees have sufficient weight or B does not have any children (*i.e.*, a leaf block), then the validator terminates the fork-choice rule at B and returns B along with its prefix. We denote the Goldfish protocol using the modified GHOST-Eph rule by $\Pi^{\phi}_{\mathrm{live}}$. Finally, to support clients, we stipulate that

[14] Note that sleepy parties can run the *GHOST-Eph* fork-choice rule because each message specifies the slot it belongs to.

upon becoming awake[15], each client maintains a buffer and bvtree, and outputs a log at rounds $3\Delta t + 2\Delta$ of every slot t via the same rule as validators, *i.e.*, after merging its buffer and bvtree.

Proof of Thm. 6. Proof is a minor modification of the proof of the Goldfish protocol [15, Appendix B]. We consider an execution of our modified Goldfish in a synchronous network with sleepy validators and sleepy, silent clients. Recall the definition of adversary fraction β from Def. 3[16].

We first note that a generalized form of [15, Lemma 1] is still true for our modified Goldfish execution as our modifications do not affect the voter selection. Therefore, w.o.p., for every slot t, the number of adversary slot t voters at round $3\Delta(t+1)+\Delta$ is less than a β fraction of the slot t voters awake at round $3\Delta t+\Delta$. Also w.o.p., all slot intervals of length κ have at least one slot t, where an honest validator is recognized as the slot t leader by all awake honest validators at round $3\Delta t$.

Furthermore, [15, Lemma 2] is also true as its proof is also not affected by our modification: If a validator v is recognized as the leader of a slot t by all awake honest validators at some round $3\Delta t + \Delta$, then, all honest slot t voters awake at round $3\Delta t + \Delta$ vote for v's proposal.

We next split [15, Lemma 3] into two parts to help with the liveness and safety proofs respectively. Proof of the lemma closely resembles that of [15, Lemma 3]:

Lemma 7. *We consider two cases for a slot t:*

1. *Suppose* all *honest slot t voters awake at round $3\Delta t+\Delta$ vote for a descendant of some block B. Then, given any $\beta \leq 1-\phi$, w.o.p.,* all *honest slot $t+1$ voters awake at round $3\Delta(t+1)+\Delta$ vote for a descendant of B.*
2. *Suppose* no *honest slot t voter awake at round $3\Delta t+\Delta$ votes for any descendant of some block B (including B itself). Then, given any $\beta < \phi$, w.o.p.,* no *honest slot $t+1$ voter awake at round $3\Delta(t+1)+\Delta$ vote for a descendant of B.*

Proof of Lem. 7. Consider an honest slot $t+1$ voter v awake at round $3\Delta(t+1)+\Delta$. Since v must have been awake at least since round $3\Delta t + 2\Delta$ due to the joining procedure of Goldfish [15, Section 3.1], its bvtree at round $3\Delta t+2\Delta$ contains all votes broadcast by honest slot t voters awake at round $3\Delta t + \Delta$. The same is true for its bvtree at round $3\Delta(t+1)+\Delta$ after merging it with the bvtree in any proposal. Moreover, when $\beta \leq 1-\phi$ the number of adversary slot t voters at round $3\Delta(t+1)+\Delta$ is at most a β fraction of the slot t voters awake at round $3\Delta t + \Delta$ ([15, Lemma 1]). Hence, in the first case, the number of slot t votes for B's descendant in v's bvtree is larger than a ϕ fraction of the total number of slot t votes by unique validators in v's bvtree at round $3\Delta(t+1)+\Delta$

[15] Through Goldfish's joining procedure [15]

[16] An adversary fraction of β as defined in Def. 3 implies a $(\beta, 3\Delta)$-compliant execution of Goldfish (cf. [15, Definition 2]), enabling us to replace the notion of compliant executions with β fraction in the proof of the modified protocol.

([15, Lemma 1]). Consequently, upon invoking the GHOST-Eph fork-choice rule at round $3\Delta(t+1)+\Delta$, v selects B's descendant over all blocks conflicting with B and moves down the tree until at least reaching a child of B. Thus, at round $3\Delta(t+1)+\Delta$, the fork choice rule returns a descendant of B, and v votes for it.

Now, in the second case, the number of slot t votes for B's descendant in v's bvtree is smaller than a $1-\phi$ fraction of the total number of slot t votes by unique validators in v's bvtree at round $3\Delta(t+1)+\Delta$. Hence, upon invoking the GHOST-Eph fork-choice rule at round $3\Delta(t+1)+\Delta$, v does not select B over blocks conflicting with B. Thus, at round $3\Delta(t+1)+\Delta$, fork choice does not return B or any of its descendants, and v does not vote for B or its descendants. □

Safety: To prove safety, we modify the proof of [15, Theorem 1]. Suppose $\beta < \phi$, and an honest validator v with proposed block B is accepted as the leader of some slot t by all awake honest validators at round $3\Delta t+\Delta$. From [15, Lemmas 1 and 2] and Lem. 7 part (ii), it follows by induction that w.o.p., for any $t' \geq t$, no honest slot t' voter awake at round $3\Delta t'+\Delta$ votes for a block that is *not* consistent with B.

By synchrony, the honest votes of slot t' reach all honest validators (and clients) awake at round $3\Delta t'+2\Delta$ by then, when they also merge the votes into their bvtrees. The number of honest slot t' voters awake at round $3\Delta t'+2\Delta$ is greater than a $1-\phi$ fraction of the total number of slot t' voters at round $3\Delta(t'+1)+2\Delta$ (by Lem. 1). Upon invoking the GHOST-Eph rule at rounds $3\Delta t'+2\Delta$, $3\Delta(t'+1)$ and $3\Delta(t'+1)+\Delta$, respectively, an awake honest validator, or client (who must have been awake since at least $3\Delta t'+2\Delta$) observes that at every iteration of the fork choice, every block that conflicts with B has less slot t' votes in its subtree (and on itself) than a ϕ fraction of the total number of slot t' votes in the bvtree. Thus, the fork choice rule returns a block that is consistent with B.

Now, let ch_1 and ch_2 denote the two chains confirmed by some clients k_1 and k_2 at slots t_1 and $t_2 \geq t_1$ respectively. Note that the slot interval $[t_1-\kappa, t_1]$ has at least one slot t, where an honest validator with proposed block B is recognized as the slot leader by all awake honest validators at round $3\Delta t+\Delta$, and, by the arguments above, no block that is not consistent with B is ever identified by any awake honest validator's or client's fork choice rule in rounds $r \geq 3\Delta t+2\Delta$. Now, as $t \geq t_1-\kappa$, but by Goldfish's confirmation rule, blocks in ch_1 are from no later than $t_1-\kappa$, ch_1 is in the prefix of B. Moreover, by the earlier argument, ch_2 is consistent with B. Therefore, ch_1 and ch_2 are consistent.

Liveness: Suppose $\beta < 1-\phi$. Then, liveness follows from [15, Theorems 1, 2 and 3], which hold given [15, Lemmas 1 and 2] and Lem. 7 part (i), the latter implying the same result as [15, Lemmas 3]. □

C.1.2 Achievability for Sleepy Silent Clients (Liveness-Favoring)

Similar to Sec. 3.1, we describe a family Π^{ϕ}_{live} of protocols, $\phi \in (0, 1/2]$, such that Π^{ϕ}_{live} is live with resilience $t^{\text{L}} \leq 1-\phi$ and safe with resilience $t^{\text{S}} < \phi$. The pro-

tocol Π^{ϕ}_{live} is very similar to its counterpart in Sec. 3.1, consisting of an *internal protocol* Π_{int} and a *liveness queue*. The internal protocol can be any SMR protocol that provides all resiliences $t^{\text{S}} < 1/2,\ t^{\text{L}} < 1/2$ under synchrony and sleepy validators (*e.g.*, [50, Section 4]). To determine whether a transactions tx should be added to the liveness queue, validators observe the number of signatures on tx and of validators that are believed to be awake; either because they signed tx, or recently announced they are awake. Validators add tx to be liveness queue if their fraction is ϕ or more.

Let u_{int} be the liveness parameter of Π_{int}. Each honest awake validator v participates in the internal protocol. At every round r that is a multiple of Δ, *i.e.*, $r = \ell\Delta$ for some $\ell \in \mathbb{Z}$, v also does the following: If it has received a transaction tx from the environment (or the other validators) for the first time within the rounds $((\ell-1)\Delta, \ell\Delta]$, it sends tx and a signature on it to all parties (including clients). It also signs and sends the number 'ℓ' as a heartbeat message.

Each client locally maintains a liveness queue and an internal log L_{int}. At every round $r = \ell\Delta$ for some $\ell \in \mathbb{Z}$, each client k calculates the tally T_{tx} of signatures observed for each transaction tx. It also calculates the number $T_{\ell-1}$ of unique validators that have either sent a signature on the number $\ell - 1$ or a signature on some transaction in the past. Then, if $T_{\mathsf{tx}}/T_{\ell-1} \geq \phi$, k adds tx to its liveness queue. To output its log at a round r, the client k appends its liveness queue to its internal log with the delay u_{int} as in Sec. 3.1.

Proof of Thm. 7. Recall the definition of β from Def. 3.

Liveness: Suppose $\beta \leq 1 - \phi$, and consider a transaction tx input to an honest validator for the first time at some round $r \in ((\ell-1)\Delta, \ell\Delta]$. At round $(\ell+1)\Delta$, tx gathers signatures from all honest validators awake at round $(\ell+1)\Delta$, and all clients observe these signatures by round $(\ell+2)\Delta$. Then, for any T_{tx} and $T_{(\ell+1)\Delta}$ in a client's view at round $(\ell+2)\Delta$, it holds that $T_{\mathsf{tx}}/T_{(\ell+1)\Delta} \geq 1-\beta \geq \phi$. Therefore, all clients awake at round $(\ell+2)\Delta$ add tx to their liveness queues. Every transaction added to the liveness queue of a client at some round r' is output as part of its log by round $r' + u_{\text{int}}$. Hence, tx is output as part of all clients' logs by round $u_{\text{int}} + (\ell+2)\Delta$, implying that Π^{ϕ}_{live} satisfies liveness with parameter $u_{\text{int}} + 3\Delta$ and resilience $1 - \phi$.

Safety: Suppose $\beta < \phi$. Then, the internal protocol is safe and live with parameter u_{int} as $\phi \leq 1/2$. Any transaction tx added to the liveness queue of a client k at some round $(\ell+1)\Delta$ must have been signed by ϕT_ℓ or more validators for the value of T_ℓ in k's view. Now, T_ℓ is the same or larger than the size of the set that contains all honest validators awake at round $\ell\Delta$ and all adversary validators whose signatures on tx were received by k. Let H, A, $\tilde{A}$ respectively denote the numbers of (i) the honest validators awake at round $\ell\Delta$, (ii) the adversary validators whose signatures on tx were received by k, and (iii) the remaining adversary validators. Then, $\phi T_\ell > \beta T_\ell \geq \beta(H+A) = \beta(H+A+\tilde{A}) - \beta\tilde{A} \geq A + (1-\beta)\tilde{A}$, since $\beta(H+A+\tilde{A}) \geq A + \tilde{A}$ by the definition of β. This implies $\phi T_\ell - A > 0$, *i.e.*, one of the signatures on tx received by k is by an honest validator. Thus, tx would be input to the internal protocol Π_{int} by round $(\ell+1)\Delta$, and by liveness, output as part of the internal log L_{int} by

round $(\ell+1)\Delta + u_{\text{int}}$. As k attempts to append tx to L_{int} for the first time at round $(\ell+1)\Delta + u_{\text{int}}$, and tx appears as part of L_{int} by round $(\ell+1)\Delta + u_{\text{int}}$, tx is not added to the tip of L_{int}. Therefore, if $\beta < \phi$, no transaction added to the liveness queue of an honest client is appended to the tip of its internal log, implying that each honest client outputs its internal log as it is. Finally, safety follows from the safety of the internal protocol. □

C.2 Sleepy Communicating Clients

C.2.1 Impossibility for Sleepy Communicating Clients

Proof of Thm. 8. Proof is by contradiction. Suppose there exists a protocol with resiliences $t^{\text{S}} = \beta$ and $t^{\text{L}} = 1 - \beta \leq t^{\text{S}}$ for some $\beta \in [1/2, 1]$. Let P and Q denote disjoint sets of βn and $(1-\beta)n$ validators. Consider the following four worlds:

World 1: Validators in P are honest and awake, and those in Q are asleep. There is a single client k_1. The environment inputs a single transaction tx_1 to the validators in P at time 0. By liveness, k_1 outputs tx_1 (and no other transaction from the environment) as part of its log by time u.

World 2: Validators in Q are honest and awake, and those in P are asleep. There is a single client k_2. The environment inputs a single transaction tx_2 to the validators in Q at time 0. By liveness, k_2 outputs tx_2 (and no other transaction from the environment) as part of its log by time u.

World 3: Validators in P are honest and awake, and those in Q are adversary. There are two honest clients, k_1 and k_3, and the adversary simulates a client k_2. Client k_3 joins the protocol at round u. The environment inputs a single transaction tx_1 to the validators in P at time 0.

Client k_2 and the validators in Q do not communicate with the client k_1 and the validators in P. Thus, for k_1, world 3 is indistinguishable from world 1, and it outputs tx_1 (and no other transaction from the environment) as part of its log by time u. In the meanwhile, k_2 and the validators in Q start with transaction tx_2, and emulate the execution in world 2 until round u.

Once k_3 joins the protocol at round u, k_2 and the validators in Q emulate towards k_3 the behavior of the honest validators (in Q) and the client k_2 in world 4. In other words, they pretend like honest validators and an honest client who have been shunned by the validators in P and client k_1. Since $|Q|/n \leq t^{\text{L}}$, by liveness, k_3 outputs tx_1 as part of its log by time $2u$.

World 4: Validators in Q are honest and awake, and those in P are adversary. There are two honest clients, k_2 and k_3, and the adversary simulates a client k_1. Client k_3 joins the protocol at round u. The environment inputs a single transaction tx_2 to the validators in Q at time 0.

Client k_1 and the validators in P do not communicate with the client k_2 and the validators in Q. Thus, for k_2, world 4 is indistinguishable from world 2, and it outputs tx_2 (and no other transaction from the environment) as part of its log by time u. In the meanwhile, k_1 and the validators in P start with transaction tx_1, and emulate the execution in world 1 until round u.

Once k_3 joins the protocol at round u, k_1 and the validators in P emulate towards k_3 the behavior of the honest validators (in P) and the client k_1 in world 3. In other words, they pretend like honest validators and an honest client who have been shunned by the validators in Q and client k_2.

Finally, note that worlds 3 and 4 are indistinguishable by k_3 with overwhelming probability, since the validators and clients send the same messages in both worlds. Therefore, k_3 outputs the same log as in world 3, which contains tx_1. Now, if the first transaction in k_3's log is tx_1, this implies a safety violation in world 4, since k_2 outputs the log $[\mathsf{tx}_2]$ by round u in world 4. This is a contradiction since the protocol must have been safe, as $|P|/n = \beta = t^{\mathrm{S}}$. On the other hand, if the first transaction in k_3's log is not tx_1, this implies a safety violation in world 3, since k_1 outputs the log $[\mathsf{tx}_1]$ by round u in world 3. This is a contradiction again, since the protocol must have been safe, as $|Q|/n = 1 - \beta \leq t^{\mathrm{S}}$. □

C.3 Always-On Communicating Clients

C.3.1 Achievability for Always-On Communicating Clients We show that the SMR protocol based on Dolev-Strong (Sec. 3.3, Alg. 5) achieves any $t^{\mathrm{L}} < 1, t^{\mathrm{S}} < 1$ even under sleepy validators. Under sleepy validators with always-on communicating clients, when a majority of the awake validators are adversary, the clients must output safe and live logs even though the validators themselves may not agree on a log (since validators are sleepy and communicating, the impossibility in Fig. 1h applies to them). Thus, the challenge is to design the validator's code to behave correctly even without knowing what happened while it was sleeping.

This challenge resolves itself due to the following observations. First, while the SMR protocol (Alg. 5) runs instances of Dolev-Strong one after the other, each instance does not depend on the previous instances. Second, within an instance, since the Dolev-Strong protocol (Alg. 4) guarantees agreement and validity when all but one validator are adversary under always-on validators (Lem. 6), it does so even when only one validator is honest and awake throughout the instance (all honest validators who sleep could be considered adversary). Moreover, we don't even require the same honest validator to be awake throughout the instance but only require that for each round during the instance, *some* honest validator is awake. Finally, since each validator (including the leader) signs only one message per instance, it may sleep after it does so without affecting the protocol's remaining execution. Thus, sleepy validators can faithfully run Alg. 4.

Proof of Thm. 10. For any $t^{\mathrm{L}}, t^{\mathrm{S}} < 1$, we know that at any given round r, there is at least one honest node awake. First, we will prove that Alg. 4 satisfies agreement and validity (Def. 10), where the definition of validity is modified to require an honest *and awake* leader. Then, using that, we will show that Alg. 5 satisfies safety and liveness.

Validity: Suppose the leader ℓ is awake and honest and BG-broadcasts value v_ℓ. By round Δ, all clients receive $\langle v \rangle_\ell$. Thus, the condition in l. 13 is true, all

clients add v to their set of candidate output values V_{out}. Moreover, since the leader is honest and signatures are unforgeable, no party ever receives $\langle v' \rangle_\ell$ for $v' \neq v$. Therefore, $V_{\text{out}} = \{v\}$ and the client BG-outputs v at the end of round $(2n+1)\Delta$.

Agreement: Let's refer to a message of the form $\langle \langle \langle v \rangle_{j_1} \rangle_{j_2} ... \rangle_{j_k}$ as a k-signature chain on v. First, we show that if a client c adds a value v to V_{out}, then all other clients do so too. Since client c added v to V_{out}, for some $k \leq n$, it received a k-signature chain m on v by round $(2k-1)\Delta$. If at least one of the validators $j_1, ..., j_k$ who signed this message is honest, then due to l. 8, for some $k' \leq n$, this validator signed the k'-th signature in the chain by round $2(k'-1)\Delta$ (when it was awake), so client c' receives a k'-signature chain by round $(2k'-1)\Delta$ and thus also adds v to V_{out}. If no signatory of m is honest, then $k \leq n-1$. In this case, client c sends m to all parties, so all validators receive m by round $2k\Delta$. Note that a validator can check the condition in l. 8 using m and knowledge of the leader, validator set, and current round, even if it has been sleeping earlier. Since the condition is satisfied, an honest awake validator (who exists and has not yet signed by assumption) signs m, and so client c' receives a $(k+1)$-signature chain by round $(2k+1)\Delta \leq (2n-1)\Delta$. So, client c' also adds v to V_{out}. Agreement follows since we have established that all clients have the same set V_{out} at the end of round $(2n+1)\Delta$.

For the SMR protocol (Alg. 5), safety again follows immediately from agreement. Liveness follows from validity since an honest awake validator will BG-broadcast a block containing all valid transactions it has received. □

C.3.2 Impossibility for Always-On Communicating Clients We conclude by showing that it is impossible to achieve safety and liveness resiliences of exactly 1 simultaneously.

Theorem 11. *In a synchronous network with always-on validators and always-on communicating clients, no protocol can achieve resiliences* $(t^{\text{L}}, t^{\text{S}})$ *such that* $t^{\text{L}} = t^{\text{S}} = 1$.

Proof. Proof is by contradiction. Suppose there exists a protocol Π with resiliences $t^{\text{S}} = t^{\text{L}} = 1$. Consider the world (called world 1), where all validators are adversary and crashed, and there are n' clients (at least two of which are honest). By assumption, the protocol satisfies safety and liveness, even though the adversary can simulate any (polynomial) number of clients.

Next, consider a world (called world 2) with a synchronous network and n' always-on validators that are connected by authenticated channels. These validators simulate n other crashed validators in their head, and run the protocol Π above, assuming the simulated validators are crashed. Even though they are connected via authenticated channels, they can run the protocol Π; since the communication among the n' clients in world 1 can be emulated by the n' validators in world 2. By the assumption above, safety and liveness are satisfied for these validators in world 2, even though the adversary can simulate $n+f$ validators for any constant $f \geq n/3$. However, this contradicts with the well-known

FLM'85 impossibility result [20], implying that the protocol cannot be safe and live in world 1. □

D Partial Synchrony

Corollary 6. *Suppose the network is partially synchronous with always-on validators. Then for any type of clients, no protocol achieves* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *such that* $2t^{\mathrm{L}} + t^{\mathrm{S}} \geq 1$.

Corollary 7. *Suppose the network is partially synchronous with always-on validators. Then for any type of clients, for all* $q \in (n/2, n]$, *HotStuff [57] with a quorum size* q *achieves all* $(t^{\mathrm{L}}, t^{\mathrm{S}})$ *with* $t^{\mathrm{L}} \leq \frac{n-q}{n}$ *and* $t^{\mathrm{S}} < \frac{2q-n}{n}$.

Corollary 8. *Suppose the network is partially synchronous with sleepy validators. Then for any* $t^{\mathrm{S}} > 0, t^{\mathrm{L}} > 0$, *no protocol can achieve* $(t^{\mathrm{L}}, t^{\mathrm{S}})$.

Cor. 6 follows from the 'split brain proof' [41, Theorem 3.1], in turn inspired by [19,6]. The impossibility is proven for the 'no client' model, which as discussed in Sec. 5, is equivalent to always-on silent clients. Due to Lem. 1, the impossibility result holds for all client types. Cor. 7 follows from [57, Theorems 2 and 4] by replacing the quorum sizes with $q \in (n/2, n]$. The protocol is proven secure for sleepy silent clients, thus for all other clients too. Other protocols Streamlet [12], Casper FFG [10], and Tendermint [7] can also be used to achieve the same result. Finally, Cor. 8 follows from the 'blockchain CAP theorem' [31,47].

E *Stubborn Nakamoto*

E.1 A Concrete Attack on the Protocol

There have been attempts at creating protocols that maintain safety against *all* adversaries ($t^{\mathrm{S}} = 1$) and liveness against 1/2 adversaries ($t^{\mathrm{L}} = 1/2$) in the setting of Bitcoin. It has been posited that communicating clients can achieve safety resilience $t^{\mathrm{S}} = 1$. The *Stubborn Nakamoto* protocol, put forth in a recent preprint [28], is akin to Alg. 2. The model is synchronous, with sleepy validators and sleepy communicating clients. The protocol [28, Def. 3] is largely identical to Alg. 2, but uses Bitcoin as its internal protocol Π_{int}. The communicating clients, upon receiving a new candidate ledger to be confirmed, in the form of a k-deep block in a longest chain, gossip it and wait 2Δ rounds before they output it. However, because the protocol aims to work in the sleepy validator setting, the *internal* protocol is not[17] *certifiable*. Concretely, the paper's "certificates" are proof-of-work blockchains starting at the genesis block and attesting to the confirmation of transactions that have been buried under k blocks. But such "certificates" do not satisfy *certifiable safety* in Def. 4. The reason is that a

[17] Certifiable safety is proven impossible in the 'unsized' sleepy validator setting such as proof-of-work [30].

block's transactions should be output in the log only if the block is k-deep *in the longest chain*, but the "certificate" only guarantees that the block is k-deep *in some chain* and does not rule out the existence of longer chains. This lack of certifiability of Π_{int} makes *Stubborn Nakamoto* insecure.

For a concrete attack, consider a majority mining adversary, and sleepy communicating clients[18]. Suppose that the adversary mining rate is very high, and consider two honest clients P_1 and P_2, initially agreeing on the genesis block. The adversary performs a *balancing attack*. She keeps mining two independent and eternally-growing chains C_1 and C_2, aiming for P_1 to output C_1 and P_2 to output C_2. Initially, the adversary mines k blocks on C_1 and another k blocks on C_2 (before the honest miners manage to mine any k-long chain). At time t_0, the adversary simultaneously sends the first k blocks of C_1 to P_1 and the first k blocks of C_2 to P_2. When P_1 receives the k blocks of C_1, he will gossip them and wait 2Δ rounds before confirming them. In the meantime, P_2 also receives the k blocks of C_2 and does the same. From that point on, the adversary will mine blocks on top of both chains and disseminate one new block of C_1 to P_1 and one new block of C_2 to P_2 every $\Delta/2$ rounds (assume she either has sufficient mining power to keep mining, or she has premined them in advance). Before P_1 has received P_2's gossiped blocks, the adversary has mined another block on top of C_1 and made it known to P_1 at time $t_0 + \Delta/2$, thereby causing P_1 to grow the longest chain in its view. When P_1 receives P_2's gossiped blocks at time $t_0 + \Delta$, the chain received from P_2 is no longer candidate for confirmation, as it is not a longest chain. Therefore, the message from P_2 to P_1 does not stop P_1 from outputting C_1. The process continues with both clients having different ever-growing longest chains, without ever halting. As a result, no matter what confirmation depth k is used, the protocol is unsafe.

Trying to patch the protocol to achieve the desired resiliences $t^{\text{L}} < 1/2, t^{\text{S}} = 1$ for sleepy communicating clients cannot work due to the impossibility shown in Thm. 8.

E.2 Impossibility for Proof-of-Work

Thm. 8 proves that with sleepy validators and sleepy communicating clients, for any $\epsilon \in (0, 1/2)$, resilience $t^{\text{L}} = \epsilon, t^{\text{S}} \geq 1 - \epsilon$ are impossible. In particular, $t^{\text{L}} > 0, t^{\text{S}} = 1$ is impossible. Our model assumed a fixed known set of validators, a number f of which are corrupted. However, Bitcoin's validator model has two key differences. First, Bitcoin uses proof-of-work and assumes that each validator has a limited hashing power [21]. Second, the number of validators, in this case, the total hash rate, is not known (to start with, we may consider it fixed as in the static difficulty Bitcoin model [21]).

On one hand, having unknown number of validators makes Bitcoin's model harder to solve SMR than in the sleepy validators model. On the other hand, the adversary's power being determined by its hashing power makes Bitcoin's model incomparable to our model. In particular, some impossibility proofs (*e.g.*,

[18] This attack works even on *always-on* communicating clients.

Thm. 3) that use a split-brain attack would not hold in Bitcoin's model because the adversary cannot simultaneously use its hashing power to simulate two different executions ("mine two chains").

However, the impossibility result in Thm. 8 holds even in Bitcoin's model because it does not use a split-brain attack. Rather, while honest miners run one execution, the adversary simulates an alternate execution without communicating with honest parties, and an adversary with $1-\epsilon$ fraction of the hashing power can simulate an alternate live execution that appears as if it was run by honest miners. The crux of the proof is that a sleepy client who awakens later in the execution cannot distinguish whether $1-\epsilon$ fraction of hashing power is adversary or ϵ fraction is adversary.

A model for Bitcoin's setting is described below. The proof of Thm. 8 follows in exactly the same way under this model too. The model is the same as that in App. A, except for the following modifications. The proof-of-work is modeled using a permitter oracle [21,32,31]. At every round, each validator can call the permitter oracle at most once[19] with a message m and the oracle responds with $\mathcal{R}(m)$ where $\mathcal{R}$ is a random oracle [32]. Following [21], each honest party is allowed unlimited "verification" queries to $\mathcal{R}$.

[19] Recall that a round is an arbitrarily small unit of time. Moreover, without loss of generality, we may assume that each validator has the same hashing power since a validator with higher hashing power may be considered as multiple validators.