The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets

Joachim Neu, Ertem Nusret Tas, and David Tse

Stanford University
{jneu,nusret,dntse}@stanford.edu

Abstract. For applications of Byzantine fault tolerant (BFT) consensus protocols where the participants are economic agents, recent works highlighted the importance of *accountability*: the ability to identify participants who provably violate the protocol. At the same time, being able to reach consensus under dynamic levels of participation is desirable for censorship resistance. We identify an *availability-accountability dilemma*: in an environment with dynamic participation, no protocol can simultaneously be accountably-safe and live. We provide a resolution to this dilemma by constructing a provably secure optimally-resilient accountability gadget to checkpoint a longest chain protocol, such that the full ledger is live under dynamic participation and the checkpointed prefix ledger is accountable. Our accountability gadget construction is black-box and can use any BFT protocol which is accountable under static participation. Using HotStuff as the black box, we implemented our construction as a protocol for the Ethereum 2.0 beacon chain, and our Internet-scale experiments with more than 4,000 nodes show that the protocol achieves the required scalability and has better latency than the current solution Gasper, which was shown insecure by recent attacks.

1 Introduction

1.1 Accountability and Dynamic Participation

Safety and liveness are the two fundamental security properties of consensus protocols. A protocol run by a distributed set of nodes is safe if the ledgers generated by the protocol are consistent across nodes and across time. It is live if all honest transactions eventually enter into the ledger. Traditionally, consensus protocols are developed for fault-tolerant distributed computing, where a set of distributed computing devices aims to emulate a reliable centralized computer. In modern decentralized applications such as cryptocurrencies, consensus nodes are no longer just disinterested computing devices but are agents acting based on economic and other rationales. To provide the proper incentives to encourage nodes to follow the protocol, it is important that they can be held accountable for their protocol-violating behavior. This point of view is advocated by Buterin and Griffith [5] in the context of their effort to add accountability (among other

Extended version: [31]. The authors contributed equally and are listed alphabetically.

things) to Ethereum's Proof-of-Work (PoW) longest chain protocol, and is also central to the design of Gasper [6], the protocol running Ethereum 2.0's Proof-of-Stake (PoS) beacon chain. In these protocols, accountability is used to incentivize proper behavior by slashing the stake of protocol-violating agents.

PoW protocols like Bitcoin [26] or Ethereum 1.0 do not assign identities to miners, and hence cannot be expected to provide accountability. Even Nakamoto-style PoS protocols such as Cardano's Ouroboros family [22,13,2] lack accountability. On the other hand, protocols that are designed to provide accountability include Polygraph [11] and Tendermint [4], and a recent comprehensive work [39] shows that accountability can be added on top of many (but not all) 'traditional' propose-and-vote-style Byzantine fault tolerant (BFT) protocols, such as HotStuff [42], PBFT [7], or Streamlet [8,29]. There is, however, another crucial difference between Nakamoto-style and propose-and-vote-style protocols. While protocols from the first group do not provide accountability, they tolerate dynamic participation, a sought after feature of public permissionless blockchains not only for censorship resistance. In Bitcoin, *e.g.*, the total hash rate varies over many orders of magnitude over the years. Yet, the blockchains remain continuously *available*, *i.e.*, live. Protocols from the second group, oppositely, provide accountability but do not tolerate dynamic participation.[1] Why is there no protocol that both supports accountability and tolerates dynamic participation?

1.2 Availability-Accountability Dilemma and Resolution via Accountability Gadgets

Our first result says that it is impossible to support accountability for *dynamically available* protocols, *i.e.*, protocols that are live under dynamic participation (*cf.* Theorem 1). We call this the *availability-accountability dilemma*.

Our second contribution is to provide a resolution to the dilemma. As no *single* ledger protocol can simultaneously be available and accountable, we design and implement an accountability gadget which, when applied to a longest chain protocol, generates a dynamically available ledger $\mathsf{LOG}_{\mathrm{da}}$ and a checkpointed prefix ledger $\mathsf{LOG}_{\mathrm{acc}}$ with provably optimal security properties.

Consider a network with a total of n permissioned nodes, and an environment where the network may partition and the nodes may go online and offline.

1. (**P1: Accountability**) The accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ can provide an accountable safety resilience of $n/3$ at all times (*i.e.*, identify that many protocol violators in case of a safety violation), and it is live after a possible partition heals and greater than $2n/3$ honest nodes come online.
2. (**P2: Dynamic Availability**) The available ledger $\mathsf{LOG}_{\mathrm{da}}$ is guaranteed to be safe after a possible network partition and live at all times, provided that fewer than 1/2 of the online nodes are adversarial.

[1] For completeness, there are also protocols which neither provide accountability nor tolerate dynamic participation, *e.g.*, Algorand [10].

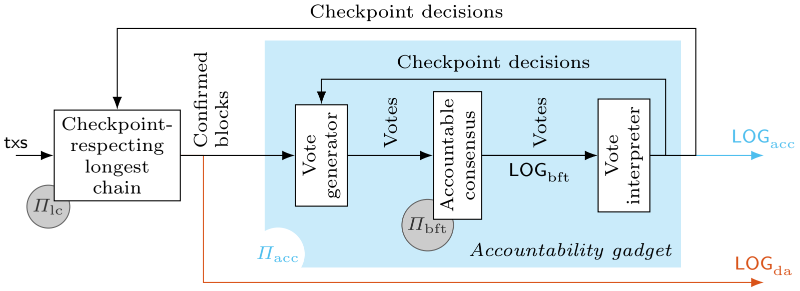

Fig. 1. We construct an accountability gadget Π_{acc} from any accountable BFT protocol Π_{bft} and apply it to a longest-chain-type protocol Π_{lc} as follows: The fork choice rule of Π_{lc} is modified to respect the latest checkpoint decision. Blocks confirmed by Π_{lc} are output as available ledger $\mathsf{LOG}_{\mathrm{da}}$. They are also the basis on which nodes generate a proposal and vote for the next checkpoint. To ensure that all nodes reach the same checkpoint decision, consensus is reached on which votes to count using Π_{bft}. Checkpoint decisions are output as accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ and fed back into the protocol to ensure consistency of future block production in Π_{lc} and future checkpoints with previous checkpoints.

Note that while the checkpointed ledger is by definition always a prefix of the full available ledger, the above result says that the checkpointed ledger will catch up with the available ledger when the network heals and a sufficient number of honest nodes come online. Users can choose individually whether to resolve the dilemma in favor of availability or accountability. For example, under exceptional circumstances, a coffee shop might rather tolerate payments reverting than stalling, while a car dealer might prefer stalling over reverting payments.

The achieved resiliences are optimal, which can be seen by comparing this result with [39, Theorem B.1] (for P1) and [34, Theorem 3] (for P2). The checkpointed ledger $\mathsf{LOG}_{\mathrm{acc}}$ cannot achieve better accountable safety resilience than $n/3$; it in fact achieves exactly that. The dynamically available ledger $\mathsf{LOG}_{\mathrm{da}}$ cannot achieve a better resilience than $1/2$; the ledger in fact achieves it. Moreover, even if the network was synchronous at all times, no protocol could have generated an accountable ledger with better resilience (Theorem 1). So we are getting partition-tolerance for free, even though accountability is the goal.

The accountability gadget construction is shown in Figure 1. It is built on top of any existing longest chain protocol modified to respect the checkpoints. That is, new blocks are proposed and the ledger of confirmed transactions is determined based on the longest chain among all the chains containing the latest checkpointed block. This gives the available full ledger $\mathsf{LOG}_{\mathrm{da}}$. Periodically, nodes vote on the next checkpoint (following a randomly selected leader's proposal). To ensure that when tallying votes all nodes base their decision for the next checkpoint on the same set of votes, any accountable BFT protocol designed for a fixed level of participation can be used (entirely as a black box) to reach consensus on the votes. The chain up to the latest checkpoint constitutes the

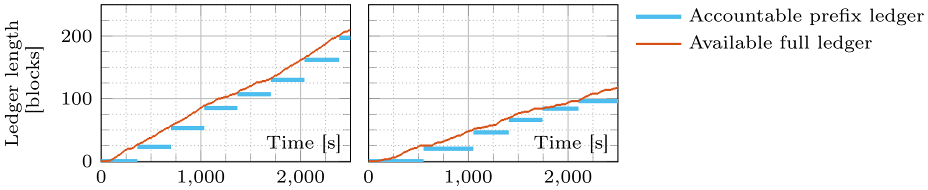

Fig. 2. Left: Ledger dynamics of a longest chain protocol outfitted with our accountability gadget based on HotStuff, measured with 4,100 nodes distributed around the world. No attack. The available full ledger grows steadily. The accountable prefix periodically catches up whenever a new block is checkpointed. **Right:** Even in the presence of a $\beta = 25\%$ adversary who mines selfishly in Π_{lc} and boycotts leader duty in Π_{bft} and Π_{acc}, $\mathsf{LOG}_{\mathrm{da}}$ grows steadily and $\mathsf{LOG}_{\mathrm{acc}}$ periodically catches up with $\mathsf{LOG}_{\mathrm{da}}$. Under attack, the growth rate of $\mathsf{LOG}_{\mathrm{da}}$ is reduced (due to selfish mining) and $\mathsf{LOG}_{\mathrm{acc}}$'s catching up is occasionally slightly delayed due to leader timeouts. (Parameters $n = 4100$, $T_{\mathrm{cp}} = 5\,\mathrm{min}$, $T_{\mathrm{to}} = 1\,\mathrm{min}$, $T_{\mathrm{hs}} = 20\,\mathrm{s}$, $k_{\mathrm{cp}} = k = 6$, all nodes online; *cf.* Sections 4.1, 5)

accountable prefix ledger $\mathsf{LOG}_{\mathrm{acc}}$. The gadget ensures that block production and confirmation in Π_{lc} and future checkpoints honor established checkpoints. When instantiated with an accountable BFT protocol that is secure under network partitions, $\mathsf{LOG}_{\mathrm{acc}}$ inherits its partition-tolerance.

Since there are many accountable BFT protocols [39], we have a lot of implementation choices. Due to its maturity and the availability of a high quality open-source implementation which we could employ practically as a black box, we decided to implement a prototype of our accountability gadget using the HotStuff protocol [42]. Taking the Ethereum 2.0's beacon chain as a target application and matching its key performance characteristics such as latency and block size, we performed Internet-scale experiments to demonstrate that our solution can meet the target specification with over 4,000 participants (see Figure 2(l)). In particular, for the chosen parameterization and even before taking reduction measures, the peak bandwidth required for a node to participate does not exceed 1.5 MB/s (with a long-term average of 78 KB/s) and hence is feasible even for many consumer-grade Internet connections. At the same time, our prototype provides 5× better average latency of $\mathsf{LOG}_{\mathrm{acc}}$ compared to the instantiation of Gasper currently used for Ethereum 2's beacon chain.

1.3 Related Work

Accountability Accountability in distributed protocols has been studied in earlier works. [20] designed a system, PeerReview, which detects faults. [21] classifies faults into different types and studies their detectability. Casper [5] focuses on accountability and fault detection when there is violation of safety, and led to the notion of accountable safety resilience we use in this work. Polygraph [11] is a partially synchronous BFT protocol which is secure when there are less than $n/3$ adversarial nodes, and when there is a safety violation, at least $n/3$

Table 1. Accountability gadgets provide security, accountability, and predictable validity, which are not found conjoint in any one of the previous works [6,29,32,37].

	Gasper [6]	Checkp. LC [37]	Snap&Chat [32,29]	Acc. gadgets (This work)
Provable security	✘	✓	✓	✓
Accountability	✓	✘	✓	✓
Predictable validity	✓	✓	✘	✓

nodes can be held accountable. [36] builds upon [11] to create a blockchain which can exclude Byzantine nodes that were found to have violated the protocol.

Many of these previous works focus on studying the accountability of *specific* protocols and think of accountability as an add-on feature in addition to the basic security properties of the protocol. [39] follows this spirit but broadens the investigation to formulate a framework to study the accountability of many existing BFT protocols. More specifically, their framework augments the traditional resilience metric with accountable safety resilience (which they call forensic support). The present work is more in the spirit of [5] where accountability is a central design goal, not just an add-on feature. To formalize this spirit, we split traditional resilience into safety and liveness resiliences, upgrade safety resilience to accountable safety resilience, and formulate accountable security as a tradeoff between liveness resilience and accountable safety resilience. Further, we broaden the study to the important dynamic participation environment, where we discovered the availability-accountability dilemma (Theorem 1). While at its heart the impossibility result Theorem B.1 of [39] is really about the tradeoff between liveness and accountable safety resiliences, although not stated as such, and it is indeed applicable very generally, when applied to the dynamic participation setting it would give a loose result and would not have been able to demonstrate the availability-accountability dilemma.

Availability-Finality Dilemma and Finality Gadgets The *availability-finality dilemma* [19,23,32] states that no protocol can provide both finality, *i.e.*, safety under network partitions, and availability, *i.e.*, liveness under dynamic participation. The *availability-accountability dilemma* states that no protocol can provide both accountable safety and liveness under dynamic participation. Although they are different, it turns out that some, but not all, protocols that resolve the availability-finality dilemma can be used to resolve the availability-accountability dilemma. Casper [5] and Gasper [6] pioneered resolution of the dilemmata but lacked a specification of the desired security properties and suffered from attacks [27,28,30,32,38,33]. Specifically, Gasper is insecure [28,30,32,38] (Table 1). The first provably secure resolution of the availability-finality dilemma is the class of snap-and-chat protocols [32], which combines a longest chain protocol with a partially synchronous BFT protocol in a black box manner to provide finality. If the partially synchronous BFT protocol is accountable, it is not too difficult to show [29] that the resulting snap-and-chat protocol would also provide a resolution to the availability-accountability dilemma. On the other hand, checkpointed longest chain [37], another resolution of the availability-finality dilemma, is not accountable (Table 1), as shown in Appendix G.

A strength of snap-and-chat protocols is its black box nature which gives it flexibility to provide additional features. A drawback is that the protocol may reorder the blocks from the longest chain protocol to form the final ledger [29]. This means that when a proposer proposes a block on the longest chain, it cannot predict the ledger state and check the validity of transactions by just looking at the earlier blocks in the longest chain. This lack of *predictable validity* (Table 1) opens the protocol up to spamming and prohibits the use of standard techniques for sharding and light client support. Checkpointed longest chain builds upon a line of work called finality gadgets [5,15,40,6] and overcomes this limitation of snap-and-chat protocols because the longest chain protocol is modified to respect the checkpoints so that the order of blocks can be preserved. However, checkpointed longest chain's finality gadget is not black box, but specifically uses Algorand BA [9], which is not accountable [39]. It is not readily apparent if and how Algorand BA could be replaced with any accountable BFT protocol.

The accountability gadget we design combines structural elements from snap-and-chat protocols and from the checkpointed longest chain to uniquely achieve the best of both worlds. It builds on the checkpointed longest chain and earlier (not provably secure) finality gadgets in that it complements a longest chain protocol with a checkpointing mechanism and thus achieves predictable validity. Like snap-and-chat protocols, it allows the use of any BFT protocol as a black box for checkpointing, retaining simplicity and flexibility and, when an accountable BFT protocol like HotStuff is used, the checkpointed ledger is accountable. Our accountability gadget provides security, accountability, and predictable validity (Table 1), which are not found conjoint in any one of the prior works.

1.4 Outline

We introduce in Section 2 the notation and model for the proof of the availability-accountability dilemma in Section 3 and the construction and security proof of accountability gadgets in Section 4. Finally, we discuss details of a prototype implementation and experimental performance results in Section 5.

2 Model

In the client-server model of state machine replication (SMR), *nodes* take inputs called *transactions* and enable clients to agree on a single sequence of transactions, called the *ledger* and denoted by LOG, that produced the state evolution. For this purpose, nodes exchange messages, *e.g.*, blocks or votes, and each node i records its view of the protocol by time t in an execution transcript T_i^t. To obtain the ledger at time t, clients query the nodes running the protocol. When a node i is queried at time t, it produces *evidence* w_i^t by applying an *evidence generation function* $\mathcal{W}$ to its current transcript: $\mathsf{w}_i^t \triangleq \mathcal{W}(\mathsf{T}_i^t)$. Upon collecting evidences from some subset S of the nodes, each client applies the *confirmation rule* $\mathcal{C}$ to this set of evidences to obtain the ledger: $\mathsf{LOG} \triangleq \mathcal{C}(\{\mathsf{w}_i^t\}_{i\in S})$. Protocols typically require to query a subset S containing at least one honest node.

Environment and Adversary: We assume that transactions are input to nodes by the environment $\mathcal{Z}$. There exists a public-key infrastructure and each of the n nodes is equipped with a unique cryptographic identity. A random oracle serves as a common source of randomness. Time is slotted and the nodes have synchronized local clocks. *Corruption:* Adversary $\mathcal{A}$ is a probabilistic poly-time algorithm. Before the protocol execution starts, $\mathcal{A}$ gets to corrupt (up to) f nodes, then called *adversarial* nodes. Adversarial nodes surrender their internal state to the adversary and can deviate from the protocol arbitrarily (Byzantine faults) under the adversary's control. The remaining $(n-f)$ nodes are called *honest* and follow the protocol as specified. *Networking:* Nodes can send each other messages. Before a *global stabilization time* GST, $\mathcal{A}$ can delay network messages arbitrarily. After GST, $\mathcal{A}$ is required to deliver all messages sent between honest nodes within a known upper bound of Δ slots. GST is chosen by $\mathcal{A}$, unknown to the honest nodes, and can be a causal function of the randomness in the protocol. *Sleeping:* To model dynamic participation, we adopt the concept of *sleepiness* [35]. Before a *global awake time*[2] GAT, $\mathcal{A}$ chooses, for every time slot and honest node, whether it is *awake* (*i.e.*, online) or *asleep* (*i.e.*, offline). After GAT, all honest nodes are awake. An awake honest node executes the protocol faithfully. An asleep honest node does not execute the protocol, and messages that would have arrived in that slot are queued and delivered in the first slot in which the node is awake again. Adversarial nodes are always awake. We define β as the maximum fraction of adversarial nodes among awake nodes throughout the execution of the protocol. GAT, just like GST, is chosen by the adversary, unknown to the honest nodes and can be a causal function of the randomness. But, while GST needs to happen eventually ($\mathsf{GST} < \infty$), GAT may be infinite.

Given above definition of a partially synchronous network with dynamic participation $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$, we model a synchronous network $(\mathcal{A}_{\mathrm{s}}, \mathcal{Z}_{\mathrm{s}})$, a partially synchronous network $(\mathcal{A}_{\mathrm{p}}, \mathcal{Z}_{\mathrm{p}})$, and a synchronous network with dynamic participation $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$ as special cases with $\mathsf{GST} = \mathsf{GAT} = 0$, $\mathsf{GAT} = 0$, and $\mathsf{GST} = 0$, respectively. Subsequently, we specify for every theorem under which of the above four $(\mathcal{A}_{\ldots}, \mathcal{Z}_{\ldots})$ it holds. Examples of Nakamoto-style and propose-and-vote-style BFT protocols framed in the above model are given in Appendix H.

Safety and Liveness Resiliences: Safety and liveness are defined as the traditional security properties of SMR protocols:

[2] Node operators are rewarded and incur little expenses for protocol participation. Thus, one naturally expects frequent periods of (near) full participation. GAT models the time when participation stabilizes, analogous to the GST of network delays.

Definition 1. *Let T_{confirm} be a polynomial function of the security parameter σ of an SMR protocol Π. We say that Π with a confirmation rule $\mathcal{C}$ is* secure *and has transaction confirmation time T_{confirm} if ledgers output by $\mathcal{C}$ satisfy:*

- ***Safety:*** *For any time slots t, t' and sets of nodes S, S' satisfying the requirements stipulated by the protocol, either $\mathsf{LOG} \triangleq \mathcal{C}(\{\mathsf{w}_i^t\}_{i \in S})$ is a prefix of $\mathsf{LOG}' \triangleq \mathcal{C}(\{\mathsf{w}_i^{t'}\}_{i \in S'})$ or vice versa.*
- ***Liveness:*** *If $\mathcal{Z}$ inputs a transaction to an awake honest node at some time t, then, for any time slot $t' \geq \max(t, \mathsf{GST}, \mathsf{GAT}) + T_{\text{confirm}}$ and any set of nodes S satisfying the requirements stipulated by the protocol, the transaction is included in $\mathsf{LOG} \triangleq \mathcal{C}(\{\mathsf{w}_i^{t'}\}_{i \in S})$.*

Definition 2. *For static (dynamic) participation,* safety resilience *of a protocol is the maximum number f of adversarial nodes (maximum fraction β of adversarial nodes among awake nodes) such that the protocol satisfies safety. Such a protocol provides* f-safety *(*β-safety*).*

Definition 3. *For static (dynamic) participation,* liveness resilience *of a protocol is the maximum number f of adversarial nodes (maximum fraction β of adversarial nodes among awake nodes) such that the protocol satisfies liveness. Such a protocol provides* f-liveness *(*β-liveness*).*

Accountable Safety Resilience: To formalize the concept of accountable safety resilience, we define an *adjudication function $\mathcal{J}$*, similar to the forensic protocol defined in [39], as follows:

Definition 4. *An adjudication function $\mathcal{J}$ takes as input two sets of evidences W and W' with conflicting ledgers $\mathsf{LOG} \triangleq \mathcal{C}(W)$ and $\mathsf{LOG}' \triangleq \mathcal{C}(W')$, and outputs a set of nodes that have provably violated the protocol rules.*

So, $\mathcal{J}$ never outputs an honest node. When the clients observe a safety violation, *i.e.*, at least two sets of evidences W and W' such that $\mathsf{LOG} \triangleq \mathcal{C}(W)$ and $\mathsf{LOG}' \triangleq \mathcal{C}(W')$ conflict with each other, they call $\mathcal{J}$ on these evidences to identify nodes that have violated the protocol. Note that $\mathsf{LOG} \triangleq \mathcal{C}(\{\mathsf{w}_i^t\}_{i \in S})$ may satisfy safety/liveness only if the evidences come from a set S of nodes that satisfies some assumptions stipulated by the protocol, *e.g.*, that S contains one honest node. On the other hand, $\mathcal{J}$ should only output nodes that have undoubtedly violated protocol, without the verdict being conditional on any presumption.

Accountable safety resilience builds on the concept of *α-accountable-safety* first introduced in [5]:

Definition 5. *For static (dynamic) participation,* accountable safety resilience *of a protocol is the minimum number f of nodes (minimum fraction β of nodes among awake nodes) output by $\mathcal{J}$ in the event of a safety violation. Such a protocol provides* f-accountable-safety *(*β-accountable-safety*).*

Note that β-accountable-safety implies β-safety of the protocol (and the same for f) since $\mathcal{J}$ outputs only adversarial nodes.

3 The Availability-Accountability Dilemma

We observe that the strictest tradeoff between the liveness and accountable safety resilience occurs for dynamically available protocols under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$, a result which was named the availability-accountability dilemma in Section 1.2:

Theorem 1. *No SMR protocol provides both β_{a}-accountable-safety and β_{l}-liveness for any $\beta_{\mathrm{a}}, \beta_{\mathrm{l}} > 0$ under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$.*

Theorem 1 states that under dynamic participation it is impossible for an SMR protocol to provide both positive accountable safety resilience and positive liveness resilience. In light of this result, protocol designers are compelled to choose between protocols that maintain liveness under fluctuating participation, and protocols that can enforce the desired incentive mechanisms highlighted in Section 1.1 via accountability. Since both of the above features are desirable properties for Internet-scale consensus protocols, the availability-accountability dilemma presents a serious obstacle in the effort to obtain an incentive-compatible and robustly live protocol for applications such as cryptocurrencies.

To build intuition for the proof of Theorem 1, let us consider a permissioned longest chain protocol under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$ where half of nodes are adversarial. Adversarial nodes avoid all communication with honest nodes and build a private chain that conflicts with the chain built collectively by the honest nodes. Such diverging chains mean the possibility of an (ostensible) safety violation. Think of an honest client towards whom adversarial nodes pretend to be asleep and who confirms a ledger based solely on the longest chain provided by the honest evidences; and a co-conspirator of the adversary who pretends to not have received any evidences from honest nodes and to have confirmed a ledger based solely on the longest chain provided by the adversarial evidences. Indeed, both would obtain non-empty ledgers, because the longest chain is dynamically available, but these two ledgers would conflict. Yet, based on the two sets of evidences, the judge $\mathcal{J}$ can neither distinguish who is honest client and who is co-conspirator, nor tell which nodes are honest or adversarial. So none of the adversarial nodes can be held accountable (without risking to falsely convict an honest node).

Formal proof of Theorem 1 (Appendix A) relies on the fact that in a dynamically available protocol, adversarial nodes, by private execution, can always create a set of evidences that yields a conflicting ledger through the confirmation rule $\mathcal{C}$. This is because dynamically available protocols cannot set a lower bound on the number of evidences eligible to generate a non-empty ledger through $\mathcal{C}$, and thus are forced to output ledgers for evidences from any number of nodes.

Theorem 1 is also related to a contemporaneous result [24] which shows that dynamically available protocols cannot produce certificates of confirmation, where such a certificate guarantees that there cannot be a conflicting confirmation so long as stipulated constraints on the adversary hold.

Algorithm 1 Checkpoint vote generator (helper functions: see Appendix E)

```
1:  $\mathrm{lastCp}, \mathrm{props} \leftarrow \bot, \{c : \bot \mid c = 0, 1, \ldots\}$    ▷ Last checkpoint, proposals
2:  for $\mathrm{currIter} \leftarrow 0, 1, \ldots$    ▷ Loop over checkpoint iterations
3:      if $\mathrm{lastCp} \neq \bot$
4:          while waiting $T_{\mathrm{cp}}$ time    ▷ Wait $T_{\mathrm{cp}}$ time after new checkpoint decision
5:              PerformBookkeeping
6:      if $\mathrm{CpLeaderOfIter}(\mathrm{currIter}) = \mathrm{myself}$    ▷ Broadcast proposal if leader of current iteration
7:          Broadcast($\langle \mathsf{propose}, \mathrm{currIter}, \textsc{GetCurrProposalTip}() \rangle_{\mathrm{myself}}$)
8:      while waiting $T_{\mathrm{to}}$ time    ▷ Wait $T_{\mathrm{to}}$ for timeout of checkpoint iteration
9:          PerformBookkeeping
10:         on $\mathrm{props}[\mathrm{currIter}] \neq \bot$, but at most once    ▷ Act on the first proposal received from
            authorized leader before end of $T_{\mathrm{cp}}$-wait and $T_{\mathrm{to}}$-timeout
11:             if IsValidProposal($\mathrm{props}[\mathrm{currIter}]$)    ▷ Valid proposal is consistent with current
                checkpoint-respecting LC
12:                 SubmitVote($\langle \mathsf{accept}, \mathrm{currIter}, \mathrm{props}[\mathrm{currIter}] \rangle_{\mathrm{myself}}$)
13:             else
14:                 SubmitVote($\langle \mathsf{reject}, \mathrm{currIter} \rangle_{\mathrm{myself}}$)    ▷ Reject invalid proposal
15:     SubmitVote($\langle \mathsf{reject}, \mathrm{currIter} \rangle_{\mathrm{myself}}$)    ▷ Reject due to timeout
16:     wait on $\mathrm{Checkpoint}(c, b)$ from checkpoint vote interpreter (Alg. 2) with $c = \mathrm{currIter}$
17:     $\mathrm{lastCp} \leftarrow b$    ▷ Keep track of checkpoint decision
18: macro PerformBookkeeping
19:     on receiving $\mathrm{Checkpoint}(c, b)$ from checkpoint vote interpreter (Alg. 2) with $c = \mathrm{currIter}$
20:         goto 17    ▷ Jump to conclusion of current iteration
21:     on receiving $\mathrm{Proposal}(c, b)$ from checkpoint leader of iteration $c$ with $\mathrm{props}[c] = \bot$
22:         $\mathrm{props}[c] \leftarrow b$    ▷ Keep track of first proposal from authorized leader per iteration $c$
```

4 Accountability Gadgets

In this section, we give a detailed description of the accountability gadget introduced in Section 1.2. For ease of exposition, we construct it from an accountable BFT protocol Π_{bft} with accountable safety and liveness resilience of $\lfloor n/3 \rfloor$.

Like the checkpointed longest chain [37], accountability gadgets output a prefix ledger safe under partial synchrony along with a full ledger live under dynamic participation. For this purpose, both protocols are deployed as overlays on top of a dynamically available longest chain protocol and periodically checkpoint its output to protect against reversals under network partition. Accountability gadgets can be instantiated from any partially synchronous BFT SMR protocol, which is used as a black box for checkpointing. If the selected protocol provides accountability, then adversarial nodes can be held to account should there ever be a reversal of a checkpoint. In contrast, the checkpointed longest chain is interwoven with a variant of a particular protocol, Algorand BA [9], which does not provide accountability [39] (*cf.* Appendix G). Furthermore, it is not readily apparent how to use another protocol instead. As a result, the checkpointed longest chain cannot provide a resolution to the availability-accountability dilemma, whereas accountability gadgets can.

4.1 Protocol Description

Accountability gadgets, denoted by Π_{acc}, can be used in conjunction with any dynamically available longest chain (LC) protocol Π_{lc} such as Nakamoto's PoW LC protocol [26], Sleepy [35], Ouroboros [22,13,2] and Chia [12] (Fig. 1). Subsequently, we focus on permissioned/PoS LC protocols. PoW and Proof-of-Space

Algorithm 2 Checkpoint vote interpreter (helper functions: see Appendix E)

```
for $\mathrm{currIter} \leftarrow 0, 1, \ldots$
    $\mathrm{currVotes} \leftarrow \{(\mathrm{pk}, \perp) \mid \mathrm{pk} \in \mathrm{committee}\}$    ▷ Latest vote of each node
    while true    ▷ Go through votes as ordered by $\Pi_{\mathrm{bft}}$
        $\mathrm{vote} \leftarrow \textsc{GetNextVerifiedVoteFromBft}()$    ▷ Verify signature
        if $\mathrm{vote} = \langle \mathsf{accept}, c, b \rangle_{\mathrm{pk}}$ with $c = \mathrm{currIter}$
            $\mathrm{currVotes}[\mathrm{pk}] \leftarrow \mathrm{Accept}(b)$    ▷ Count accept vote for block b
        else if $\mathrm{vote} = \langle \mathsf{reject}, c \rangle_{\mathrm{pk}}$ with $c = \mathrm{currIter}$
            $\mathrm{currVotes}[\mathrm{pk}] \leftarrow \mathrm{Reject}$    ▷ Count reject vote
        if $\exists b : |\{\mathrm{pk} \mid \mathrm{currVotes}[\mathrm{pk}] = \mathrm{Accept}(b)\}| \geq 2n/3$
            $\textsc{OutputCp}(\mathrm{Checkpoint}(\mathrm{currIter}, b))$    ▷ New checkpoint decision
            break
        else if $|\{\mathrm{pk} \mid \mathrm{currVotes}[\mathrm{pk}] = \mathrm{Reject}\}| \geq n/3$
            $\textsc{OutputCp}(\mathrm{Checkpoint}(\mathrm{currIter}, \perp))$    ▷ Abort current iteration
            break
```

are discussed in Appendix F. The protocol Π_{lc} then follows a modified chain selection rule where honest nodes build on the tip of the LC that contains all of the *checkpoints* they have observed.[3] We call such chains *checkpoint-respecting LCs*. At each time slot t, each honest node i outputs the k-deep prefix of the checkpoint-respecting LC (or the prefix of the latest checkpoint, whichever is longer) in its view as $\mathsf{LOG}_{\mathrm{da},i}^{t}$.

The accountability gadget Π_{acc} has three main components as shown on Fig. 1: a checkpoint vote generator (Alg. 1) issues checkpoint proposals and votes, an accountable SMR protocol Π_{bft} is used to reach consensus on which votes to count for the checkpoint decision, and a checkpoint vote interpreter (Alg. 2) outputs checkpoint decisions computed deterministically from the checkpoint votes sequenced by Π_{bft}. The protocol Π_{bft} can be instantiated with any accountable BFT protocol, *e.g.*, Streamlet [8], LibraBFT [25], or HotStuff [42]. It is used as a black box ordering service within Π_{acc} and is assumed to have confirmation time T_{confirm}. We denote the ledger output by Π_{bft} as $\mathsf{LOG}_{\mathrm{bft}}$, and emphasize that it is internal to Π_{acc}. Checkpoint vote generator and interpreter are run locally by each node and interact with Π_{bft} and $\mathsf{LOG}_{\mathrm{bft}}$. Hence, when we refer to $\mathsf{LOG}_{\mathrm{bft}}$ in the following, we mean the ledger in the view of a specific node.

The accountability gadget Π_{acc} proceeds in *checkpoint iterations* denoted by c, each of which attempts to checkpoint a block in Π_{lc}. The checkpoint vote generator produces requests which can be of three forms: $\langle \mathsf{propose}, c, b \rangle_i$ proposes block b for checkpointing in iteration c, $\langle \mathsf{accept}, c, b \rangle_i$ votes in favor of block b in iteration c, $\langle \mathsf{reject}, c \rangle_i$ votes to reject iteration c. Here, $\langle \ldots \rangle_i$ denotes a message signed by node i. Each iteration c has a publicly verifiable and unique random leader $\mathrm{L}^{(c)}$. The leader obtains the k_{cp}-deep block b on its checkpoint-respecting LC and broadcasts it to all other nodes as the checkpoint proposal for c (Alg. 1, l. 7). Nodes receive checkpoint proposals (signed by the legitimate leader $\mathrm{L}^{(c)}$) from the network, and order them with respect to their checkpoint iteration

[3] There are no conflicting checkpoints unless a safety violation has already occurred. Upon detecting a safety violation, honest nodes stop participating in the protocol. Punishment of parties identified by the accountability mechanism as malicious and system recovery are handled by mechanisms external to the protocol.

(Alg. 1, l. 21). A proposal is *valid* in view of node i if the proposed block is within i's checkpoint-respecting LC and extends all previous checkpoints observed by i. During an iteration c, each node i checks if the proposal received for c is valid (Alg. 1, l. 11). If it has received a valid proposal with block b, it votes $\langle \mathsf{accept}, c, b\rangle_i$ (Alg. 1, l. 12). Otherwise, if i does not receive any valid proposal for a timeout period T_{to}, i votes $\langle \mathsf{reject}, c\rangle_i$ (Alg. 1, l. 14, 15). Votes are input as payload to Π_{bft}, which sequences them into ledger $\mathsf{LOG}_{\mathrm{bft}}$. Thus, nodes reach consensus on which votes to count for checkpoint decision of the given iteration.

The checkpoint vote interpreter (Alg. 2) processes the sequence of votes in $\mathsf{LOG}_{\mathrm{bft}}$ to produce checkpoint decisions. Each node processes verified votes (*i.e.*, with valid signature) in the order they appear on $\mathsf{LOG}_{\mathrm{bft}}$ (Alg. 2, l. 4). Upon observing $2n/3$ unique $\langle \mathsf{accept}, c, b\rangle_i$ votes for a block b and the current iteration c, each node outputs b as the *checkpoint* for c (Alg. 2, l. 10). The checkpointed blocks output over time, together with their respective prefixes, constitute $\mathsf{LOG}^t_{\mathrm{acc},i}$. Furthermore, checkpoint decisions are fed back to Π_{lc} and the checkpoint vote generator to ensure consistency of future block production in Π_{lc} and of checkpoint proposals with prior checkpoints. Oppositely, upon observing $n/3$ unique $\langle \mathsf{reject}, c\rangle_i$ votes for the current iteration c, each node outputs $\perp$ as the checkpoint decision for c (Alg. 2, l. 13) to signal that c was aborted with no new checkpointed block. This happens if honest nodes reject because they have not seen progress for too long. Once a node outputs a decision for current iteration c, the checkpoint vote interpreter proceeds to $c+1$; thus, only a single decision is output per iteration.

Upon receiving a new checkpoint for the current iteration c, nodes leave c of the checkpoint vote generator and enter $c+1$ (Alg. 1, l. 20). If the checkpoint decision was for $b \neq \perp$, nodes wait for T_{cp} time (*checkpoint interval*) before considering checkpoint proposals for $c+1$. As will become clear in the analysis, the checkpoint interval is crucial to ensure that Π_{lc}'s chain dynamics are 'not disturbed too much' by accommodating and respecting checkpoints. Note that throughout the execution there is only a single instantiation Π_{bft}, since the votes for different checkpoint iterations can still be ordered into a single sequence.

4.2 Security Properties

In this section, we formalize and prove the security properties **P1** and **P2** of Section 1.2 for accountability gadgets based on *permissioned* LC protocols [35,13,22,2]. (For an extension of the security analysis to Proof-of-Work and Proof-of-Space LC protocols, see Appendix F.)

For the worst case, we first fix $f = \lceil n/3 \rceil - 1$ and consider an accountability gadget Π_{acc} instantiated with a partially synchronous BFT protocol Π_{bft} that provides $(n-2f)$-accountable-safety at all times, and f-liveness under partial synchrony after the network partition heals and sufficiently many honest nodes are awake. (An example Π_{bft} is HotStuff [42] with a quorum size $(n-f)$.)

Let λ and σ denote the security parameters associated with the employed cryptographic primitives and the LC protocol Π_{lc}, respectively. Then, the secu-

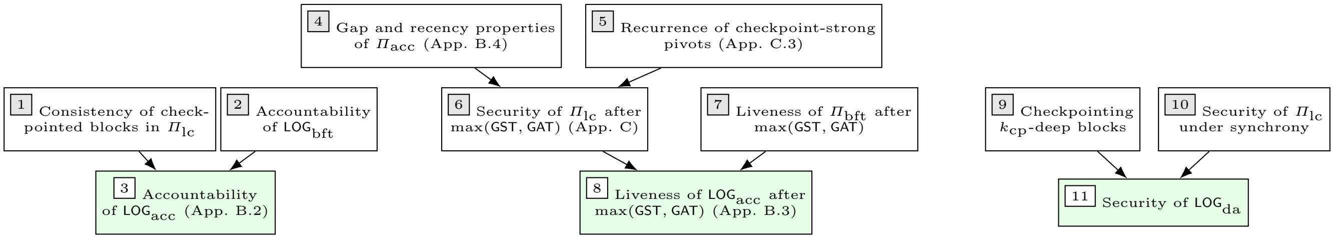
4 Gap and recency properties of Π_{acc} (App. B.4)
5 Recurrence of checkpoint-strong pivots (App. C.3)
1 Consistency of checkpointed blocks in Π_{lc}
2 Accountability of $\mathsf{LOG}_{\text{bft}}$
6 Security of Π_{lc} after $\max(\mathsf{GST}, \mathsf{GAT})$ (App. C)
7 Liveness of Π_{bft} after $\max(\mathsf{GST}, \mathsf{GAT})$
9 Checkpointing k_{cp}-deep blocks
10 Security of Π_{lc} under synchrony
3 Accountability of $\mathsf{LOG}_{\text{acc}}$ (App. B.2)
8 Liveness of $\mathsf{LOG}_{\text{acc}}$ after $\max(\mathsf{GST}, \mathsf{GAT})$ (App. B.3)
11 Security of LOG_{da}

Fig. 3. Dependency of the security properties of $\mathsf{LOG}_{\text{acc}}$ and LOG_{da} on the properties of Π_{acc}, Π_{lc} and Π_{bft}.

rity properties of $\mathsf{LOG}_{\text{acc}}$ and LOG_{da} output by the accountability gadget Π_{acc} and the LC protocol Π_{lc} (modified to be checkpoint-respecting) are:

Theorem 2. *For any λ, σ, and $T_{\text{confirm}}, k, k_{\text{cp}}$ linear in σ:*

1. ***(P1: Accountability)*** *Under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$, the accountable ledger $\mathsf{LOG}_{\text{acc}}$ provides $(n-2f)$-accountable-safety at all times (except with probability $\operatorname{negl}(\lambda)$), and there exists a constant $\mathbf{C}$ such that $\mathsf{LOG}_{\text{acc}}$ provides f-liveness with confirmation time T_{confirm} after $\mathbf{C}\max(\mathsf{GST}, \mathsf{GAT})$ (except with probability $\operatorname{negl}(\sigma)$).*
2. ***(P2: Dynamic Availability)*** *Under $(\mathcal{A}_{\text{da}}, \mathcal{Z}_{\text{da}})$, the available ledger LOG_{da} provides $1/2$-safety and $1/2$-liveness at all times (except with probability $\operatorname{negl}(\sigma) + \operatorname{negl}(\lambda)$).*
3. ***(Prefix)*** *$\mathsf{LOG}_{\text{acc}}$ is always a prefix of LOG_{da}.*

Here, negl(.) denotes a negligible function that decays faster than all polynomials. To prove Theorem 2, we first focus on the security of LOG_{da} under $(\mathcal{A}_{\text{da}}, \mathcal{Z}_{\text{da}})$, synchronous network with dynamic availability ($\boxed{11}$ of Fig. 3). We know from [35,13,22,2] that Π_{lc} is safe and live with some security parameter σ under the original LC rule when $\beta < 1/2$ ($\boxed{10}$). Hence, if k_{cp} is selected as an appropriate linear function of σ, once a block becomes k_{cp}-deep at time s in the LC held by an honest node, it stays on the LCs held by all honest nodes forever. Since there are at least $n - f > f$ accept votes for any block checkpointed by an honest node at time s, there is at least one honest node that voted accept for any such block. As honest nodes accept only proposals that are at least k_{cp}-deep in their LCs, ($\boxed{9}$), checkpointed blocks are already part of the LCs held by every other honest node at time s under $(\mathcal{A}_{\text{da}}, \mathcal{Z}_{\text{da}})$. Thus, new checkpoints can only appear in the common prefix of the honest nodes' LCs and do not affect the security of the LC protocol.

Next accountability and liveness of $\mathsf{LOG}_{\text{acc}}$ under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$ ($\boxed{3}$, $\boxed{8}$). The pseudocode of Π_{acc} stipulates that honest nodes accept only proposals that are consistent with previous checkpoints ($\boxed{1}$), and a new checkpoint requires $(n-f)$ accept votes (l. 9 of Alg. 2). Thus, in the event of a safety violation, either there are two inconsistent ledgers $\mathsf{LOG}_{\text{bft}}$ held by honest nodes, or $(n-2f)$ nodes have voted for inconsistent checkpoints. In both cases, $(n-2f)$ adversarial nodes are identified as violators by invoking either $(n-2f)$-accountable-safety of $\mathsf{LOG}_{\mathrm{bft}}$ ($\boxed{2}$) or the consistency requirement for checkpoints ($\boxed{1}$), implying $(n-2f)$-accountable-safety of $\mathsf{LOG}_{\mathrm{acc}}$. Detailed proof in App. B.2.

Liveness of $\mathsf{LOG}_{\mathrm{acc}}$ ($\boxed{8}$) requires the existence of iterations after $\max(\mathsf{GST}, \mathsf{GAT})$ where all honest nodes accept honest proposals. This, in turn, depends on whether the proposals by honest leaders are consistent with the checkpoint-respecting LCs at honest nodes after $\max(\mathsf{GST}, \mathsf{GAT})$. To show this, we prove that Π_{lc} recovers its security after $\max(\mathsf{GST}, \mathsf{GAT})$ ($\boxed{6}$). We first observe that with checkpoints, honest nodes abandon their LC if a new checkpoint appears on another (possibly shorter) chain. Then, some honest blocks produced meanwhile may not contribute to chain growth. This feature of checkpoint-respecting LCs violates a core assumption of the standard proof techniques [17,35,22] for LC protocols. To bound the number of abandoned honest blocks and demonstrate the *self-healing* property of checkpoint respecting LCs, we follow an approach introduced in [37]. We first observe the *gap* and *recency* properties for Π_{acc} (App. B.4) which are necessary conditions for any checkpointing mechanism to ensure self-healing of Π_{lc} ($\boxed{4}$). The gap property states that T_{cp} has to be sufficiently longer than the time it takes for a proposal to get checkpointed. The recency property requires that newly checkpointed blocks were held in the checkpoint-respecting LC of at least one honest node within a short time interval before the checkpoint decision.

Using the gap and recency properties, we next extend the analysis of [37] to permissioned protocols by introducing the concept of *checkpoint-strong pivots*, a generalization of strong pivots [35]. Whereas strong pivots count honest and adversarial blocks to claim convergence of the LC in the view of different honest nodes, checkpoint-strong pivots consider only honest blocks that are guaranteed to extend the checkpoint-respecting LC, thus resolving non-monotonicity for these chains. Recurrence of checkpoint-strong pivots after $\max(\mathsf{GST}, \mathsf{GAT})$ ($\boxed{5}$) along with the gap and recency properties lead to security of Π_{lc} after $\max(\mathsf{GST}, \mathsf{GAT})$. Details in App. C. Given self-healing of Π_{lc}, liveness of $\mathsf{LOG}_{\mathrm{acc}}$ follows from liveness of Π_{bft} after $\max(\mathsf{GST}, \mathsf{GAT})$ ($\boxed{7}$). Full proof in App. B.3.

Finally, the prefix property follows readily from the way in which both $\mathsf{LOG}_{\mathrm{da}}$ and $\mathsf{LOG}_{\mathrm{acc}}$ are derived from the checkpoint-respecting LC.

5 Experimental Evaluation

To evaluate whether the protocol of Section 4.1 can be a drop-in replacement for the Ethereum 2 beacon chain, we have implemented a prototype[4]. Our protocol incurs average required bandwidth comparable to Gasper at reduced latency of $\mathsf{LOG}_{\mathrm{acc}}$. Gasper's resilience decreases as the number of nodes increases, for fixed latency of $\mathsf{LOG}_{\mathrm{acc}}$, due to a new attack [30], whereas our protocol is provably secure. Supplemental material of experimental evaluation is given in Appendix D.

A diagram of the different components of our prototype and their interactions is provided in Figure 4. We use a longest chain protocol modified to respect latest

[4] Source code: https://github.com/tse-group/accountability-gadget-prototype

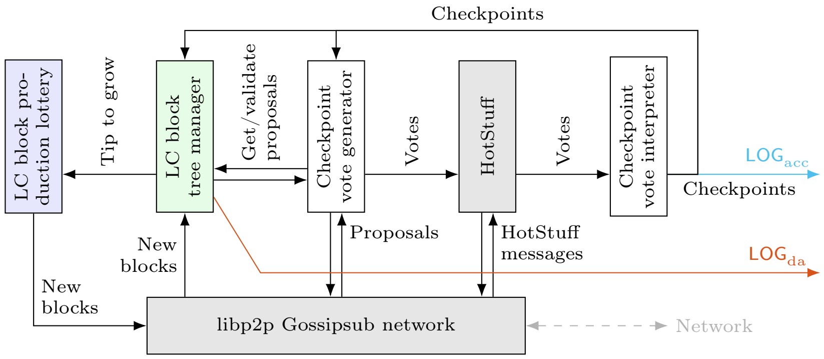

Fig. 4. Components and their interactions in implementation of Fig. 1. Gray: off the shelf components used as black boxes. Blue: taken from Π_{lc} without modification. Green: taken from Π_{lc}, modified to respect checkpoints.

checkpoints as Π_{lc}, with a permissioned block production lottery; and a variant of HotStuff[5] as Π_{bft}. All communication (including HotStuff's) takes place in a broadcast fashion via `libp2p`'s `Gossipsub` protocol[6], mimicking Ethereum 2 [1]. The parameters of our protocol match the number of validators ($n = 4096$), average block inter-arrival time (12 s) and block payload size (22 KBytes) of the Ethereum 2 beacon chain. We chose $k_{\mathrm{cp}} = 6$ so that an honest checkpoint proposal is likely accepted by honest nodes, and $k = 6$ for swift 72 s average latency of $\mathsf{LOG}_{\mathrm{da}}$. Setting $T_{\mathrm{hs}} = 20\,\mathrm{s}$ and $T_{\mathrm{to}} = 1\,\mathrm{min}$ avoids HotStuff timeouts escalating into checkpoint timeouts unnecessarily. Finally, to target 5× improvement in average $\mathsf{LOG}_{\mathrm{acc}}$ latency over Gasper (*cf.* Figure 7), we set $T_{\mathrm{cp}} = 5\,\mathrm{min}$.

Adversarial nodes in the experiment boycott leader duty in Π_{bft} and mine selfishly [16] in Π_{lc}. We ran our prototype (a) with no adversary (Fig. 2(l)), and (b) with $\beta = 25\%$ adversary (Fig. 2(r)), each for 2500 s on five AWS EC2 `c5a.8xlarge` instances in each of ten AWS regions with 82 nodes per machine, for a total of 4100 nodes. Each honest (adversarial) node connected to 15 (15 honest, 15 adversarial) randomly selected peers for the peer-to-peer network. Both without (Fig. 2(l)) and under attack (Fig. 2(r)) $\mathsf{LOG}_{\mathrm{da}}$ (—) grows steadily, albeit under attack slower due to selfish mining. In both cases, $\mathsf{LOG}_{\mathrm{acc}}$ (—) periodically catches up with $\mathsf{LOG}_{\mathrm{da}}$. Timeouts cause minor delayed catch-up.

Network traffic (Figs. 6, 5 *for an exemplary AWS instance*, *i.e.*, for 82 nodes) shows frequent small spikes for Π_{lc} blocks and infrequent wide spikes for Π_{acc} votes and Π_{bft} blocks and votes. Traffic increases slightly under attack (per node: avg. 78 KB/s vs. 56 KB/s, peak 1.5 MB/s vs. 1.34 MB/s) because inactive adversarial leaders cause more iterations in Π_{acc} and Π_{bft}. The bandwidth requirement does not limit participation using consumer-grade Internet access. Note that our prototype does not employ bandwidth reduction techniques that

[5] We used this Rust implementation: https://github.com/asonnino/hotstuff [18]

[6] We used this Rust implementation: https://github.com/libp2p/rust-libp2p [41]

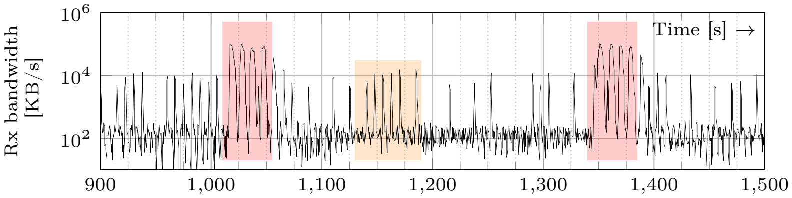

Fig. 5. Setting of Fig. 2(l): The network traffic for each AWS instance (*i.e.*, 82 nodes) shows four marked spikes (red) for every new checkpoint ($T_{\mathrm{cp}} = 5\,\mathrm{min}$ interval) and smaller spikes (orange) for every new Π_{lc} block ($T_{\mathrm{slot}} = 7.5\,\mathrm{s}$ interval).

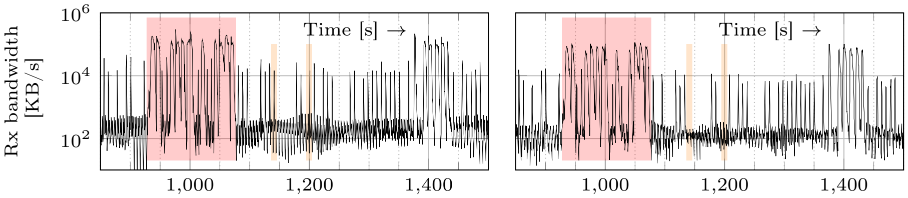

Fig. 6. Setting of Fig. 2(r): Leader timeouts in Π_{bft} and Π_{acc} can delay new checkpoints (red). *E.g.*, after the end of a checkpoint interval ($t \approx 870\,\mathrm{s}$), and subsequent Π_{acc} leader timeout ($t \approx 930\,\mathrm{s}$), honest nodes vote to reject the current checkpoint iteration, but the decision is delayed by another Π_{bft} leader timeout. The next checkpoint iteration has an honest leader, but a decision is again delayed by a Π_{bft} leader timeout, until a new checkpoint is finally reached ($t \approx 1070\,\mathrm{s}$). Traffic at honest nodes (**right**) lacks some of the small spikes (orange) of traffic at adversarial nodes (**left**), since the adversary temporarily withholds some of its blocks from honest nodes due to selfish mining.

are orthogonal to the consensus problem, such as aggregate and short signatures or spreading the vote out over time. Figure 7 corroborates that even if voting was artificially rate-limited and thus spread out over time (as is the case in Gasper), bandwidth and latency comparable to Gasper could be achieved.

Figure 7 compares bandwidth and latency of $\mathsf{LOG}_{\mathrm{acc}}$ for varying parameters and $\beta = 0, \Delta = 0$. Gasper transmits $2 \cdot \frac{n}{C}$ votes per 12 s, with C the number of slots per epoch, our protocol transmits $5 \cdot n$ votes per T_{cp} time. A transaction takes on average $\frac{1}{2}+2$ epochs to enter into $\mathsf{LOG}_{\mathrm{acc}}$ for Gasper, and $k_{\mathrm{cp}} \cdot 12\,\mathrm{s} + \frac{1}{2} \cdot T_{\mathrm{cp}}$ time to enter $\mathsf{LOG}_{\mathrm{acc}}$ for our protocol. Our protocol offers slightly improved latency at comparable bandwidth, or comparable bandwidth and latency but for a larger number of nodes.

Acknowledgment

JN, ENT, and DT are supported by the Reed-Hodgson Stanford Graduate Fellowship, the Stanford Center for Blockchain Research, and the Center for Science of Information (CSoI), an NSF Science and Technology Center under grant agreement CCF-0939370, respectively.

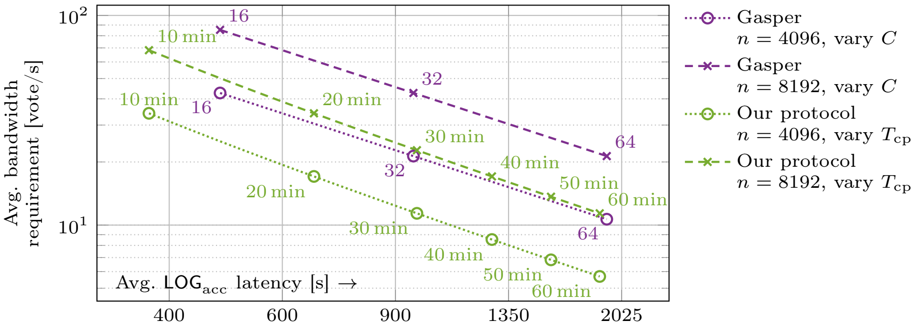

Fig. 7. For fixed n, the average latency of $\mathsf{LOG}_{\mathrm{acc}}$ for Gasper and our protocol (here for $k_{\mathrm{cp}} = 6$) increases with the number C of slots per epoch and with T_{cp}, respectively, while the bandwidth required for votes reduces proportionally. Our protocol offers a better tradeoff and can tolerate twice the n at comparable latency and bandwidth (our protocol for $n = 8192, T_{\mathrm{cp}} = 30\,\mathrm{min}$ vs. Gasper for $n = 4096, C = 32$).

References

1. Ethereum 2.0 networking specification (2021), https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md
2. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In: Conference on Computer and Communications Security. pp. 913–930. CCS '18, ACM (2018)
3. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Consensus Redux: Distributed ledgers in the face of adversarial supremacy. IACR Cryptology ePrint Archive, Report 2020/1021 (2020)
4. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938 (2018), https://arxiv.org/abs/1807.04938
5. Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv:1710.09437 (2019), https://arxiv.org/abs/1710.09437
6. Buterin, V., Hernandez, D., Kamphefner, T., Pham, K., Qiao, Z., Ryan, D., Sin, J., Wang, Y., Zhang, Y.X.: Combining GHOST and Casper. arXiv:2003.03052 (2020), https://arxiv.org/abs/2003.03052
7. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Symposium on Operating Systems Design and Implementation. p. 173–186. OSDI '99, USENIX Association (1999)
8. Chan, B.Y., Shi, E.: Streamlet: Textbook streamlined blockchains. In: Advances in Financial Technologies. p. 1–11. AFT '20, ACM (2020)
9. Chen, J., Gorbunov, S., Micali, S., Vlachos, G.: ALGORAND AGREEMENT: Super fast and partition resilient Byzantine agreement. IACR Cryptology ePrint Archive, Report 2018/377 (2018)
10. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theoretical Computer Science **777**, 155–183 (2019)
11. Civit, P., Gilbert, S., Gramoli, V.: Polygraph: Accountable Byzantine agreement. In: ICDCS. pp. 403–413. IEEE (2021)
12. Cohen, B., Pietrzak, K.: The Chia network blockchain. https://www.chia.net/assets/ChiaGreenPaper.pdf (2019)

13. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: EUROCRYPT 2018. pp. 66–98. Springer (2018)
14. Dembo, A., Kannan, S., Tas, E.N., Tse, D., Viswanath, P., Wang, X., Zeitouni, O.: Everything is a race and Nakamoto always wins. In: Conference on Computer and Communications Security. p. 859–878. CCS '20, ACM (2020)
15. Dinsdale-Young, T., Magri, B., Matt, C., Nielsen, J., Tschudi, D.: Afgjort: A partially synchronous finality layer for blockchains. In: Conference on Security and Cryptography for Networks. pp. 24–44. SCN '20 (2020)
16. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM **61**(7), 95–102 (2018)
17. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT 2015. pp. 281–310. Springer (2015)
18. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and Ditto: Network-adaptive efficient consensus with asynchronous fallback. arXiv:2106.10362 (2021), https://arxiv.org/abs/2106.10362, Forthcoming in Financial Cryptography 2022
19. Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. In: CRYPTO 2019. pp. 499–529. Springer (2019)
20. Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. SIGOPS Oper. Syst. Rev. **41**(6), 175–188 (Oct 2007)
21. Haeberlen, A., Kuznetsov, P.: The Fault Detection Problem. In: International Conference on Principles of Distributed Systems. OPODIS '09 (2009)
22. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO 2017. pp. 357–388. Springer (2017)
23. Lewis-Pye, A., Roughgarden, T.: Resource pools and the CAP theorem. arXiv:2006.10698 (2020), https://arxiv.org/abs/2006.10698
24. Lewis-Pye, A., Roughgarden, T.: How does blockchain security dictate blockchain implementation? In: CCS. pp. 1006–1019. ACM (2021)
25. Libra Association: Libra white paper. https://libra.org/en-US/white-paper/ (2020)
26. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
27. Nakamura, R.: Analysis of bouncing attack on FFG (2019), https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113
28. Neu, J., Tas, E.N., Tse, D.: A balancing attack on Gasper, the current candidate for Eth2's beacon chain (2020), https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079
29. Neu, J., Tas, E.N., Tse, D.: Snap-and-Chat protocols: System aspects. arXiv:2010.10447 (2020), https://arxiv.org/abs/2010.10447
30. Neu, J., Tas, E.N., Tse, D.: Attacking Gasper without adversarial network delay (2021), https://ethresear.ch/t/attacking-gasper-without-adversarial-network-delay/10187
31. Neu, J., Tas, E.N., Tse, D.: The availability-accountability dilemma and its resolution via accountability gadgets. arXiv:2105.06075 (2021), https://arxiv.org/abs/2105.06075
32. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: Symposium on Security and Privacy. S&P '21, IEEE (2021)
33. Neu, J., Tas, E.N., Tse, D.: Two attacks on proof-of-stake GHOST/Ethereum. arXiv:2203.01315 (2022), https://arxiv.org/abs/2203.01315

34. Pass, R., Shi, E.: Rethinking large-scale consensus. In: Computer Security Foundations Symposium. pp. 115–129. CSF '17, IEEE (2017)
35. Pass, R., Shi, E.: The sleepy model of consensus. In: ASIACRYPT 2017. pp. 380–409. Springer (2017)
36. Ranchal-Pedrosa, A., Gramoli, V.: Blockchain is dead, long live blockchain! Accountable state machine replication for longlasting blockchain. arXiv:2007.10541 (2020), https://arxiv.org/abs/2007.10541
37. Sankagiri, S., Wang, X., Kannan, S., Viswanath, P.: Blockchain CAP theorem allows user-dependent adaptivity and finality. In: Financial Cryptography and Data Security. FC '21 (2021)
38. Schwarz-Schilling, C., Neu, J., Monnot, B., Asgaonkar, A., Tas, E.N., Tse, D.: Three attacks on proof-of-stake Ethereum. arXiv:2110.10086 (2021), https://arxiv.org/abs/2110.10086, Forthcoming in Financial Cryptography 2022
39. Sheng, P., Wang, G., Nayak, K., Kannan, S., Viswanath, P.: BFT protocol forensics. In: CCS. pp. 1722–1743. ACM (2021)
40. Stewart, A., Kokoris-Kogia, E.: GRANDPA: A Byzantine finality gadget. arXiv:2007.01560 (2020), https://arxiv.org/abs/2007.01560
41. Vyzovitis, D., Napora, Y., McCormick, D., Dias, D., Psaras, Y.: GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks. arXiv:2007.02754 (2020), http://arxiv.org/abs/2007.02754
42. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Symposium on Principles of Distributed Computing. p. 347–356. PODC '19, ACM (2019)

A Proof of Availability-Accountability Dilemma

A formal proof of Theorem 1 building on the observations discussed in Section 3 is as follows:

Proof. For the sake of contradiction, suppose there exists an SMR protocol Π that provides β_{l}-liveness and β_{a}-accountable-safety for some $\beta_{\mathrm{l}}, \beta_{\mathrm{a}} > 0$ under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$. Then, there exists an adjudication function $\mathcal{J}$, which given two sets of evidences attesting to conflicting ledgers, outputs a non-empty set of adversarial nodes.

Suppose there are n nodes in $\mathcal{Z}$. Without loss of generality, we may assume that n is even; otherwise, $\mathcal{Z}$ puts one node to sleep throughout the execution. Let P and Q partition the n nodes into two disjoint equal groups with $|P| = |Q| = n/2$. We denote by $[\mathsf{tx}]$ a ledger consisting of a single transaction tx at its first index.

Next consider the following worlds:

World 1: Nodes in P are honest and awake throughout the execution. $\mathcal{Z}$ inputs tx_1 to them. Nodes in Q are asleep. Since Π satisfies liveness for some $\beta_{\mathrm{l}} > 0$ under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$, nodes in P eventually generate a set of evidences W_1 such that $\mathcal{C}(W_1) = [\mathsf{tx}_1]$.

World 2: Nodes in Q are honest and awake throughout the execution. $\mathcal{Z}$ inputs tx_2 to them. Nodes in P are asleep. Since Π satisfies liveness for some $\beta_{\mathrm{l}} > 0$ under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$, nodes in Q eventually generate a set of evidences W_2 such that $\mathcal{C}(W_2) = [\mathsf{tx}_2]$.

World 3: $\mathcal{Z}$ wakes up all n nodes, and inputs tx_1 to the nodes in P and tx_2 to the nodes in Q. Nodes in P are honest. Nodes in Q are adversarial and do not communicate with the nodes in P. All nodes stay awake throughout the execution. Since the worlds 1 and 3 are indistinguishable for the nodes in P, they eventually generate a set of evidences W_1 such that $\mathcal{C}(W_1) = [\mathsf{tx}_1]$. Nodes in Q simulate the execution in world 2 without any communication with the nodes in P. Hence, they eventually generate a set of evidences W_2 such that $\mathcal{C}(W_2) = [\mathsf{tx}_2]$. Thus, there is a safety violation. So, $\mathcal{J}$ takes W_1 and W_2, and outputs a non-empty set $S_3 \subseteq Q$ of adversarial nodes.

World 4: $\mathcal{Z}$ wakes up all n nodes, and inputs tx_1 to the nodes in P and tx_2 to the nodes in Q. Nodes in Q are honest. Nodes in P are adversarial and do not communicate with the nodes in Q. All nodes stay awake throughout the execution. Since the worlds 2 and 4 are indistinguishable for the nodes in Q, they eventually generate a set of evidences W_2 such that $\mathcal{C}(W_2) = [\mathsf{tx}_2]$. Nodes in P simulate the execution in world 1 without any communication with the nodes in Q. Hence, they eventually generate a set of evidences W_1 such that $\mathcal{C}(W_1) = [\mathsf{tx}_1]$. Thus, there is a safety violation. So, $\mathcal{J}$ takes W_1 and W_2, and outputs a non-empty set $S_4 \subseteq P$ of adversarial nodes.

Note however that worlds 3 and 4 are indistinguishable from the perspective of the adjudication function $\mathcal{J}$. Thus, it is not possible that $\mathcal{J}$ reliably outputs a non-empty set which in the case of world 3 contains only elements of Q and in the case of world 4 contains only elements of P, as would be required by Definition 4. □

B Security Proof for Accountability Gadgets

B.1 Theorem Statement and Notation

In this section, we consider an accountability gadget Π_{acc} instantiated with a BFT protocol Π_{bft} that provides $(n-2f)$-accountable-safety at all times, and f-liveness after $\max(\mathsf{GST}, \mathsf{GAT})$ under $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$. To match the accountable safety resilience of Π_{bft} on Π_{acc}, we tune the thresholds for the number of accept and reject votes required to output a new checkpoint as $(n-f)$ and $f+1$ respectively on lines 9 and 12 of Algorithm 2.

Recall that Π_{acc} is used on top of a Nakamoto-style permissioned longest chain (LC) protocol Π_{lc}. For concreteness and notational purposes, we assume that Π_{acc} is the Sleepy consensus protocol [35] although we could have used any other permissioned LC protocol in its place.

Given the accountability gadget Π_{acc} and the LC protocol Π_{lc}, goal of this section is to prove that the ledgers $\mathsf{LOG}_{\mathrm{acc}}$ and $\mathsf{LOG}_{\mathrm{da}}$ outputted by Π_{acc} and Π_{lc} satisfy Theorem 2 repeated below:

Given any security parameter σ and $f \leq \lceil n/2 \rceil$,

1. (**P1:Accountability**) Under $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$, the accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ provides $n-2f$-accountable safety at all times, and there exists a constant $\mathbf{C}$ such that $\mathsf{LOG}_{\mathrm{acc}}$ provides f-liveness (with confirmation time polynomial in

σ) after $\mathbf{C}\max(\mathsf{GST},\mathsf{GAT})$ except with probability $\mathrm{negl}(\sigma)$.

2. (**P2:Dynamic Availability**) Under $(\mathcal{A}_{\mathrm{da}},\mathcal{Z}_{\mathrm{da}})$, the available ledger $\mathsf{LOG}_{\mathrm{da}}$ is guaranteed to be safe and live at all times, provided that $\beta < 1/2$.
3. (**Prefix**) $\mathsf{LOG}_{\mathrm{acc}}$ is always a prefix of $\mathsf{LOG}_{\mathrm{da}}$.

Before proceeding with the proofs, we formalize the concept of *security after a certain time* (We write $\mathsf{LOG} \preceq \mathsf{LOG}'$ if LOG is a prefix of LOG'.):

Definition 6. *Let T_{confirm} be a polynomial function of the security parameter σ. We say that a ledger LOG is* secure after time T *and has transaction confirmation time T_{confirm} if LOG satisfies:*

- ***Safety:*** *For any two times $t \geq t' \geq T$, and any two honest nodes i and j awake at times t and t' respectively, either $\mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^{t'}$ or $\mathsf{LOG}_j^{t'} \preceq \mathsf{LOG}_i^t$.*
- ***Liveness:*** *If a transaction is received by an awake honest node at some time $t \geq T$, then, for any time $t' \geq t + T_{\mathrm{confirm}}$ and honest node j that is awake at time t', the transaction will be included in $\mathsf{LOG}_j^{t'}$.*

Definition 6 formalizes the meaning of 'safety, liveness and security *after a certain time T*'. In general, there might be two different times after which a protocol is safe or live. A protocol that is safe (live) at all times (i.e, after $T = 0$) is simply called *safe* (*live*) without further qualification.

B.2 Accountable Safety Resilience

We first show that $\mathsf{LOG}_{\mathrm{acc}}$ provides $n-2f$-accountable safety under $(\mathcal{A}_{\mathrm{pda}},\mathcal{Z}_{\mathrm{pda}})$.

Proposition 1. *Suppose the number of adversarial nodes is less than $n-2f$. Then, if a block b is checkpointed for iteration c in the view of an honest node i at slot t, for any honest node j and slot s, either b is checkpointed for iteration c at slot s or no block has been checkpointed for iteration c yet.*

Proposition 1 follows from the safety of $\mathsf{LOG}_{\mathrm{bft}}$ when the number of adversarial nodes is less than $n-2f$.

Theorem 3 (Accountable Safety of $\mathsf{LOG}_{\mathrm{acc}}$). $\mathsf{LOG}_{\mathrm{acc}}$ *provides $n-2f$-accountable-safety.*

Proof. To show that $\mathsf{LOG}_{\mathrm{acc}}$ provides $n-2f$-accountable-safety, we construct an adjudication protocol $\mathcal{J}$, which in the case of a safety violation on $\mathsf{LOG}_{\mathrm{acc}}$, outputs at least $n-2f$ nodes as adversarial and never outputs an honest node. For this purpose, suppose there is a safety violation on $\mathsf{LOG}_{\mathrm{acc}}$. Then, there exist honest nodes i and j, iterations c and c' (without loss of generality $c' \leq c$) and slots s and t such that (i) a block $b_1 \neq \perp$ is checkpointed for iteration c' in the view of node i at slot s, (ii) a block $b_2 \neq \perp$ is checkpointed for iteration c in the view of node j at slot t, (iii) b_1, b_2 conflict with each other. Then, within $\mathsf{LOG}^i_{\mathrm{bft},s}$, there are accept votes $\langle \mathsf{accept}, c', b_1\rangle_k$ from at least $n-f$ nodes for the proposal b_1 and iteration c'. Similarly, within $\mathsf{LOG}^j_{\mathrm{bft},t}$, there are accept votes $\langle \mathsf{accept}, c, b_2 \rangle_k$ from at least $n-f$ nodes for the proposal b_2 and iteration c. Thus, more than $2(n-f)-n = n-2f$ nodes voted both $\langle \mathsf{accept}, c', b_1 \rangle_k$ and $\langle \mathsf{accept}, c, b_1 \rangle_k$. Let S denote the set of these nodes.

Next, consider the following two cases:

(i) There is a safety violation on $\mathsf{LOG}_{\mathrm{bft}}$. (Recall that $\mathsf{LOG}_{\mathrm{bft}}$ provides $n-2f$-accountable safety.) In this case, since the adjudication protocol for $\mathsf{LOG}_{\mathrm{bft}}$ identifies at least $n-2f$ nodes as adversarial, $\mathcal{J}$ simply returns the output of the adjudication protocol for $\mathsf{LOG}_{\mathrm{bft}}$.

(ii) Suppose there is no safety violation on $\mathsf{LOG}_{\mathrm{bft}}$ and $c' < c$. Then, via Proposition 1, every node k in S have either seen b_1 become checkpointed for iteration c' before voting accept $\langle \mathsf{accept}, c, b_2 \rangle_k$ or voted for iteration c before seeing any checkpoint for iteration c'. However, an honest node votes accept for the proposal b_2 of iteration c only if it has already seen a block checkpointed for all past iterations including c', and if b_2 is consistent with all of the checkpoints from the past iterations, including b_1. (This is because an honest node votes accept for a proposal only if it is part of the node's checkpoint-respecting LC.) Then, no honest node could have voted both $\langle \mathsf{accept}, c', b_1 \rangle_k$ and $\langle \mathsf{accept}, c, b_2 \rangle_k$, implying that all of the nodes in S have violated the protocol. If $c' = c$, then all of the nodes in S voted accept twice for two different proposals for iteration c, which is again a protocol violation.

Finally, when there is no safety violation on $\mathsf{LOG}_{\mathrm{bft}}$, $\mathsf{LOG}^s_{\mathrm{bft},i} \preceq \mathsf{LOG}^t_{\mathrm{bft},j}$ or $\mathsf{LOG}^t_{\mathrm{bft},j} \preceq \mathsf{LOG}^s_{\mathrm{bft},i}$. In either case, all of the accept votes $\langle \mathsf{accept}, c', b_1 \rangle_k$ and $\langle \mathsf{accept}, c, b_2 \rangle_k$ are within the longer of $\mathsf{LOG}^s_{\mathrm{bft},i}$ and $\mathsf{LOG}^t_{\mathrm{bft},j}$, which can be used to prove that the nodes in S violated the protocol. Hence, in this case, $\mathcal{J}$ returns S, which contains at least $n-2f$ nodes as the set of nodes that have irrefutably violated the protocol. □

B.3 Liveness Resilience

We next focus on the liveness of $\mathsf{LOG}_{\mathrm{acc}}$.

Proposition 2. *Π_{bft} satisfies f-liveness after* $\max(\mathsf{GST}, \mathsf{GAT})$ *with transaction confirmation time T_{confirm}.*

Proposition 3. *Suppose a block from iteration c was checkpointed in the view of an honest node at time t. Then, every honest node enters iteration $c+1$ by time* $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$.

Proof. Suppose a block from iteration c was checkpointed in the view of an honest node i at time t. Then, there are at least $n-f$ accept votes for the block from iteration c on $\mathsf{LOG}^t_{\mathrm{bft},i}$. Note that all checkpointing votes and BFT protocol messages observed by node i by time t are delivered to all other honest nodes by time $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$. Hence, via Theorem 3, for any honest node j, $\mathsf{LOG}^i_{\mathrm{bft},t} \preceq \mathsf{LOG}^j_{\mathrm{bft},\max(\mathsf{GST},\mathsf{GAT},t)+\Delta}$. Thus, for any honest node j, there are at least $n-f$ accept votes on $\mathsf{LOG}^j_{\mathrm{bft},\max(\mathsf{GST},\mathsf{GAT},t)+\Delta}$ for the same block from iteration c. This implies that every honest node enters iteration $c+1$ by time $\max(\mathsf{GST}, \mathsf{GAT}, t) + \Delta$. □

Theorem 4 (Liveness of $\mathsf{LOG}_{\mathrm{acc}}$). *Suppose Π_{lc} is secure (safe and live) after some slot $T \geq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\mathrm{cp}}$. Then, $\mathsf{LOG}_{\mathrm{acc}}$ satisfies f-liveness after slot T with transaction confirmation time σ except with probability $e^{-\Omega(\sigma)}$.*

Proof. If there are f or more adversarial nodes, we know via Proposition 2 that $\mathsf{LOG}_{\mathrm{bft}}$, and by implication $\mathsf{LOG}_{\mathrm{acc}}$ will not be live. Thus, to show f-liveness of $\mathsf{LOG}_{\mathrm{acc}}$, we assume that there are less than f adversarial nodes and prove that $\mathsf{LOG}_{\mathrm{acc}}$ satisfies liveness. In this case, again via Proposition 2, we know that $\mathsf{LOG}_{\mathrm{bft}}$ satisfies liveness with transaction confirmation time T_{confirm} after $\max(\mathsf{GST}, \mathsf{GAT})$, a property which we will use subsequently.

Let c' be the largest iteration such that a block was checkpointed in the view of some honest node before $\max(\mathsf{GAT}, \mathsf{GST})$. (Let $c' = 0$ if there does not exist such an iteration.) By Proposition 3, all honest nodes would have entered iteration $c'+1$ by slot $\max(\mathsf{GAT}, \mathsf{GST})+\Delta$. Then, all honest nodes observe a block proposed for iteration $c'+1$ by slot $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{cp}}$. Thus, entrance times of the honest nodes to subsequent iterations have become synchronized by slot $\max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{cp}}$: If an honest node enters an iteration $c > c'$ at slot $t \geq \max(\mathsf{GAT}, \mathsf{GST}) + \Delta + T_{\mathrm{cp}}$, every honest node enters iteration c' by slot $t + \Delta$.

Suppose iteration $c > c'$ has an honest leader $\mathrm{L}^{(c)}$, which sends a proposal $\hat{b}_c$ at slot $t > T + T_{\mathrm{cp}}$. Note that $\hat{b}_c$ is received by every honest node by slot $t + \Delta$. Since the entrance times of nodes are synchronized by $T \leq \max(\mathsf{GST}, \mathsf{GAT}) + \Delta + T_{\mathrm{cp}}$, every honest node votes by slot $t+\Delta$. Now, as Π_{lc} is secure after slot T, $\hat{b}_c$ is on all of the checkpoint-respecting LCs held by the honest nodes. Moreover, as we will argue in the paragraph below, $\hat{b}_c$ extends all of the checkpoints seen by the honest nodes by slot $t + \Delta$. (Honest nodes see the same checkpoints from iterations preceding c due to synchrony.) Consequently, every honest node votes $\langle \mathsf{accept}, c, \hat{b}_c \rangle_k$ for $\hat{b}_c$ by slot $t+\Delta$, all of which appear within $\mathsf{LOG}_{\mathrm{bft}}$ in the view of every honest node by slot $t + \Delta + T_{\mathrm{confirm}}$. Thus, $\hat{b}_c$ becomes checkpointed in the view of every honest node by slot $t + \Delta + T_{\mathrm{confirm}}$. (Here, we assume that T_{to} was chosen large enough for $T_{\mathrm{to}} > \Delta + T_{\mathrm{confirm}}$ to hold.)

Note that $\mathrm{L}^{(c)}$ waits for T_{cp} slots before broadcasting $\hat{b}_c$ after observing the last checkpoint block before iteration c. Since $t - T_{\mathrm{cp}} > T$, during the period $[t-T_{\mathrm{cp}}, t]$, Π_{lc} satisfies the chain growth and quality properties (see Appendix C). Thus, for a large enough T_{cp}, the checkpoint-respecting LC of $\mathrm{L}^{(c)}$ at time t contains at least one honest block between $\hat{b}_c$ and the last checkpointed block on it from before iteration c. (As a corollary, $\mathrm{L}^{(c)}$ extends all of the previous checkpoints seen by itself and all other honest nodes.) This implies that $\hat{b}_c$ contains at least one fresh honest block that enters $\mathsf{LOG}_{\mathrm{acc}}$ by slot $t + \Delta + T_{\mathrm{confirm}}$.

Next, we show that an adversarial leader cannot make an iteration last longer than $\Delta+T_{\mathrm{to}}+T_{\mathrm{confirm}}$ for any honest node. Indeed, if an honest node i enters an iteration c at slot t, by slot $t+\Delta+T_{\mathrm{to}}$, either it sees a block become checkpointed for iteration c, or every honest node votes reject. In the first case, every honest node sees a block checkpointed for iteration c by slot at most $t+2\Delta+T_{\text{to}}$. In the second case, reject votes $\langle \mathsf{reject}, c\rangle_k$ from at least $n-f>f$ of the nodes appear in $\mathsf{LOG}_{\text{bft}}$ in the view of every honest node by slot at most $t+\Delta+T_{\text{to}}+T_{\text{confirm}}$. Hence, a new checkpoint, potentially $\perp$, is outputted in the view of every honest node by slot $t+\Delta+T_{\text{to}}+T_{\text{confirm}}$.

Finally, we observe that except with probability $(f/n)^m$, there exist an iteration with an honest leader within m consecutive iterations. Since an iteration lasts at most $\max(\Delta+T_{\text{to}}+T_{\text{confirm}}, \Delta+T_{\text{confirm}}+T_{\text{cp}}) \leq \Delta+T_{\text{to}}+T_{\text{confirm}}+T_{\text{cp}}$ slots and a new checkpoint containing a fresh honest block in its prefix appears when an iteration has an honest leader, any transaction received by an awake honest node at slot t appears within $\mathsf{LOG}_{\text{acc}}$ in the view of every honest node by slot at most $\max(t,T)+m(\Delta+T_{\text{to}}+T_{\text{confirm}}+T_{\text{cp}})$ except with probability $(f/n)^m$. Hence, via a union bound over the total number of iterations (which is a polynomial in σ), we observe that if Π_{lc} is secure after some slot T, then $\mathsf{LOG}_{\text{acc}}$ satisfies liveness after T with a transaction confirmation time polynomial in σ except with probability $e^{-\Omega(\sigma)}$. □

Observe that Theorem 4 requires Π_{lc} to eventually regain its security under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$ when there are less than f adversarial nodes. Although it is not possible to guarantee any security property for Π_{lc} before GST, the following theorem states that Π_{lc} recovers its security after $\max(\mathsf{GST}, \mathsf{GAT})$. Note that Π_{lc} is initialized with a parameter p which denotes the probability that a given node is elected as a block producer in a given slot.

Theorem 5 (Security of Π_{lc}). *If $p < (n-2f)/(2\Delta n(n-f))$ and there are f (or less) adversarial nodes, for each sufficiently large T_{cp}, there exists a constant $\mathbf{C}>0$ such that for any GST and GAT specified by $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$, $\Pi_{\text{lc}}(p)$ is secure after $\mathbf{C}(\max(\mathsf{GST},\mathsf{GAT})+\sigma)$, with transaction confirmation time σ, except with probability $e^{-\Omega(\sqrt{\sigma})}$.*

Proof of the theorem is given in Section C and relies on a combination of the method outlined in [37, Appendix C] with the concept of strong pivots from [35].

Finally, since $f<n/2$, we can always find a p such that $p < (n-2f)/(2\Delta n(n-f))$. Then, given Theorems 5 and 4 and a sufficiently small p, we can assert that $\mathsf{LOG}_{\text{acc}}$ satisfies f-liveness with a transaction confirmation time polynomial in σ after time $\mathbf{C}(\max(\mathsf{GST},\mathsf{GAT})+\sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$.

B.4 Recency and Gap Properties

Proof of Theorem 5 requires the accountability gadget Π_{acc} to satisfy two main properties first introduced in [37]: *recency* and *gap* properties.

Gap property states that blocks are checkpointed sufficiently apart in time, controlled by the parameter T_{cp}:

Proposition 4 (Gap Property). *Given any time interval $[t_1,t_2]$, no more than $(1+(t_2-t_1))/T_{\text{cp}}$ blocks can be checkpointed in the interval.*

Proof of Proposition 4 follows from the fact that upon observing a new checkpoint that is not $\perp$ for an iteration, honest nodes wait for T_{cp} slots before voting for the proposal of the next iteration.

Following the notation in [37], we say that a block checkpointed for iteration c at slot $t > \max(\mathsf{GST}, \mathsf{GAT})$ in the view of an honest node i is T_{rec}-recent if it has been part of the checkpoint-respecting LC of some honest node j at some slot within $[t - T_{\mathrm{rec}}, t]$. Then, we can express the recency property as follows:

Lemma 1 (Recency Property). *Every checkpointed block proposed after* $\max(\mathsf{GST}, \mathsf{GAT})$ *is* T_{rec}*-recent for* $T_{\mathrm{rec}} = \Delta + T_{\mathrm{to}} + T_{\mathrm{confirm}}$.

Proof. We have seen in the proof of Theorem 4 that if a block b proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is checkpointed in the view of an honest node at slot t, it should have been proposed after slot $t - (\Delta + T_{\mathrm{to}} + T_{\mathrm{confirm}})$. Thus, at least $n - f$ nodes must have voted $\langle \mathsf{accept}, c, b \rangle_k$ by time t. Let j denote an honest nodes which voted $\langle \mathsf{accept}, c, b \rangle_j$. Note that j would vote only after it sees the proposal for iteration c, i.e after slot $t - T_{\mathrm{rec}} = t - (\Delta + T_{\mathrm{to}} + T_{\mathrm{confirm}})$. Hence, b should have been on the checkpoint-respecting LC of node j at some slot within $[t - T_{\mathrm{rec}}, t]$. This concludes the proof that every checkpoint block proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is T_{rec}-recent. □

C Security Proof for Checkpoint-Respecting LC

In this section, we prove Theorem 5, which states that the security of Π_{lc} is restored after $\max(\mathsf{GST}, \mathsf{GAT})$ under $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$ provided that the election probability p of each node is sufficiently small. Via [32, Appendix C], we know that the Sleepy consensus protocol [35] regains its safety and liveness within $O(\max(\mathsf{GST}, \mathsf{GAT}))$ slots under $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$. On a similar note, [3] has shown the same self-healing property for Nakamoto's PoW LC protocol and other LC based PoS protocols. However, all of these works analyze the LC protocols in their original form without considering checkpoints in the chain selection rule. In this context, [37] is the first work to show the recovery of security for checkpoint-respecting LC protocols. As argued in [37], length of a checkpoint-respecting LC held by an honest node can decrease when a new checkpoint appears at a conflicting chain, thus, requiring a careful analysis to bound how many honest blocks are lost due to such instances. However, it is not immediately obvious if the analysis of [37] that relies on [17] carries over to the case of PoS protocols, where an adversary can generate multiple blocks when it is elected. Hence, our goal is to show that the PoS protocols such as [35,13,2] recover their safety and liveness after $O(\max(\mathsf{GST}, \mathsf{GAT}))$ time under $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$. In this endeavor, we enhance the proof technique of [37] by using the concept of strong pivots from [35].

In the proof below, we follow the same notation as [35]. Each node is elected as the leader of a time slot with probability p. Total number of slots $T_{\max}$ is fixed and is a polynomial in the security parameter σ. There are n nodes in total, among which f nodes are controlled by the adversary. We denote the checkpoint-respecting longest chain held by an honest node i at slot t by ch_i^t.

Define $\beta = pf$ as an upper bound on the expected number of adversary nodes elected leader in a single slot. Similarly, define α as a lower bound on the expected number of awake honest nodes elected leader in a single slot. After GAT, every honest node wakes up and $\alpha = p(n-f) > \beta$ as $f < n/2$. For the convergence opportunities, we adopt the definition given in [35, Section 5.2] and denote the number of convergence opportunities within a time interval $[t_1, t_2]$ by $\mathrm{C}([t_1, t_2])$.

We say that a block b is checkpointed at slot t if t is the first slot an honest node sees b as checkpointed. Note that after GST, if b is checkpointed at slot t, then every honest node sees b as checkpointed by slot $t+\Delta$. Let $\max(\mathsf{GST}, \mathsf{GAT}) + T_{\mathrm{rec}} < t_1^* \leq t_2^* \leq ... \leq$ be the slots at which new blocks b_1^*, b_2^*, ... are checkpointed after $\max(\mathsf{GST}, \mathsf{GAT})$. To show that checkpointing a new block does not forfeit too many convergence opportunities, we follow the approach of [37] and divide time into two sets of intervals. Let $I_l := [t_l^* + \Delta, t_{l+1}^* - T_{\mathrm{rec}} - \Delta]$ and define $I := \cup_{l \geq 0} I_l$ as the union of the *inter-checkpoint intervals* I_l. (Recall the definition of T_{rec} from Lemma 1.) Using the definition of I, we can now proceed to prove the chain growth, quality and the common prefix properties.

C.1 Chain Growth

Definition 7 ([35, Section 3.2.1]). *Predicate* $\mathrm{growth}(\tau, k)$ *is satisfied if and only if for every slot* $t \leq T_{\max} - \tau$, $\min_{i,j}(|\mathsf{ch}_j^{t+\tau}| - |\mathsf{ch}_i^t|) \geq k$.

We use the same results given in [37] to lower bound the chain growth in terms of convergence opportunities that lie within the inter-checkpoint intervals.

Lemma 2 ([37, Lemma 5]). *Let s, t be two slots such that $s \in I$ and and $t \geq s + \Delta$. Let ch_i^s be a chain held by some honest node i at slot s. Then all honest nodes will hold a chain of length at least $|\mathsf{ch}_i^s|$ in slot t.*

Proof is (almost) the same as [37, Lemma 5] and uses Lemma 1.

Proof. Let i and j (potentially $i = j$) be honest nodes awake at slots s and $t \geq s + \Delta$ respectively. Since $s \in I$, there exists an l such that $s \in I_l$. Let b denote the last checkpoint block within ch_i^s. Since $s \geq \max(\mathsf{GST}, \mathsf{GAT})$, all honest nodes accept b as a checkpoint by slot t. Next, consider the following two cases: (i) b is the last checkpoint block in j's view by slot t. Then, $|\mathsf{ch}_j^t| \geq |\mathsf{ch}_i^s|$, as ch_i^s contains all of the checkpoints observed by j by slot t. (ii) j observes at least one new block become checkpointed by slot t. In this case, let b' denote the first block that becomes checkpointed in j's view after b within slot $t_{l'}^*$. $l' > l$. (In this case, $t \geq t_{l'}^*$ by definition.) Via Lemma 1 (recency property), b' must be on the checkpoint-respecting LC $\mathsf{ch}_k^{t'}$ held by an honest node k at some slot $t' \in [t_{l'}^* - T_{\mathrm{rec}}, t_{l'}^*]$, during which b' was not checkpointed yet in the view of any honest node. Thus, via case (i), $|\mathsf{ch}_k^{t'}| \geq |\mathsf{ch}_i^s|$ since $t' \geq t_{l'}^* - T_{\mathrm{rec}} \geq s + \Delta$ for $l' > l$.

Note that the length of the checkpoint-respecting LC held by any honest node at the time it observes b' become checkpointed must be at least $|\mathsf{ch}_k^{t'}| \geq |\mathsf{ch}_i^s|$. Hence, by induction, we can state that all honest nodes that observe at least one new checkpoint (after b) by slot t hold chains of length at least $|\mathsf{ch}_i^s|$ at slot t, implying that $|\mathsf{ch}_j^t| \geq |\mathsf{ch}_i^s|$. □

A useful corollary of Lemma 2 is given below:

Corollary 1. *All honest blocks produced at convergence opportunities within I have distinct heights.*

Proof. Suppose an honest block b is produced at height ℓ at a convergence opportunity t within I. Then, the honest producer of b holds a chain of length ℓ at slot $t \in S$. Via Lemma 2, all honest nodes will hold a chain of length at least ℓ at all slots $\geq t + \Delta$. Since the next convergence opportunity after t happens at some slot $\geq t + \Delta$, the next honest block will be at a height larger than ℓ. □

Lemma 3 ([37, Lemma 6]). *Consider a slot $s \in I$ and an honest node i awake at s such that $|\mathsf{ch}_i^s| = \ell$. (Alternatively, consider a slot t and an honest node i awake at t such that ch_i^t contains an honest block at height ℓ produced in some slot $s < t$.) Then, for any slot $t \geq s + 2\Delta$ and honest node j awake at slot t, $|\mathsf{ch}_j^t| \geq \ell + \mathrm{C}(I \cap [s+\Delta, t-\Delta])$.*

Proof is similar to [37, Lemma 6] and uses Lemma 2. A direct consequence of Lemma 3 is that $\mathrm{growth}(\tau, k)$ is satisfied if for any interval $[t_1, t_2]$ of length $t_2 - t_1 = \tau \geq 2\Delta$, $\mathrm{C}(I \cap [t_1 + \Delta, t_2 - \Delta]) \geq k$.

C.2 Chain Quality

Definition 8 ([35, Section 3.2.2]). *Predicate $\mathrm{quality}(\mu, k)$ is satisfied if and only if for every slot t and every honest node i awake at t, among any consecutive sequence of k blocks within ch_i^t, the fraction of blocks produced by honest nodes is at least μ.*

Lemma 4. *If $\mathrm{quality}(\mu, 1/\mu) = 0$, then there exist slots s, t such that $\mathrm{A}([s,t]) \geq 1/\mu$ and $\mathrm{A}([s,t]) \geq \mathrm{C}(I \cap [s+\Delta, t-\Delta]) - 1$.*

Proof. If $\mathrm{quality}(\mu, 1/\mu) = 0$, then there exists a slot t' and an honest node i awake at t' such that $\mathsf{ch}_i^{t'}$ contains a consecutive sequence of $1/\mu$ blocks $b_1, ..., b_{1/\mu}$ produced by the adversary. Let b_s^* denote the last honest block before b_1 within $\mathsf{ch}_i^{t'}$ and let ℓ_s and s respectively denote its height and the slot it was produced in. Similarly, let b_t^* denote the first honest block after $b_{1/\mu}$ within $\mathsf{ch}_i^{t'}$ and let ℓ_t and t respectively denote its height and the slot it was produced in. (Note that the genesis block can be taken as an honest block.) If there is no honest block following $b_{1/\mu}$ within $\mathsf{ch}_i^{t'}$, let $b_t^* = \mathsf{ch}_i^{t'}[-1]$, $\ell_t = |\mathsf{ch}_i^{t'}|$ and $t = t'$. In either case, there exists an honest node which holds a chain of length $\ell_t - 1$ at slot t and this chain contains an honest block at height ℓ_s produced in slot s.

Thus, via Lemma 3, we know that $\ell_t \geq \ell_s + \mathrm{C}(I \cap [s+\Delta, t-\Delta])$. Note that every block within $\mathsf{ch}_i^{t'}$ with height in (ℓ_s, ℓ_t) was produced by the adversary within the interval $[s,t]$, and lie on the same chain. Hence, $\mathrm{A}([s,t]) \geq \ell_t - \ell_s - 1$, where $\ell_t - \ell_s - 1 \geq \mathrm{C}(I \cap [s+\Delta, t-\Delta]) - 1$. Moreover, the blocks $b_1, ..., b_{1/\mu}$ were produced within the interval $[s,t]$ and lie on the same chain, implying that $\mathrm{A}([s,t]) \geq 1/\mu$. Hence, we can conclude that if $\mathrm{quality}(\mu, 1/\mu) = 0$, $\mathrm{A}([s,t]) \geq 1/\mu$ and $\mathrm{A}([s,t]) \geq \mathrm{C}(I \cap [s+\Delta, t-\Delta]) - 1$. □

C.3 Common Prefix

Definition 9 ([35, Section 3.2.3]). *Predicate* $\mathrm{prefix}(\tau)$ *is satisfied if and only if for all slots $s \leq t$ and honest nodes i, j such that i and j are awake at slots s and t respectively, prefix of ch_t^j consisting of blocks produced at slots $\leq t - \tau$ is a prefix of ch_s^i.*

To show the common prefix property in the context of checkpointed PoS protocols, we extend the definition of strong pivots in [35, Section 5.6.1] as shown below:

Definition 10. *A slot t is said to be a* checkpoint-strong pivot*, if for any $t_0 \leq t \leq t_1$, it holds that either* $\mathrm{A}([t_0, t_1]) < \mathrm{C}(I \cap [t_0+\Delta, t_1-\Delta])$ *or* $\mathrm{A}([t_0, t_1]) = 0$.

Observe that when we count the number of convergence opportunities for a checkpoint-strong pivot, we only take those that lie within the inter-checkpoint intervals. Intuitively, this is because the convergence opportunities that arrive during checkpoint intervals do not offer any guarantee of growth for the chains held by honest nodes. Conversely, as Corollary 1 states, all of the honest blocks that arrive at convergence opportunities within I have a unique height. Hence, by counting only the convergence opportunities in I, we can inherit all of the qualitative results presented in [35] about the prefix property. In this context, following proposition and lemma extend [35, Fact 4, Lemma 5]:

Proposition 5 (Unique Honest Blocks at Convergence Opportunities in I). *Let i and j be two honest nodes awake at slots r_1 and $r_2 \geq r_1$ respectively. If $\mathsf{ch}_i^{r_1}[\ell]$ and $\mathsf{ch}_j^{r_2}[\ell]$ are both honest blocks and there exists a convergence opportunity t^*, $t^* \in I$, such that an honest block b^* was produced at height ℓ, then, $\mathsf{ch}_i^{r_1}[\ell] = \mathsf{ch}_j^{r_2}[\ell] = b^*$.*

Proof. For the sake of contradiction, suppose $\mathsf{ch}_i^{r_1}[\ell]$ and $\mathsf{ch}_j^{r_2}[\ell]$ are both honest blocks at least one of which is different from b^*. Let k denote the honest producer of b^* such that $\mathsf{ch}_k^{t^*}[\ell] = b^*$. Without loss of generality, suppose $\mathsf{ch}_i^{r_1}[\ell] = b \neq b^*$, and let m and t denote the honest block producer and the production slot of b. As $b \neq b^*$, either $t < t^* - \Delta$ or $t > t^* + \Delta$. Now, if $t < t^* - \Delta$, either at least one honest node holds a checkpoint-respecting LC of length ℓ at time $t^* - \Delta$, or b conflicts with one of the blocks checkpointed before slot $t^* - \Delta$. In the first case, $|\mathsf{ch}_k^{t^{*-}}| \geq \ell$, which implies b^* could not have been produced at height ℓ, leading to a contradiction. In the latter case, no checkpoint-respecting LC of an honest node will contain b after slot t^*, which is a contradiction as $b \in \mathsf{ch}_i^{r_1}$. Conversely, if $t > t^* + \Delta$, via Lemma 2, $|\mathsf{ch}_m^{t^-}| \geq |\mathsf{ch}_k^{t^*}| \geq \ell$, which implies b could not have been produced at height ℓ, leading to a contradiction. Thus $b = b^*$ and $\mathsf{ch}_i^{r_1}[\ell] = \mathsf{ch}_j^{r_2}[\ell] = b^*$. □

Lemma 5. *Let i and j be two honest nodes awake at slots r_1 and $r_2 \geq r_1$ respectively. Let t be a checkpoint-strong pivot such that there is a convergence opportunity in $[t, r_1] \cap I$. Then, the last common block within $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ should have been produced in a slot $\geq t$.*

Proof. Since t is a checkpoint-strong pivot, there exist convergence opportunities $t'' \leq t \leq t'$ such that $t'', t' \in I$ and no adversary block is produced in the interval $[t'', t']$. (Note that if t is a convergence opportunity in I, $t'' = t' = t$.) In any case, t' is also a checkpoint-strong pivot. By the assumption that there is a convergence opportunity in $[t, r_1] \cap I$, we know that $t' \leq r_1$.

As $t' \in I$ is a convergence opportunity, there exists an honest block b^* produced at some height ℓ^* at slot t'. Next, we claim that since t' is a checkpoint-strong pivot, there cannot be an adversarial block within $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ at height ℓ^*. For the sake of contradiction, suppose there exists an adversary block b at height ℓ^* in a chain ch_m^s held by some honest node m at some slot $s \geq t' + \Delta$. Let b_1, produced at slot t_1 and height $\ell_1 < \ell^*$, denote the last honest block in the prefix of b within ch_m^s. If there is no such block, we take the genesis block as b_1. Similarly, let b_2, produced at slot t_2 and height $\ell_2 > \ell^*$, denote the first honest block in the suffix of b within ch_m^s. If there is no such block, we take the last block on ch_m^s as b_2. If b_2 is an honest block, by definition, all of the blocks on ch_m^s at heights (ℓ_1, ℓ_2) are adversarial (there is at least one such block b) and have been mined at distinct times within the interval (t_1, t_2). On the other hand, if b_2 is adversarial, all of the blocks on ch_m^s at heights $(\ell_1, \ell_2]$ are adversarial and have been mined at distinct times within the interval $(t_1, t_2]$. In this context, we first consider the case that b_2 is honest.

Observe that as t' is a convergence opportunity, either $t' \leq t_1 - \Delta$, $t' \in [t_1 + \Delta, t_2 - \Delta]$ or $t' \geq t_2 + \Delta$. First, if $t' \leq t_1 - \Delta$, via Lemma 2, all honest nodes will hold a chain of length at least ℓ^* by time t_1, implying that b_1 could not have been mined at height $\ell_1 < \ell^*$, which is a contradiction. Second, if $t' \geq t_2 + \Delta$, then the honest producer of b^* has already seen b_2; yet decided to produce a block at height $\ell^* < \ell_2$. However, this is only possible if b_2 conflicts with a checkpoint observed by the producer of b^* by time t'. However, this checkpoint would also be seen by node m by time $t' + \Delta \leq s$, implying that b_2 cannot be on ch_m^s at time s, which again is a contradiction. Consequently, the only possible scenario for t' is $t' \in [t_1 + \Delta, t_2 - \Delta]$.

Finally, suppose b_2 was produced by some honest node m'. As $\mathsf{ch}_{m'}^{t_2}$ contains an honest block, b_1, at height ℓ_1 produced at slot $t_1 < t_2 - \Delta$, via Lemma 3, $\ell_2 = |\mathsf{ch}_{m'}^{t_2}| > \ell_1 + \mathrm{C}(I \cap [t_1 + \Delta, t_2 - \Delta])$. Moreover, all of the blocks between b_1 and b_2 (perhaps including b_2) are adversarial and has been mined in the interval $(t_1, t_2]$ at distinct times, implying that $A([t_1, t_2]) \geq \mathrm{C}(I \cap [t_1 + \Delta, t_2 - \Delta])$. However, this is a contradiction with the fact that $t' \in [t_1 + \Delta, t_2 - \Delta]$ is a checkpoint-strong pivot.

Next, we consider the case that b_2 is an adversarial. From the analysis above, we know that $t' > t_1 + \Delta$. Moreover, every block on ch_m^s following b_1 are adversarial blocks. As the honestly-held chain ch_m^s, $s > t' + \Delta$, contains an honest block, b_1, at height ℓ_1 produced at slot $t_1 < s - \Delta$, via Lemma 3, $\ell_2 = |\mathsf{ch}_m^s| > \ell_1 + \mathrm{C}(I \cap [t_1 + \Delta, s - \Delta])$. Moreover, all of the blocks between b_1 and b_2 (including b_2) are adversarial and have been mined in the interval $(t_1, s]$ at distinct times, implying that $A([t_1, s]) \geq \mathrm{C}(I \cap [t_1 + \Delta, s - \Delta])$. However, this is a contradiction with the fact that $t' \in [t_1 + \Delta, s - \Delta]$ is a checkpoint-strong pivot.

Consequently, there cannot exist an adversary block b at height ℓ in any chain held by honest nodes after slot $t' + \Delta$, an assertion that includes the chains $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$. Then, as $\mathsf{ch}_i^{r_1}[\ell^*]$ and $\mathsf{ch}_j^{r_2}[\ell^*]$ are both honest blocks and t' is a convergence opportunity, via Proposition 5,

$$\mathsf{ch}_i^{r_1}[\ell^*] = \mathsf{ch}_j^{r_2}[\ell^*] = b^*. \tag{1}$$

Hence, the last common block within $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ must have been produced in a slot $\geq t' \geq t$. □

C.4 Probabilistic Analysis

To lower bound the number of convergence opportunities within I, we can use the following observation from [37, Proposition 4] which relies on Proposition 4 (gap property): If $t \geq s \geq \max(\mathsf{GST}, \mathsf{GAT})$,

$$\mathrm{C}([t_1 + \Delta, t_2 - \Delta] \cap I) \geq \mathrm{C}([t_1 + \Delta, t_2 - \Delta]) - (1 + (t_2 - t_1))(T_{\mathrm{rec}} + 2\Delta + 1)/T_{\mathrm{cp}} \tag{2}$$

Combining this expression with [35, Lemma 2, Corollary 2, Fact 2] yields the following proposition:

Proposition 6. *For any $\epsilon > 0$, there exists an ϵ' such that given $t_2 \geq t_1 \geq \max(\mathsf{GST}, \mathsf{GAT})$, $t \triangleq t_2 - t_1 \geq 2\Delta$,*

$$\mathbb{P}\left[\mathrm{C}([t_1 + \Delta, t_2 - \Delta] \cap I) \leq \left((1-\epsilon)(1 - 2pn\Delta)\alpha - \frac{T_{\mathrm{rec}} + 2\Delta + 1}{T_{\mathrm{cp}}}\right) t\right]$$
$$< \exp(-\epsilon'\alpha t)$$
$$\mathbb{P}[\mathrm{A}([t_1, t_2]) > (1+\epsilon)\beta t] < \exp(-\epsilon^2\beta t/3).$$

We also note that for any given $p < (n - 2f)/(2\Delta n(n - f))$ and sufficiently large T_{cp}, there exists a constant $\epsilon > 0$ such that

$$(1+\epsilon)\beta < (1-\epsilon)(1 - 2pn\Delta)\alpha - \frac{T_{\mathrm{rec}} + 2\Delta + 1}{T_{\mathrm{cp}}} \tag{3}$$

Next, we define T as the minimum slot $t \geq \max(\mathsf{GST}, \mathsf{GAT})$ such that $\mathrm{C}([0, t - \Delta] \cap I) = \mathrm{A}([0, t])$. In other words, T is an upper bound on the slot by which checkpoint-respecting LCs held by honest nodes would have caught up with the checkpoint-respecting LCs held by the adversary nodes. Thus, we can view T as the time Π_{lc} resets itself such that after T, it behaves like a checkpoint-respecting LC protocol that has just started running in a synchronous network. As long as T_{cp} is selected sufficiently large for equation 3 to hold, combining [32, Propositions 2,3,4] with Proposition 6, we can assert the following proposition bounding T:

Proposition 7. *There exists a constant* $\mathbf{C}$ *such that for any given security parameter* σ*, and* $\mathsf{GST}, \mathsf{GAT}$ *specified by* $(\mathcal{A}_{\mathrm{pda}}, \mathcal{Z}_{\mathrm{pda}})$*,* $T \leq \mathbf{C}(\max(\mathsf{GST}, \mathsf{GAT})+\sigma)$ *except with probability* $e^{-\Omega(\sigma)}$*.*

Using Proposition 7, we can complete the proof of Theorem 5:

Proof. Using Proposition 6 and Lemma 3, we can assert that for any given $\epsilon > 0$, $\mathrm{growth}(\sigma, k)$ is satisfied after $\max(\mathsf{GST}, \mathsf{GAT})$ except with probability $e^{-\Omega(\sigma)}$, where $k = g_0\sigma$ for

$$g_0 = (1-\epsilon)(1-2pn\Delta)\alpha - \frac{T_{\mathrm{rec}} + 2\Delta + 1}{T_{\mathrm{cp}}}. \tag{4}$$

Similarly, using Lemma 4, Proposition 6 and Proposition 7, we can assert that for any given $\epsilon > 0$, $\mathrm{quality}(\mu, 1/\mu)$ is satisfied after slot $\mathbf{C}(\max(\mathsf{GST}, \mathsf{GAT})+\sigma)$, except with probability $e^{-\Omega(\sigma)}$, where

$$\mu = \frac{(1-\epsilon)(1-2pn\Delta)\alpha - (1+\epsilon)\beta - (T_{\mathrm{rec}} + 2\Delta + 1)/T_{\mathrm{cp}}}{(1-2pn\Delta)\alpha - (T_{\mathrm{rec}} + 2\Delta + 1)/T_{\mathrm{cp}}}. \tag{5}$$

Finally, we know via Lemma 5 that checkpoint-strong pivots force convergence of the checkpoint-respecting LCs seen by all honest nodes. Hence, we can use [35, Theorem 7] to show that $\mathrm{prefix}(\sigma)$ is satisfied after slot $\mathbf{C}(\max(\mathsf{GST}, \mathsf{GAT})+\sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$. Then, using [35, Lemma 1], we conclude that Π_{lc} is secure with confirmation time $O(\sigma/g_0)$ after slot $\mathbf{C}(\max(\mathsf{GST}, \mathsf{GAT})+\sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$. □

Via Lemma 1, $T_{\mathrm{rec}} = \Delta + T_{\mathrm{to}} + T_{\mathrm{confirm}}$. On the other hand, T_{cp} should be chosen large enough for the inequality (3) to be satisfied. Thus, placing the definitions of α and β, setting $f = n/4$, $T_{\mathrm{to}} = 60$ seconds as in Section 5, and choosing $\epsilon = 0.1$ and $p = 0.8(n-2f)/(2\Delta n(n-f)) = 0.8/(3n\Delta)$, we can obtain inequality (3) from the following expression:

$$\frac{2T_{\mathrm{to}} + 3\Delta}{(1+\epsilon)pf - (1-\epsilon)(1-2pn\Delta)p(n-f)2T_{\mathrm{to}}} = 100(120+3\Delta)\Delta < T_{\mathrm{cp}} \tag{6}$$

Thus, for any given value of Δ, there exists a T_{cp} that satisfies inequality (3) for these set of parameters. Experimentation in Section 5 shows that for Δ approximately a few seconds, T_{cp} of 300 seconds is sufficiently large to ensure security under real network conditions.

C.5 Security Argument for Chia

While the sections above prove Theorem 2 for PoS, and by implication PoW protocols, security of Chia [12] does not immediately follow from the analysis of checkpoint-strong-pivots due to nothing-at-stake attacks [14], which enable the adversary to mine blocks on top of each existing block via independent Poisson processes. The first paper to show security for such protocols given the possibility of nothing-at-stake attacks is [14] which introduced a novel method called blocktree-partitioning. This method splits the overall blocktree into adversarial trees that build on a *fictitious honest tree* with a chain growth property analogous to the one in Appendix C.1. Thus, as in the case of convergence opportunities, we can once again count only the honest blocks that arrive within inter-checkpoint intervals I_l to provide a non-trivial lower bound on the growth of the fictitious honest tree in the context of checkpoint-respecting LCs. This lower bound follows from the fact that each honest block produced during such an interval I_l has the potential to contribute to the growth of honest chains just like in the case of original LC protocols. Using the modified definition for the fictitious honest tree, we can then prove [14, Theorem 3.2] that ties protocol security to the evolution of the fictitious honest tree for checkpoint-respecting LC protocols. Finally, the probabilistic analysis of [14, Section 4.2] goes through provided that $\beta < 1/e$ and the parameters p and $1/T_{\mathrm{cp}}$ are sufficiently small. Details of this analysis is left as future work.

D Experimental Evaluation Details

Implementation: Our prototype is implemented in the programming language Rust. A diagram of the different components and their interactions is provided in Figure 4. We use a longest chain protocol modified to respect latest checkpoints as Π_{lc}, with a permissioned block production lottery with winning probability p per node and per time slot of duration T_{slot}; and HotStuff[7] as Π_{bft}. Honest nodes pause HotStuff (including its timeouts) while waiting for the next checkpoint proposal. All communication (including HotStuff's) takes place in a broadcast fashion via `libp2p`'s `Gossipsub` protocol[8], mimicking Ethereum 2's network layer [1], to be able to scale to thousands of nodes. Thus, we assume that under normal conditions every message received by one honest node will be received by all honest nodes within some bounded delay. Since responsiveness is not so important for our checkpointing application and to avoid broadcasting quorum certificates, we use a variant of HotStuff where to ensure liveness the leader waits for the network delay bound before proposing a block. Our prototype does not implement the application logic of the beacon chain (such as validators joining and leaving, integration with shard chains and Ethereum 1, etc.) which can be realized on top of consensus in the same way as currently done in Ethereum 2, and our prototype does not use any orthogonal techniques

[7] We used this Rust implementation: https://github.com/asonnino/hotstuff

[8] We used this Rust implementation: https://github.com/libp2p/rust-libp2p

to reduce bandwidth by constant factors (such as signature aggregation, short signature schemes, compression of network communication, etc.) which are not fundamental to the consensus problem.

Choice of parameters: We chose the parameters of our protocol in the experiments to match the parameters of Ethereum 2's beacon chain. The beacon chain has $C = 32$ slots per epoch and $m = 128$ validators per slot, for a total of $n = 4096$ validators (per epoch), which is the approximate number of nodes that we run our experiments with. To match the block inter-arrival time (*i.e.*, the duration of one slot) of 12 s in the beacon chain, we set $p = 1/n$ and account for the probability of no node winning the block production lottery and choose $T_{\mathrm{slot}} = 7.5\,\mathrm{s}$. We also match the block payload size of 22 KBytes. In terms of Π_{lc}, we chose $k_{\mathrm{cp}} = 6$ so that a checkpoint proposal by an honest leader is reasonably likely (although not 'guaranteed') to be accepted by other honest nodes, and $k = 6$ for a swift 72 s average confirmation delay of $\mathsf{LOG}_{\mathrm{da}}$. Note that k_{cp} should be the same for all nodes, while each client can choose an individual k to trade off latency and confirmation error probability of $\mathsf{LOG}_{\mathrm{da}}$. To leave enough time for message propagation, we set the HotStuff timeout $T_{\mathrm{hs}} = 20\,\mathrm{s}$. To avoid HotStuff timeouts escalating into checkpoint timeouts for honest leaders, we set $T_{\mathrm{to}} = 1\,\mathrm{min}$. Finally, to target 5× improvement in average $\mathsf{LOG}_{\mathrm{acc}}$ latency over Gasper (*cf.* Figure 7), we set $T_{\mathrm{cp}} = 5\,\mathrm{min}$.

Experiment setup: Adversarial nodes in the experiment aim to stall consensus as much as possible. Thus, they do not share a proposal when elected leader in Π_{bft} or Π_{acc}, so that honest nodes have to wait for a timeout before they can move on, and they mine selfishly [16] in Π_{lc} to reduce honest chain growth. We ran our prototype (a) with no adversary (Figure 2(l)), and (b) with $\beta = 25\%$ adversary (Figure 2(r)), each for 2500 s on five AWS EC2 `c5a.8xlarge` instances in each of ten AWS regions[9], with 82 nodes per machine, for a total of 4100 nodes. Each honest (adversarial) node connected to 15 (15 honest and 15 adversarial) randomly selected peers for the peer-to-peer gossip network.

Observations: We observe that both without faults (Figure 2(l)) as well as under the 25%-attack (Figure 2(r)) the available full ledger (—) shows steady growth, albeit under attack at a reduced rate due to selfish mining. In both cases, the accountable prefix ledger (—) periodically catches up with the available ledger. Timeouts cause occasional but overall minor delays of the catch-up.

In terms of bandwidth (reported in Figures 6 and 5 *for an exemplary AWS instance*, *i.e.*, for 82 nodes), we observe a distinct spiky pattern with frequent small spikes corresponding to the propagation of Π_{lc} blocks and infrequent wide spikes corresponding to the propagation of checkpoint votes and Π_{bft} blocks and votes as part of checkpointing. There is more traffic under attack than without attack (per node: avg. 78 KB/s peak 1.5 MB/s vs avg. 56 KB/s peak

[9] `eu-north-1`, `eu-west-3`, `ap-south-1`, `ap-northeast-1`, `ap-southeast-2`, `sa-east-1`, `ca-central-1`, `us-west-1`, `us-east-2`, `us-east-1`

1.34 MB/s), since timeouts due to adversarial non-action lead to additional iterations of checkpointing and HotStuff. In either case, the bandwidth requirement does not pose a severe limitation to participation even using consumer-grade Internet connectivity.

Bandwidth requirement and accountable ledger latency: We examine the tradeoff between the average number of votes communicated per time (as a surrogate for average required bandwidth, to avoid confounding factors such as compression or signature aggregation) and the average latency of the accountable ledger (see Figure 7), for varying number n of nodes and varying parameters C and T_{cp} for Gasper and our protocol, respectively, under ideal operation, *i.e.*, $\beta = 0, \Delta = 0$. In this case, Gasper transmits $2 \cdot \frac{n}{C}$ votes per 12 s (per slot, each committee member issues an LMD GHOST and a Casper FFG vote), while our protocol transmits $5 \cdot n$ votes per T_{cp} time (broadcast checkpoint votes, checkpoint votes in HotStuff proposal, three rounds of HotStuff voting for confirmation). A transaction takes on average $\frac{1}{2} + 2$ epochs to enter into the accountable ledger for Gasper (wait until end of ongoing epoch, then two epochs to reach finality), and $k_{\mathrm{cp}} \cdot 12\,\mathrm{s} + \frac{1}{2} \cdot T_{\mathrm{cp}}$ time to enter into the accountable ledger for our protocol (k_{cp}-deep to enter checkpoint proposal, then wait until next checkpoint iteration). As evident from Figure 7, our protocol offers slightly improved latency at comparable bandwidth, or comparable bandwidth and latency but for a larger number of nodes. Let us point out that, as currently implemented, nodes in our protocol broadcast votes at the highest throughput feasible once T_{cp} has expired, so that the resulting traffic pattern is more bursty than that produced by Gasper, where voting is taking place throughout each epoch. However, Figure 7 also corroborates that even if voting after T_{cp} was artificially rate-limited, bandwidth and latency comparable to Gasper can be achieved.

Note that the gross bandwidth measured in Figures 5 and 6 is roughly 6× the bandwidth estimate based on the number of votes per time. This is largely due to two factors: *1)* We have not optimized HotStuff in our prototype to remove quorum certificates from blocks (although we may do so due to the broadcast nature of the gossip network, as discussed in the earlier 'Implementation' paragraph), *2)* the amplification factor that comes with nodes flooding every new message to all their peers in the gossip network.

Scaling the number of nodes for fixed accountable ledger latency: To scale the number of nodes for a fixed accountable ledger latency (and hence fixed C and T_{cp}, respectively), both Gasper and our protocol need to increase their vote bandwidth proportionally. However, the attack described in [30] suggests that even without adversarial but with merely random network delay, Gasper is susceptible to balancing by an adversary controlling $O(\sqrt{C/n})$ of nodes. Thus, for fixed C, the relative adversarial resilience decreases (to 0) as n increases (to ∞).

Improvements: Obvious improvements would be to customize the block proposal generation of HotStuff to account for the semantics of votes and the state of the checkpointing protocol, *e.g.*, do not propose votes from past checkpoint iterations, delay proposals when there are no pending votes, propose blocks with the minimum set of votes to reach a new checkpoint decision, etc.

E Helper Functions for Algorithms 1 and 2

- PerformBookkeeping: Macro for bookkeeping in Alg. 1 of checkpoint decisions and checkpoint proposals from checkpoint leaders in charge
- CpLeaderOfIter(c): Returns randomly selected publicly verifiable unique leader $\mathrm{L}^{(c)}$ of checkpoint iteration c
- Broadcast(...): Broadcasts checkpoint proposal to other nodes
- GetCurrProposalTip(): Returns k_{cp}-deep block on node's checkpoint-respecting LC from Π_{lc}
- IsValidProposal(b): Checks whether block b is consistent with k_{cp}-deep block on node's checkpoint-respecting LC as obtained from Π_{lc}
- SubmitVote(v): Inputs vote v as payload to Π_{bft} for ordering
- GetNextVerifiedVoteFromBft(): Retrieves next vote with valid signature from the output ledger $\mathsf{LOG}_{\mathrm{bft}}$ as ordered by Π_{bft}
- OutputCp(...): Alg. 1 outputs a new checkpoint decision (used as input in Alg. 2 and to determine checkpoint-respecting LC in Π_{lc})

F Proof-of-Work and Proof-of-Space

We have so far focused on accountability gadgets built for permissioned LC protocols. We conclude with an outlook on accountability gadgets for permissionless LC protocols such as those based on Proof-of-Work (PoW), *e.g.*, Bitcoin [26], or those based on Proof-of-Space (PoSpace), *e.g.*, Chia [12]. In such constructions, nodes fulfill two different roles: *miners* control a unit of a rate-limiting resource, *e.g.*, compute power or storage space, and are responsible for extending the permissionless checkpoint-respecting LC; and *validators* with unique cryptographic identities are responsible for providing accountability. Security of Π_{lc} is primarily maintained by miners, security of Π_{acc} and the associated protocol Π_{bft} are primarily maintained by validators. While miners are free to participate dynamically, validators are expected to be always present to provide accountability.

In the following, let β be the fraction of online rate-limiting resource controlled by adversarial miners, and β^* be a quantity depending on the underlying Π_{lc} protocol: $\beta^* = 1/2$ for Bitcoin and $\beta^* = 1/e$ for Chia, our two examples. Then, the accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ and the available ledger $\mathsf{LOG}_{\mathrm{da}}$ satisfy:

Consider a network with validators (with n unique cryptographic identities) and miners (at least one of which is honest and awake), with the properties discussed above. The network may partition, and the miners may come online and go offline subject to the constraints below. Then, for any $f \leq \lceil n/2 \rceil$:

1. (**P1: Accountability**) The accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ provides accountable safety resilience $(n-2f+2)$ at all times, and is live after network partition, *if $\beta < \beta^*$ holds*, and if the number of honest validators is greater than $(n-f)$.

2. (**P2: Dynamic Availability**) The available ledger $\mathsf{LOG}_{\mathrm{da}}$ is guaranteed to be safe after network partition and live at all times, if $\beta < \beta^*$ holds, and *if the number of adversarial validators is at most* $(n - f)$.

Again, $\mathsf{LOG}_{\mathrm{acc}}$ is always a prefix of $\mathsf{LOG}_{\mathrm{da}}$ by construction.

Observe that the requirements of having greater than $(n - f)$ honest nodes under **P1** and having $\beta < \beta^*$ under **P2** are analogous to the permissioned case formulated in Theorem 2. However, accountability gadgets for permissionless LC protocols have further requirements. First, under **P1**, liveness of $\mathsf{LOG}_{\mathrm{acc}}$ requires $\beta < \beta^*$. Otherwise, the adversarial miners might stall Π_{lc} and hence $\mathsf{LOG}_{\mathrm{da}}$ might not become live after network partition, which was argued to be a necessary condition for liveness of $\mathsf{LOG}_{\mathrm{acc}}$ in Section 4.2. Second, under **P2**, security of $\mathsf{LOG}_{\mathrm{da}}$ requires the number of adversarial validators to be bounded by $(n - f)$. Otherwise, if the number of adversarial validators was greater than $(n - f)$, it would be possible for a block that is not on a checkpoint-respecting LC held by any honest nodes to become checkpointed solely by adversarial votes (l. 9 of Alg. 2), thus causing safety violations on $\mathsf{LOG}_{\mathrm{da}}$.

For Bitcoin, the security proof of Appendices B and C applies. For Chia, security can be proven via an extension of *blocktree partitioning* [14] to checkpoint-respecting LCs. Details in Appendix C.5. Finally, note that the reduced threshold $\beta^* = 1/e$ for Chia as the permissionless LC protocol is due to nothing-at-stake attacks [14], since randomness is updated per block in Chia. Other embodiments of PoSpace may provide different β^*.

G Proof of Non-Accountability of Checkpointed LC

In this section, we show that the checkpointed longest chain protocol presented in [37] does not provide accountable safety. The checkpointing protocol used by [37] is a slight modification of the Algorand BA protocol from [9]. Thus, the attack below on the accountability of the Algorand BA in [37] is very similar to the one described for Algorand BBA* [10] in [39, Appendix C.3].

Theorem 6. *Algorand BA [9] does not provide accountable safety.*

Proof. For the sake of contradiction, suppose there exists an adjudication function $\mathcal{J}$ that can identify at least one adversary node when there is a safety violation, and never identifies an honest node. Let $n = 3f + 1$ denote the total number of nodes among which f nodes are controlled by a Byzantine adversary. Note that Algorand BA requires each node i to hold a *starting value* st_i^p (that is distinct from the input values) for each period p of the protocol. Starting values are set to $st_i^1 = \perp$ for period $p = 1$. Each period consists of 4 steps and an optional 5-th step.

Consider three disjoint sets of nodes P, Q and R, where $|P| = f - 1$ and $|Q| = |R| = f + 1$. We next construct two worlds each with a different set of Byzantine nodes.

World 1: Nodes in R are controlled by the adversary. Suppose that the nodes in P and Q start with the input bits 0 and 1 respectively, and a node from

P is elected leader at period $p = 1$. Then, the following steps are executed by Algorand BA.

- Period 1, Step 1: Leader proposes its input, 0.
- Period 1, Step 2: Every node soft-votes the value 0 proposed by the leader. Adversary nodes also soft-vote 0 and share their votes with all honest nodes.
- Period 1, Step 3: Nodes in P and Q see more than $2f+1$ soft-votes for 0, thus they cert-vote for 0. Nodes in R also cert-vote for 0, but send their cert-votes only to the nodes in P.
- Period 1, Step 4: Nodes in P receive $3f+1 > 2f+1$ cert-votes, thus terminate using the halting condition and output 0. Nodes in Q receive only $|P|+|Q| = 2f$ cert-votes, thus are not able to certify any value. Nodes in R pretend as if they do not receive any cert-votes from the nodes in Q, thus are not able to certify any value either. Hence, nodes in Q and R go to the period's first finishing step. They all next-vote their starting values st_i^1, which is set to $\perp$ for period $p = 1$. Note that the nodes in R send their next-votes to the nodes in Q.
- Period 2, Step 1: Since the nodes in Q and R observe a total of $2f+1$ $\perp$-next-votes, they do not go to the second finishing step of period 1 and instead jump to step 1 of period $p = 2$, with $st_i^2 = \perp$. Suppose a node from Q is elected leader at period $p = 2$. It proposes its input 1.
- Period 2, Step 2: Nodes in Q and R soft-vote the value 1 proposed by the leader.
- Period 2, Step 3: Nodes in Q and R see more than $2f + 1$ soft-votes for 1, thus they cert-vote for 1.
- Period 2 (Halting Condition): Nodes in Q and R terminate using the halting condition and output 1.

Since the honest nodes in P and Q outputted different values, there is a safety violation, upon which all of the nodes send their evidences to the adjudication function $\mathcal{J}$. Evidences sent by the nodes in Q and R state that they did not hear from the nodes in R and Q respectively in step 3 of period 1. By assumption, $\mathcal{J}$ identifies at least one node from the set R as an adversarial node.

World 2: This world is identical to World 1 except that

- Nodes in Q are adversarial and the nodes in R are honest.
- Nodes in the set Q behave exactly like the nodes in R behaved in World 1, i.e. the nodes in Q do not send any cert-votes to the nodes in R in step 3 of period 1 and ignore their votes at the beginning of step 4 of period 1.

Thus, again the honest nodes in P and Q output different values, upon which all of the nodes send their evidences to $\mathcal{J}$. As worlds 1 and 2 are indistinguishable in the perspective of $\mathcal{J}$, it again identifies at least one node from the set R as an adversary node with non-negligible probability. However, this is a contradiction with the definition of $\mathcal{J}$ as R consists of honest nodes in world 2. □

H Example Protocols Framed in Model of Section 2

To illustrate the model in Section 2, consider a client that queries nodes running a Nakamoto-style longest chain protocol under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$ at some time t. Suppose $\beta < 1/2$. The transcript T_i^t held by node i at time t consists of the blocks received by node i by time t. Given the transcript T_i^t, $\mathcal{W}$ outputs as evidence the longest chain implied by T_i^t. Upon collecting evidences from a subset S of awake nodes with at least one honest node, a client calls $\mathcal{C}$ which selects the longest chain in the set $\{\mathsf{w}_i^t\}_{i \in S}$ and outputs the k-deep prefix of that longest chain as the ledger.

We can also consider propose-and-vote-style BFT protocols such as HotStuff, LibraBFT, Streamlet and PBFT [42,25,8,7] with $n = 3f + 1$ nodes under $(\mathcal{A}_{\mathrm{p}}, \mathcal{Z}_{\mathrm{p}})$. In this case, the transcript T_i^t held by a node i at time t consists of all received messages such as proposals and votes. Given T_i^t, $\mathcal{W}$ outputs as evidence a sequence of proposals with votes attesting to them. Upon collecting evidences from a subset S of nodes containing at least one honest node, a client calls $\mathcal{C}$, which outputs the largest possible sequence of proposals that can be confirmed given the votes attesting to them. The confirmation rule typically requires votes from $n - f + 1$ nodes on consecutive proposals to guarantee safety which follows from a quorum intersection argument. Liveness ensues from the fact that the honest evidence within S includes all of the confirmed proposals submitted by honest nodes. Existence of an honest evidence in S is typically enforced by collecting evidences from at least $f + 1$ nodes.