The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets

Joachim Neu
jneu@stanford.edu

Ertem Nusret Tas
nusret@stanford.edu

David Tse
dntse@stanford.edu

ABSTRACT

Byzantine fault tolerant (BFT) consensus protocols are traditionally developed to support reliable distributed computing. For applications where the protocol participants are economic agents, recent works highlighted the importance of *accountability*: the ability to identify participants who provably violate the protocol. We propose to evaluate the security of an accountable protocol in terms of its liveness resilience, the minimum number of Byzantine nodes when liveness is violated, and its accountable safety resilience, the minimum number of accountable Byzantine nodes when safety is violated. We characterize the optimal tradeoffs between these two resiliences in different network environments, and 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 an 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 4000 nodes show that the protocol can achieve the required scalability and has better latency than the current solution Gasper, while having the advantage of being provably secure. To contrast, we demonstrate a new attack on Gasper.

1 INTRODUCTION

1.1 Accountability

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 such a context, the security of consensus protocols is naturally measured by its *resilience*: the minimum number of Byzantine protocol-violating nodes needed to cause a loss of safety or liveness. In modern applications such as cryptocurrencies and decentralized applications platforms, consensus nodes are no longer just disinterested computing devices but are agents acting based on economic and other incentives. 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 in a provable way. This point of view is advocated by Buterin and Griffith [6] in the context of their effort to add accountability (among other things) to Ethereum's Proof-of-Work (PoW) longest chain protocol, and is also central to the design of Gasper [7], 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 of the stake of protocol-violating agents. Other protocols that are designed to provide accountability include Polygraph [13] and GRANDPA [40]. A recent comprehensive work [25] shows that accountability can also be added on top of some but not all existing Byzantine fault tolerant (BFT) protocols.

Given the importance of accountability, a key question is how the traditional definition of consensus protocol security needs to be modified to reflect its accountability?

1.2 Accountable Security

Defining an appropriate notion of *accountable security* of a protocol is the first contribution of this work.

As a starting point, we observe there is an inherent asymmetry between safety and liveness as far as accountability is concerned. While safety violations such as due to double-voting can be caught in a provable way, liveness violations such as transaction censoring cannot be. Indeed, the goal in [6] is to provide accountable *safety*. Hence, to set the stage to incorporate accountability, it is helpful to split the single metric of resilience into two individual metrics: *safety resilience* and *liveness resilience*. Safety resilience is the minimum number of Byzantine faults to cause a safety violation. Liveness resilience is the minimum number of Byzantine faults to cause a liveness violation. Classic resilience is simply the minimum of the two.

To capture accountable security, the notion of liveness resilience remains the same, but safety resilience should be strengthened to a notion of *accountable safety resilience*: the minimum number of faults that can be found *accountable* when there is a safety violation. More precisely, a protocol has an accountable safety resilience of f means that in the case of a safety violation, at least f nodes can be held accountable in a provable manner. By definition, the accountable safety resilience of a protocol cannot be larger than its safety resilience.

Splitting the traditional notion of resilience into safety and liveness resiliences, although usually not explicitly done, is implicit in many previous works in the BFT literature. Indeed, one can think of the design objective of a consensus protocol as achieving a good tradeoff between safety and liveness resiliences. For example, increasing the threshold of the quorum, a central concept in many BFT protocols, increases the protocol's safety resilience while decreases its liveness resilience. This separate treatment of safety and liveness resiliences is recently formalized in [29] through the notion of alive-but-corrupt faults. Indeed, results in the literature on

The authors contributed equally and are listed alphabetically.

optimal resilience achievable in a given network environment (synchronous, partially synchronous, etc.) can be refined into results on the optimal tradeoffs achievable between safety and liveness resiliences.

In applications with an economic context, treating safety and liveness resiliences separately makes sense, because the mechanisms of attacking safety are different from the mechanisms of attacking liveness and therefore the costs to the attacker are also different. Trading off the two resiliences allows the protocol designer to maximize the cost to the attacker. In this light, shifting the attention from safety resilience to accountable safety resilience makes sense under the assumption that the dominant cost to the attacker is the cost from being punished, from slashing of the stake for example.

A natural question then is: given a network environment, what is the optimal tradeoff achievable between *accountable safety resilience* and liveness resilience by any protocol?

1.3 Optimal Accountable-Safety vs Liveness Tradeoffs

Our second contribution is the characterization of the optimal tradeoffs between accountable safety resilience and liveness resilience in three network environments: a) **synchronous**, where all nodes are online and all messages are delivered between honest nodes within a known delay bound; b) **partially synchronous**, where all nodes are online but messages suffer arbitrary delays before a Global Stabilization Time (to model network partition), after which the network becomes synchronous; c) **dynamic participation**, where the number of nodes online is varying but messages delivered between online honest nodes have a known delay bound. The results are shown in Figure 1 (the lower part) and compared to the optimal tradeoff between (traditional) safety and liveness resiliences (the upper part).

The point in each of the optimal tradeoff regions which maximizes the traditional single resilience metric is the point on the boundary where the safety and liveness resiliences are the same. In general, one may want to operate at a different point on the boundary, reflecting the different costs of attacking safety versus liveness.

Several interesting conclusions regarding Figure 1:

- As is well known, stronger security guarantees can be provided in the synchronous environment than in the partially synchronous environment. This is reflected in an optimal tradeoff between (traditional) safety and liveness resiliences which is better in the synchronous environment than in the partially synchronous environment. In contrast, the optimal tradeoff between *accountable safety resilience* and liveness resilience is the same in the two environments. Thus the synchrony assumption does not improve accountable security. This is related to an impossibility result in [25], which we discuss in Section 1.6.
- Many protocols achieve the optimal tradeoffs in synchronous and partially synchronous environments. For example, Polygraph [13], HotStuff [41] and Streamlet [10] achieve the optimal tradeoff in the partially synchronous environment. Sync HotStuff [2] and Sync Streamlet [10] achieve the optimal tradeoff in the synchronous environment (see Appendix E). In fact, in all these cases, the *same* protocol simultaneously maximizes the safety resilience and the accountable safety resilience for a given liveness resilience. That means one can simultaneously have optimal accountable security without sacrificing traditional security.
- In contrast to the synchronous and partially synchronous environments, no accountability can be supported under dynamic participation: the accountable safety resilience is zero for any positive liveness resilience (Figure 1 (vi)). Protocols like Ouroboros [16] and SnowWhite [15] can tolerate dynamic participation and are safe and live under a resilience of 50%, thus achieving the optimal point in Figure 1 (iii). But they cannot be made accountable.

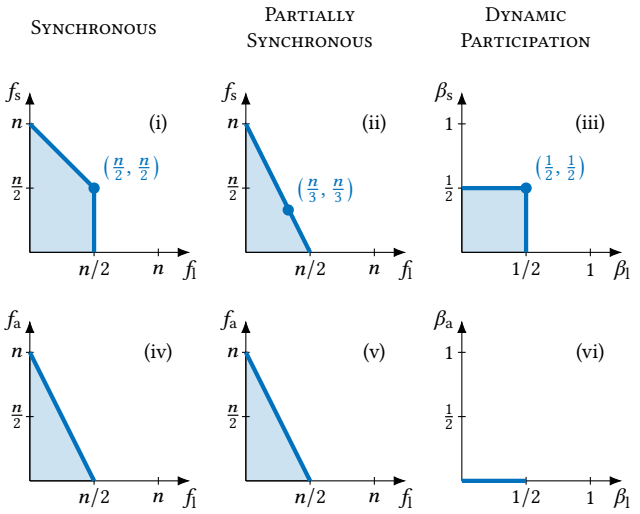

Figure 1: Above: optimal tradeoffs between (traditional) safety and liveness resilience in three environments. Below: optimal tradeoffs between accountable safety resilience and liveness resiliences. In each environment, the optimal tradeoff is described by a region of feasible resilience points, each point achievable by some protocol; points outside the region cannot be achieved by any protocol. The classic single resilience metric is maximized at the point on the region boundary where the two resiliences are the same. In the synchronous and partially synchronous environments, resilences are expressed in terms of the number of Byzantine nodes; in the dynamic participation environment, resiliences are expressed in terms of the number of Byzantine nodes as a fraction of the total online nodes.

1.4 Availability-Accountability Dilemma

In public permissionless blockchains like Bitcoin and Ethereum, dynamic participation is a central feature. In Bitcoin, for example, the total hash rate varies over many orders of magnitude over the years. Yet, the blockchains remain continuously *available*, *i.e.*, live. Our results say that it is impossible to support accountability for such *dynamically available* protocols, *i.e.*, protocols that are live under dynamic participation. We call this the *availability-accountability dilemma.*

The works Casper [6] and Gasper [7] provide some hints on how to get around this dilemma. One way to interpret these works is that they aim to design an *accountability gadget* to provide a *checkpointing* mechanism on top of a dynamically available chain, so that the available chain can continue to grow while the ledger up to the latest checkpoint is accountable. While the availability-accountability dilemma says that a single ledger cannot be made dynamically available and accountable at all times, these works can be interpreted as aiming to generate *two* ledgers: 1) a full ledger $\mathsf{LOG}_{\mathrm{da}}$, which is dynamically available, 2) an accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$, which is the checkpointed prefix of the full ledger. Unfortunately these works have several significant limitations:

- They lack a formulation to specify what security properties they want to achieve for these two ledgers. In particular, it is not clear how closely the checkpointed ledger can track the available ledger.
- Liveness attacks have been discovered for these protocols [31, 33, 33, 35]. (To reinforce this point, in Appendix I we present a new practical attack which dispenses with the adversarial network delay employed by earlier attacks.)

1.5 Resolution via Accountability Gadgets

The third contribution of this work is the design and implementation of 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**) For any $f < n/2$, the accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ can provide an accountable safety resilience of $n-2f+2$ at all times, and it is live after the partition heals and greater than $n - f$ honest nodes come online.
(2) (**P2: Dynamic Availability**) The available ledger $\mathsf{LOG}_{\mathrm{da}}$ is guaranteed to be safe after network partition and live at all times, provided that fewer than $1/2$ of the online nodes are adversarial.

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.

The achieved resiliences are optimal. This can be seen by comparing this result with Figure 1 (iii) and (v). The checkpointed ledger $\mathsf{LOG}_{\mathrm{acc}}$ cannot achieve a better tradeoff between accountable safety resilience and liveness resilience than the tradeoff in (v); it in fact achieves exactly the same tradeoff. The dynamically available ledger $\mathsf{LOG}_{\mathrm{da}}$ cannot achieve a better resilience than the $(1/2, 1/2)$ point in (iii); the ledger in fact achieves it. Moreover, even if the network were synchronous at all times, no protocol could have generated an accountable ledger with better resilience, in light of Figure 1 (iv). So we are getting partition-tolerance for free, even though accountability is the goal.

The accountability gadget construction is shown in Figure 2. 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 accountable prefix ledger $\mathsf{LOG}_{\mathrm{acc}}$. Consistency of blocks confirmed by Π_{lc} and future checkpoint proposals with established checkpoints is ensured throughout.

Since there are many accountable BFT protocols [25], we have a lot of implementation choices. Due to its maturity and the avail-

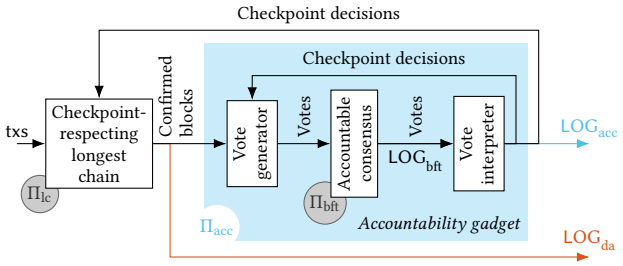

Figure 2: We construct an accountability gadget Π_{acc} from any accountable BFT protocol Π_{bft} and apply it to a longest-chain-type protocol Π_{lc} as follows: The fork choice rule of Π_{lc} is modified to respect the latest checkpoint decision. Blocks confirmed by Π_{lc} are output as available ledger $\mathsf{LOG}_{\mathrm{da}}$. 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.

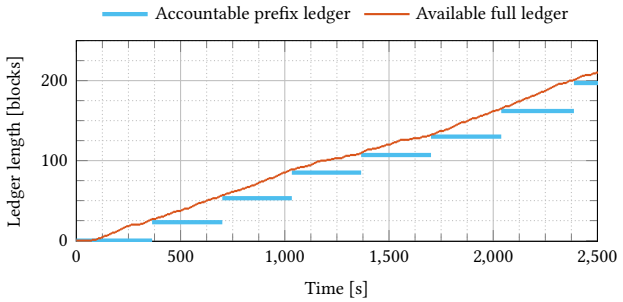

Figure 3: Ledger dynamics of a longest chain protocol outfitted with our accountability gadget based on HotStuff, measured with 4,100 nodes distributed around the world. The available full ledger grows steadily. The accountable prefix periodically catches up whenever a new block is checkpointed. (Here, no attack; for attack, *cf.* Figure 6.)

ability 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 [41]. 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 4000 participants (see Figure 3). 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.

1.6 Related Works

1.6.1 Accountability. Accountability in distributed protocols has been studied in earlier works [23, 24]. [23] designed a system, PeerReview, which detects faults. [24] classifies faults into different types and studies their detectability. Casper [6] 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 [13] 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 $n/3$ nodes can be held accountable. In our formulation, this corresponds to achieving the point $(n/3, n/3)$ on Figure 1 (v). [38] builds upon [13] to create a blockchain which can exclude Byzantine nodes that were found to have provably 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. [25] 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 [6] 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. Despite these differences in formulation and in scope, we are able to adopt the proof of the impossibility result Theorem B.1 in [25], because at the heart of it, that theorem is really about the tradeoff between liveness and accountable safety resiliences, although not stated as such.

1.6.2 Availability-Finality Dilemma and Finality Gadgets. The *availability-finality dilemma* [22, 27, 35] 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. The first resolution of the availability-finality dilemma is the class of snap-and-chat protocols [35], 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 [34] that the resulting snap-and-chat protocol would also provide a resolution to the availability-accountability dilemma. On the other hand, checkpointed longest chain [39], another resolution of the availability-finality dilemma, is not accountable, as shown in Appendix F.

The accountability gadget we designed combines elements from snap-and-chat protocols and from the checkpointed longest chain. A strength of snap-and-chat protocols is its black box nature which gives it a flexibility to provide additional features. A drawback is that the protocol may reorder the blocks in the longest chain protocol to form the final ledger [34]. This means that when a proposer proposes a block on the longest chain, it cannot predict the ledger state and check the validity of the transactions by just looking at the earlier blocks in the longest chain. This lack of *predictive validity* opens the protocol to spamming and limits the use of standard techniques to support light clients and sharding. Checkpointed longest chain builds upon a line of work called finality gadgets [6, 7, 18, 40] and overcomes this limitation of snap-and-chat protocols because the longest chain protocol is modified to respect the checkpoints. However, checkpointed longest chain's finality gadget is not black box but specifically uses Algorand BA [11], which is not accountable [25]. Our accountability gadget solution builds on the checkpointed longest chain but, like snap-and-chat protocols, allows the use of any BFT protocol as a black box. When an accountable BFT protocol like HotStuff is used, the checkpointed ledger is guaranteed to be accountable.

1.7 Outline

The remainder of this paper is structured as follows: First, we introduce the notation and model for a formal treatment of the tradeoffs among safety, liveness and accountable safety resiliences in Section 2. Then, Section 3 presents the proof of the availability-accountability dilemma and formalizes the tradeoffs visualized in Figure 1. Section 4 elaborates on the accountability gadgets introduced in Section 1.5 and argues for their security. We discuss details of a prototype implementation and experimental performance results in Section 5. Finally, we conclude with a generalization of accountability gadgets to Proof-of-Work and Proof-of-Space longest chain protocols in Section 6.

2 MODEL

We first give an overview of the client-server model for state machine replication (SMR) protocols and introduce the notation that will be used in subsequent proofs. In the classical SMR formulation, *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* C to this set of evidences to obtain the ledger: $\mathsf{LOG} \triangleq C(\{\mathsf{w}_i^t\}_{i\in S})$. Protocols typically require to query a subset S containing at least one honest node.

Environment: We assume that the transactions are input to the nodes by the environment $\mathcal{Z}$. There are in total n nodes numbered from 1 thru n. There exists a public-key infrastructure and each node is equipped with a unique cryptographic identity. There is a random oracle, serving as a common source of randomness for the protocols. Time is slotted and the nodes have synchronized 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.

Sleeping: To model dynamic participation, we adopt the concept of *sleepiness* from [37]. In the setting with dynamic participation, $\mathcal{A}$ chooses, for every time slot and node, whether an honest node is *awake* (*i.e.*, online) or *asleep* (*i.e.*, offline) in that slot. $\mathcal{Z}$ then wakes up or puts nodes to sleep following the schedule determined by $\mathcal{A}$. An honest node that is awake in a slot executes the protocol faithfully in that slot. An honest node that is asleep in a slot does not execute the protocol in that slot, 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 value of the fraction of adversarial nodes over the total number of awake nodes throughout the execution of the protocol.

Networking: Nodes can send each other messages, which arrive with a certain delay controlled by the adversary, subject to constraints elaborated below.

Network Environments: Given the definitions above, we provide three sets of assumptions on the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ to model a synchronous network, a partially synchronous network and a synchronous network with dynamic participation, respectively. These assumptions are expressed as the $(\mathcal{A}, \mathcal{Z})$ tuples:

$(\mathcal{A}_\mathrm{s}, \mathcal{Z}_\mathrm{s})$ formalizes the model of a synchronous network, where the adversary corrupts f nodes, and all of the n nodes are awake throughout the execution. At all times, adversary is required to deliver all messages sent between honest nodes in at most Δ slots.

$(\mathcal{A}_\mathrm{p}, \mathcal{Z}_\mathrm{p})$ formalizes a partially synchronous network, where the adversary corrupts f nodes, and all of the n nodes are awake throughout the execution. 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 in at most 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.

$(\mathcal{A}_\mathrm{da}, \mathcal{Z}_\mathrm{da})$ formalizes the model of a synchronous network with dynamic participation: For any given time slot, $\mathcal{A}$ determines which honest nodes are awake/asleep at that slot, subject to the constraint that at all time slots, at most β fraction of awake nodes are adversarial and at least one honest node is awake. At all times, the adversary is required to deliver messages sent between honest nodes in at most Δ slots. Adversarial nodes are assumed to be always awake.

Examples: To illustrate the model above, 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 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 [9, 10, 28, 41] 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 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.

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

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 C is *secure* and has transaction confirmation time T_confirm if ledgers output by C satisfy:

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

Definition 2. For static (dynamic) participation, *safety resilience* of a protocol is the minimum number f of adversarial nodes (minimum fraction β of adversarial nodes among awake nodes) to cause a safety violation. Such a protocol provides f*-safety* (β*-safety*).

Definition 3. For static (dynamic) participation, *liveness resilience* of a protocol is the minimum number f of adversarial nodes (minimum fraction β of adversarial nodes among awake nodes) to cause a liveness violation. 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 [25], as follows:

Definition 4. An adjudication function $\mathcal{J}$ takes as input two sets of evidences W and W' with conflicting ledgers $\mathsf{LOG} \triangleq C(W)$ and $\mathsf{LOG}' \triangleq C(W')$, 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 $\mathsf{LOG} \triangleq C(W)$ and $\mathsf{LOG}' \triangleq C(W')$ conflict with each other, they call $\mathcal{J}$ on these evidences to identify nodes that have violated the protocol.

Accountable safety resilience builds on the concept of *α-accountable-safety* first introduced in [6]:

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

In this section, we investigate the fundamental tradeoffs between liveness, safety and accountable safety resiliences shown in Figure 1 under three different network environments: synchrony $(\mathcal{A}_\mathrm{s}, \mathcal{Z}_\mathrm{s})$, partial synchrony $(\mathcal{A}_\mathrm{p}, \mathcal{Z}_\mathrm{p})$ and dynamic participation $(\mathcal{A}_\mathrm{da}, \mathcal{Z}_\mathrm{da})$.

3.1 Accountability and Liveness are Incompatible Under Dynamic Participation

We observe that the strictest tradeoff between the liveness and accountable safety resilience occurs for dynamically available protocols under $(\mathcal{A}_\mathrm{da}, \mathcal{Z}_\mathrm{da})$ (Figure 1 (vi)), a result which was named the availability-accountability dilemma in Section 1.4:

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 some 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).

A formal proof building on this observation 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 $C(W_1) = [\mathsf{tx}_1]$.

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 $C(W_2) = [\mathsf{tx}_2]$.

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 $C(W_1) = [\mathsf{tx}_1]$. 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 $C(W_2) = [\mathsf{tx}_2]$. 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 $C(W_2) = [\mathsf{tx}_2]$. 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 $C(W_1) = [\mathsf{tx}_1]$. 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. □

3.2 Tradeoff Between Accountable Safety and Liveness Resiliences

Proof of Theorem 1 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 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 C, and thus are forced to output ledgers for evidences from any number of nodes. However, in the case of a BFT protocol with a fixed level of n participating nodes, such an attack on accountability will not be possible as the protocol could require a valid input to C to contain evidences from at least a certain fraction of the n nodes. In this context, instead of an availability-accountability dilemma, we can talk about a softer tradeoff between the accountable safety and liveness resiliences (Figure 1 (iv) and (v)), which is formalized below:

Theorem 2. *For any SMR protocol that satisfies f_a-accountable-safety and f_l-liveness, $f_a \leq \max(0, n - 2f_l + 2)$.*

Proof of Theorem 2 follows from the proof of [25, Theorem B.1] and is given in Appendix A. Both proofs rely on the observation that for an SMR protocol (or BA protocol in the case of [25, Theorem B.1]) that satisfies f_l-liveness (f_l-validity for [25, Theorem B.1]), no adjudication function is able to output more than $\max(0, n-2f_l+2)$ nodes in the event of a safety (agreement for [25, Theorem B.1]) violation without incorrectly accusing an honest node.

Finally, note that no SMR protocol provides f_l-liveness for $f_l > \lceil n/2 \rceil$ under any setting, as will be explained in Section 3.3. This, together with Theorems 1 and 2, completes the characterization of the curves of Figure 1 (iv)–(vi). We proceed to show that these curves are tight. Tightness of (vi) under $(\mathcal{A}_{da}, \mathcal{Z}_{da})$ follows directly from the nature of the dilemma, any dynamically available protocol 'achieves' it. On the other hand, Sync Streamlet [10] and Sync HotStuff [2] achieve all liveness and accountable safety resilience points (f_l, f_a) shaded in blue in Figure 1 (iv) for $(\mathcal{A}_s, \mathcal{Z}_s)$, and Streamlet and HotStuff [41] achieve all (f_l, f_a) shaded in blue in Figure 1 (v) for $(\mathcal{A}_p, \mathcal{Z}_p)$. A more detailed discussion on the protocols, in particular on how the synchronous protocols can be made to provide accountability, is given in Appendix E.

3.3 Tradeoffs Between Safety and Liveness Resiliences

In this section, we formalize the tradeoffs between safety and liveness resiliences shown by Figure 1 (i)—(iii) under the three different network environments $(\mathcal{A}_s, \mathcal{Z}_s)$, $(\mathcal{A}_p, \mathcal{Z}_p)$ and $(\mathcal{A}_{da}, \mathcal{Z}_{da})$:

Theorem 3. *For any SMR protocol that satisfies f_s-safety and f_l-liveness, $f_l \leq \lceil n/2 \rceil$ and $f_s \leq n - f_l + 1$.*

Proof of Theorem 3 is given in Appendix B. The theorem applies to all SMR protocols under any network environment. It formalizes the common intuition that *in the presence of clients*, no consensus protocol can maintain security when the adversary controls half or more of the nodes. In particular, it shows that no safe SMR protocol can provide liveness if the adversary controls the majority of the nodes, as the adversary can always use its majority to either commit safety violations or to censor transactions.

Theorem 4. *For any SMR protocol that satisfies f_s-safety and f_l-liveness under $(\mathcal{A}_p, \mathcal{Z}_p)$, $f_l \leq \lceil n/2 \rceil$ and $f_s \leq n - 2f_l + 2$.*

Proof of Theorem 4 is given in Appendix C. It states the fundamental safety-liveness resilience tradeoff for partially synchronous protocols. It is a generalization of the celebrated $n/3$ resilience bound [19] for the security of partially synchronous protocols.

Theorem 5. *For any SMR protocol that satisfies β_s-safety and β_l-liveness under $(\mathcal{A}_{da}, \mathcal{Z}_{da})$, $\beta_l, \beta_s < 1/2$.*

Proof of Theorem 5 is given in Appendix D. It states the fundamental safety-liveness resilience tradeoff for dynamically available protocols, and generalizes the intuition behind a similar result for Proof-of-Work protocols given by [36, Theorem 3]. Dynamically available protocols are designed to output ledgers for sets of evidences containing responses from any number of nodes, as they do not know *a priori* the number of awake nodes. In this case, given two sets of evidences for conflicting ledgers, their *relative sizes* become the only selection criterion for such protocols, unlike in a static environment where the protocol can require evidences from a *fixed number* of nodes. Thus, the adversary is able to violate safety and liveness whenever it controls the majority among awake nodes.

Finally, we show that the curves of Figure 1 (i)–(iii) implied by Theorems 3, 4, and 5 are tight. Sync Streamlet [10] and Sync HotStuff [2] achieve all liveness and safety resilience points (f_l, f_s) shaded in blue in Figure 1 (i) for $(\mathcal{A}_s, \mathcal{Z}_s)$. Streamlet and HotStuff [41] achieve all (f_l, f_s) shaded in blue in Figure 1 (ii) for $(\mathcal{A}_p, \mathcal{Z}_p)$. Sleepy and Ouroboros [3, 15, 16, 26, 37] achieve all (β_l, β_s) shaded in blue in Figure 1 (iii) for $(\mathcal{A}_{da}, \mathcal{Z}_{da})$. A more detailed discussion on these protocols is given in Appendix E.

4 ACCOUNTABILITY GADGETS

In this section, we give a detailed description of the accountability gadget introduced in Section 1.5. For ease of exposition, we construct an accountability gadget from an accountable BFT protocol Π_{bft} with accountable safety and liveness resilience of $\lfloor n/3 \rfloor$.

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 [30], Sleepy [37], Ouroboros [3, 16, 26] and Chia [14] (Figure 2). 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. 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 $\mathsf{LOG}^t_{da,i}$.

The accountability gadget Π_{acc} has three main components as shown on Figure 2: a checkpoint vote generator (Algorithm 1) that issues checkpoint proposals and votes, an accountable SMR protocol Π_{bft} that is used to reach consensus on which votes to count for the checkpoint decision, and a checkpoint vote interpreter (Algorithm 2) that outputs checkpoint decisions computed deterministically from the ordered sequence of checkpoint votes output by Π_{bft}. The protocol Π_{bft} can be instantiated with any accountable BFT protocol such as Streamlet [10], LibraBFT [28] or HotStuff

Algorithm 1 Pseudocode for Checkpoint Vote Generator

```
1:  lastCp, props ← ⊥, {c : ⊥ | c = 0, 1, ...}                ▷ Last checkpoint, proposals
2:  for currIter ← 0, 1, ...
3:      if lastCp ≠ ⊥
4:          while waiting T_cp time                            ▷ Wait T_cp time after new checkpoint
5:              on Checkpoint(c, b) ← GetNextCp() with c = currIter
6:                  goto 23                                    ▷ Jump to conclusion of current iteration
7:              on Proposal(c, b) ← RecvVerifiedProp() with props[c] = ⊥
8:                  props[c] ← b    ▷ Keep track of proposals from authorized leader
9:      if CpLeaderOfIter(currIter) = myself
10:         Broadcast(⟨propose, currIter, GetCurrProposalTip()⟩_myself)
11:     while waiting T_to time
12:         on props[currIter] ≠ ⊥, but at most once    ▷ Act on the first proposal received from authorized leader before end of T_cp-wait and T_to-timeout
13:             if IsValidProposal(props[currIter])    ▷ Valid proposal extends latest checkpoint and is consistent with current checkpoint-respecting LC
14:                 SubmitVote(⟨accept, currIter, props[currIter]⟩_myself)
15:             else
16:                 SubmitVote(⟨reject, currIter⟩_myself)    ▷ Reject invalid proposal
17:         on Checkpoint(c, b) ← GetNextCp() with c = currIter
18:             goto 23                                    ▷ Jump to conclusion of current iteration
19:         on Proposal(c, b) ← RecvVerifiedProp() with props[c] = ⊥
20:             props[c] ← b        ▷ Keep track of proposals from authorized leader
21:     SubmitVote(⟨reject, currIter⟩_myself)              ▷ Reject due to timeout
22:     wait on Checkpoint(c, b) ← GetNextCp() with c = currIter
23:     lastCp ← b                                         ▷ Keep track of checkpoint decision
```

Algorithm 2 Pseudocode for Checkpoint Vote Interpreter

```
1:  for currIter ← 0, 1, ...
2:      currVotes ← {(pk, ⊥) | pk ∈ committee}             ▷ Latest vote of each node
3:      while true                                         ▷ Go through votes as ordered by Π_bft
4:          vote ← GetNextVerifiedVoteFromBft()            ▷ Verify signature
5:          if vote = ⟨accept, c, b⟩_pk with c = currIter
6:              currVotes[pk] ← Accept(b)                  ▷ Count accept vote for block b
7:          else if vote = ⟨reject, c⟩_pk with c = currIter
8:              currVotes[pk] ← Reject                     ▷ Count reject vote
9:          if ∃b : |{pk | currVotes[pk] = Accept(b)}| > 2n/3
10:             OutputCp(Checkpoint(currIter, b))          ▷ New checkpoint decision
11:             break
12:         else if |{pk | currVotes[pk] = Reject}| ≥ n/3
13:             OutputCp(Checkpoint(currIter, ⊥))          ▷ Abort current iteration
14:             break
```

[41]. 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 $\mathsf{LOG}_{\text{bft}}$, and emphasize that it remains internal to the protocol Π_{acc}. Checkpoint vote generator and interpreter are run locally by each node and interact with Π_{bft} and $\mathsf{LOG}_{\text{bft}}$. Hence, whenever we refer to $\mathsf{LOG}_{\text{bft}}$ in the following paragraphs, 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 new block in Π_{lc}. The checkpoint vote generator produces requests which can be of three forms: a proposal $\langle \text{propose}, c, b\rangle_i$ proposing block b for checkpointing in iteration c, an accept vote $\langle \text{accept}, c, b\rangle_i$ in favor of checkpointing a block b in iteration c, or a reject vote $\langle \text{reject}, c\rangle_i$ for iteration c. Here, $\langle \ldots\rangle_i$ denotes a message signed by node i. Each iteration c has a publicly verifiable and unique leader $\mathsf{L}^{(c)}$ sampled using a random oracle. The leader obtains the k_{cp}-deep block b on its checkpoint-respecting LC via the procedure GetCurrProposalTip() and broadcasts it to all other nodes as the checkpoint proposal for iteration c. (line 10 of Algorithm 1). Nodes receive checkpoint proposals from the network via the procedure RecvVerifiedProp(), and order according them with respect to their checkpoint iterations (lines 7 and 19 of Algorithm 1). For any given iteration c, RecvVerifiedProp() returns only proposals that were signed by the legitimate leader $\mathsf{L}^{(c)}$ of that iteration. A proposal is said to be *valid* in the view of a 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 that iteration is valid using the procedure IsValidProposal() (line 13 of Algorithm 1). If it has indeed received a valid proposal with the proposed block b, it votes $\langle \text{accept}, c, b\rangle_i$ (line 14 of Algorithm 1). Otherwise, if the proposal received for iteration c is invalid or if i does not receive any proposal for a timeout period T_{to}, it votes $\langle \text{reject}, c\rangle_i$ (lines 16 and 21 of Algorithm 1). Votes are input as payload to Π_{bft}, which outputs an ordered sequence of votes in the form of the ledger $\mathsf{LOG}_{\text{bft}}$. Thus, it enables nodes to reach consensus on which votes to count for an upcoming checkpoint decision.

The checkpoint vote interpreter (Algorithm 2) processes the sequence of votes in $\mathsf{LOG}_{\text{bft}}$ to produce checkpoint decisions. Each node receives verified votes (*i.e.*, with valid signature) in the order they appear on $\mathsf{LOG}_{\text{bft}}$ via the procedure GetNextVerifiedVoteFromBft() (line 4 of Algorithm 2). Upon observing unique accept votes $\langle \text{accept}, c, b\rangle_i$ from more than $2n/3$ nodes for a block b and the current iteration c, each node outputs b as the *checkpoint* for iteration c via the procedure OutputCp() (line 10 of Algorithm 2). The checkpointed blocks output by OutputCp() over time, together with their respective prefixes, constitute the ledger $\mathsf{LOG}^t_{\text{acc},i}$. Furthermore, the checkpoint decisions are fed back to Π_{lc} and the checkpoint vote generator so that they can ensure consistency of future block production in Π_{lc} and checkpoint proposals with prior checkpoints.

On the other hand, upon observing reject votes $\langle \text{reject}, c\rangle_i$ from $n/3$ nodes, each node outputs $\perp$ as the checkpoint decision for the current iteration c (line 13 of Algorithm 2). Here, $\perp$ signals that an iteration is aborted with no new checkpointed block, which happens if honest nodes suspect a faulty checkpoint leader and vote 'reject' because they have not seen progress for too long. Note that once a node outputs a checkpoint decision for its current iteration c, the checkpoint vote interpreter jumps to iteration $c+1$; thus, only a single decision is output per iteration.

Upon receiving a new checkpoint for the current iteration c via the procedure GetNextCp(), nodes terminate iteration c of the checkpoint vote generator and enter iteration $c+1$ (lines 6 and 18 of Algorithm 1). If the checkpoint decision was for a non-empty block b, nodes wait for T_{cp} time, denoted as the *checkpoint interval*, before they consider checkpoint proposals for iteration $c+1$. Similarly, an honest leader for iteration $c+1$ waits for T_{cp} time before it broadcasts the new checkpoint proposal. 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.

4.2 Security Properties

In this section, we formalize and prove the security properties **P1** and **P2** from Section 1.5 for accountability gadgets based on *permissioned* LC protocols [3, 16, 26, 37]. (For an extension of the security analysis to Proof-of-Work and Proof-of-Space LC protocols, see Sec-

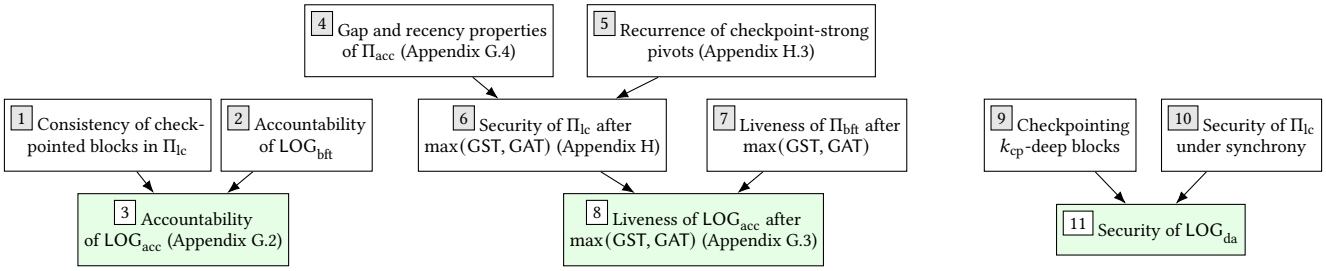

Figure 4: Dependency of the security properties of LOG_{acc} and LOG_{da} on the properties of Π_{acc}, Π_{lc} and Π_{bft}.

tion 6.) For this purpose, we first fix $f \leq \lceil n/2 \rceil$ and consider an accountability gadget Π_{acc} instantiated with a partially synchronous BFT protocol Π_{bft} that provides $(n-2f+2)$-accountable-safety at all times, and f-liveness under partial synchrony after the network partitions heal and sufficiently many honest nodes come online. (An example of such a protocol is HotStuff [41] with a quorum size of $n-f+1$. See Appendix E for further discussion.) To provide the same accountable safety resilience as Π_{bft} for LOG_{acc}, we select the thresholds for the number of accept and reject votes required to output a new checkpoint as $n-f$ and f in lines 9 and 12 of Algorithm 2, respectively.

Let $(\mathcal{A}_{pda}, \mathcal{Z}_{pda})$ denote a partially synchronous network with dynamic participation. It extends the partially synchronous network defined in Section 2 (with a global stabilization time GST) to include asleep/awake nodes and a *global awake time* GAT. Before GAT, the adversary determines which honest nodes are awake or asleep at each slot. After GAT, all honest nodes are awake. Here, GAT, just like GST, is chosen by the adversary, unknown to the honest nodes and can be a causal function of the randomness in the protocol. But, while GST needs to happen eventually (GST $< \infty$), GAT may be infinite. So, whereas GST represents the time when the partition heals, GAT represents the time when sufficiently many honest nodes wake up. With GST and GAT, $(\mathcal{A}_{pda}, \mathcal{Z}_{pda})$ enables us to express security properties for accountability gadgets under environments with both network partitions and dynamic participation.

Let λ denote the security parameter associated with the employed cryptographic primitives. Similarly, let σ denote the security parameter associated with the LC protocol Π_{lc}. Then, we can state the following theorem for the security properties of the ledgers LOG_{acc} and LOG_{da} output by the accountability gadget Π_{acc} and the permissioned LC protocol Π_{lc} (modified to be checkpoint-respecting):

Theorem 6. *For any λ, σ, and $T_{confirm}, k, k_{cp}$ linear in σ:*

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

Here, negl(.) denotes a negligible function, *i.e.*, a function that decays faster than all polynomials.

The resiliences achieved by LOG_{acc} and LOG_{da} are optimal, as can be seen from Theorem 2 which states that for any protocol satisfying f_a-accountable-safety and f_l-liveness, it must be the case that $f_a \leq n - 2f_l + 2$, and from Theorem 5 which states that for any protocol satisfying β_s-safety and β_l-liveness under $(\mathcal{A}_{da}, \mathcal{Z}_{da})$, it must be the case that $\beta_l, \beta_s \leq 1/2$.

To prove Theorem 6, we will first focus on the security of LOG_{da} under $(\mathcal{A}_{da}, \mathcal{Z}_{da})$ (box 11 of Figure 4). We know from [3, 16, 26, 37] that Π_{lc} is safe and live with some security parameter σ under the original LC rule when $\beta < 1/2$ (box 10 of Figure 4). 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 a node, it stays on the LCs held by all of the honest nodes for all times $t \geq s$. Since any block checkpointed by an honest node at time s has to be at least k_{cp}-deep in the LC held by the proposer of that block (box 9 of Figure 4), checkpoints are and will stay as part of the original LCs held by every other honest node for all times $t \geq s$. Then, the evolution of the checkpoint-respecting LCs will be the same as the evolution of the LCs in the absence of checkpoints. This implies that LOG_{da} inherits the security properties of the original LC protocol Π_{lc}.

We next focus on the accountability and liveness of LOG_{acc} under $(\mathcal{A}_{pda}, \mathcal{Z}_{pda})$ with global stabilization and awake times GST and GAT (boxes 3 and 8 of Figure 4). The pseudocode of Π_{acc} stipulates that honest nodes vote accept only for checkpoint proposals that are consistent with previous checkpoints they have observed (box 1 of Figure 4). Moreover, each new checkpoint requires at least $n-f+1$ accept votes (line 9 of Algorithm 2). Thus, in the event of a safety violation, either there should be two inconsistent ledgers LOG_{bft} held by honest nodes, or at least $n-2f+1$ nodes (which cannot be honest) must have voted on inconsistent checkpoints with respect to a single ledger LOG_{bft}. In both cases, at least $n-2f+2$ adversarial nodes can be identified by either invoking $(n-2f+2)$-accountable-safety of LOG_{bft} (box 2 of Figure 4) or the consistency requirement for checkpoints (box 1 of Figure 4). This implies $(n-2f+2)$-accountable-safety of LOG_{acc}, a detailed proof of which can be found in Appendix G.2.

Liveness of LOG_{acc} (box 8 of Figure 4) is the most involved part of the proof and requires the existence of iterations after max(GST, GAT) where all honest nodes vote accept for proposals by honest leaders. This, in turn, depends on whether the proposals by honest leaders are consistent with the checkpoint-respecting LCs held by the honest nodes after max(GST, GAT). To show that this is indeed the case, we prove that Π_{lc} recovers its security af-

ter max(GST, GAT) (box 6 of Figure 4). For this purpose, we first observe that in the presence of checkpoints, honest nodes abandon their LC if a new checkpoint is revealed on another (possibly shorter) chain. Then, there can be honest blocks that do not contribute to chain growth due to checkpoints arising at competing chains. This feature of checkpoint-respecting LCs violates a core assumption of the standard proof techniques [21, 26, 37] for LC protocols. Hence, to bound the number of such abandoned honest blocks and demonstrate the *self-healing* property of checkpoint respecting LCs, we follow a different approach introduced in [39]. In this context, we first observe the *gap* and *recency* properties for Π_{acc} (Appendix G.4) which were highlighted in [39] as necessary conditions for any checkpointing mechanism to ensure the self-healing of Π_{lc} (box 4 of Figure 4). The gap property states that the checkpoint interval T_{cp} has to be sufficiently longer than the time it takes for a checkpoint proposal to be checkpointed by Π_{acc}. 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 preceding the checkpoint decision.

Using the gap and recency properties, we next extend the analysis of [39] to Proof-of-Stake protocols by introducing the concept of *checkpoint-strong pivots*, a generalization of strong pivots from [37]. Whereas strong pivots count honest and adversarial blocks to claim the convergence of the LC in the view of different honest nodes, checkpoint-strong pivots consider only the honest blocks that are guaranteed to extend the checkpoint-respecting LC, thus resolving the issue of non-monotonicity for these chains. Recurrence of checkpoint-strong pivots after max(GST, GAT) (box 5 of Figure 4) along with the gap and recency properties of Π_{acc} enable us to assert the security of Π_{lc} after max(GST, GAT). Details of the security analysis for Π_{lc} can be found in Appendix H. Given the self-healing property of Π_{lc}, liveness of LOG_{acc} follows from the liveness of Π_{bft} after max(GST, GAT) (box 7 of Figure 4), full proof of which is given in Appendix G.3.

Finally, the prefix property follows from the fact that both LOG_{da} and LOG_{acc} are derived from the checkpoint-respecting LC. In particular, while LOG_{acc} corresponds to the prefix of the latest checkpoint, LOG_{da} corresponds to either the prefix of the latest checkpoint or that of the k-deep block on the checkpoint-respecting LC, selecting whichever one is longer. Hence, by construction, LOG_{acc} is always a prefix of LOG_{da}.

5 EXPERIMENTAL EVALUATION

To evaluate whether the protocol of Section 4.1 can serve as a drop-in replacement for Gasper as the Ethereum 2 beacon chain protocol, we have implemented a prototype[1]. Having proved security and accountability in Section 4.2, we are interested in its real-world behavior and performance characteristics at operating points chosen to match those of Ethereum 2. We then compare Gasper and our protocol in terms of the average required bandwidth and the latency of the accountable ledger, and conclude that our protocol can exceed the performance of Gasper. In particular, our protocol incurs comparable bandwidth at reduced latency. Finally, we observe that Gasper's resilience decreases as the number of nodes increases, for fixed latency of the accountable ledger, due to a new attack which we have discovered (detailed in Appendix I).

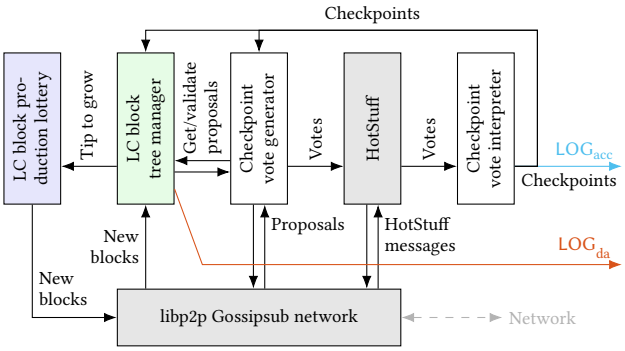

Figure 5: Components and their interactions in our implementation of Figure 2. Gray: off the shelf components used as black boxes. Blue: taken from Π_{lc} without modification. Green: taken from Π_{lc}, modified to respect checkpoints.

Implementation: Our prototype is implemented in the programming language Rust. A diagram of the different components and their interactions is provided in Figure 5. 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[2] 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[3], 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 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 $T_{\text{slot}} = 7.5$ s. We also match the block payload size of

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

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

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

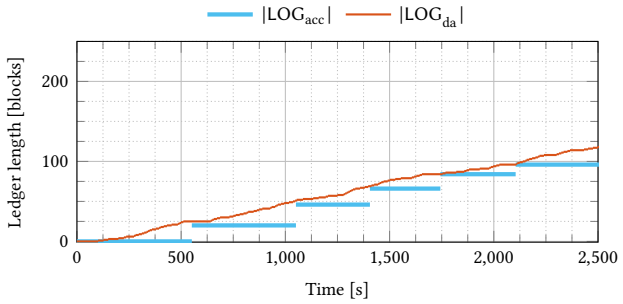

Figure 6: Both without attack (Figure 3) and in the presence of a $\beta = 25\%$ adversary who mines selfishly in Π_{lc} and boycotts leader duty in Π_{bft} and Π_{acc} (Figure 6) LOG_{da} grows steadily and LOG_{acc} periodically catches up with LOG_{da}. Under attack, the growth rate of LOG_{da} is reduced (due to selfish mining) and LOG_{acc}'s catching up is occasionally slightly delayed due to leader timeouts. (Parameters $n = 4100$, $T_{cp} = 5\,\text{min}$, $T_{to} = 1\,\text{min}$, $T_{hs} = 20\,\text{s}$, $k_{cp} = k = 6$, all nodes online.)

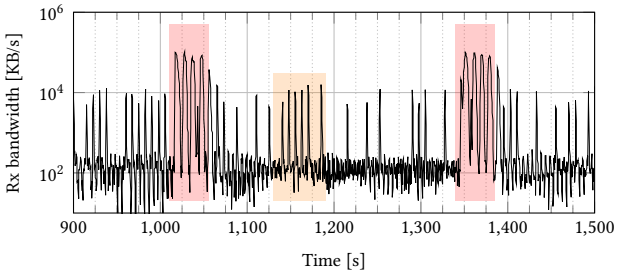

Figure 7: Setting of Figure 3: The network traffic for each AWS instance (*i.e.*, 82 nodes) shows four marked spikes (red) for every new checkpoint ($T_{cp} = 5\,\text{min}$ interval) and smaller spikes (orange) for every new Π_{lc} block ($T_{slot} = 7.5\,\text{s}$ interval).

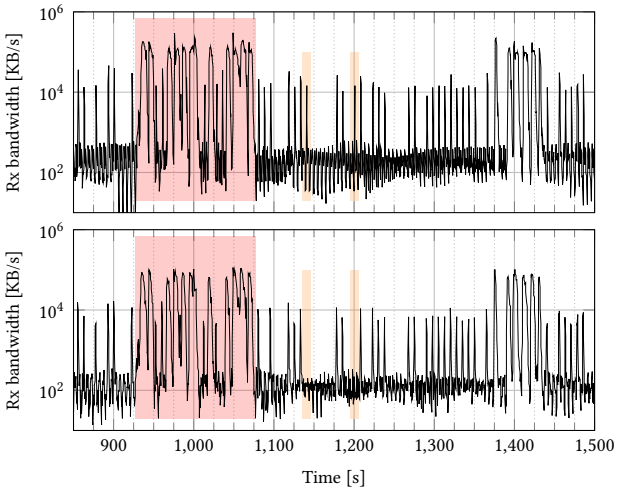

Figure 8: Setting of Figure 6: Leader timeouts in Π_{bft} and Π_{acc} can delay new checkpoints (red). *E.g.*, after the end of a checkpoint interval ($t \approx 870\,\text{s}$), and subsequent Π_{acc} leader timeout ($t \approx 930\,\text{s}$), honest nodes vote to reject the current checkpoint iteration, 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 ($t \approx 1070\,\text{s}$). Traffic at honest nodes (bottom) lacks some of the small spikes (orange) compared to traffic at adversarial nodes (top), since the adversary temporarily withholds some of its blocks from honest nodes due to selfish mining.

22 KBytes. In terms of Π_{lc}, we chose $k_{cp} = 6$ 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 LOG_{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 LOG_{da}. To leave enough time for message propagation, we set the HotStuff timeout $T_{hs} = 20\,\text{s}$. To avoid HotStuff timeouts escalating into checkpoint timeouts for honest leaders, we set $T_{to} = 1\,\text{min}$. Finally, to obtain 5× improvement in average LOG_{acc} latency over Gasper (*cf.* Figure 9), we set $T_{cp} = 5\,\text{min}$.

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 [20] in Π_{lc} to reduce honest chain growth. We ran our prototype (a) with no adversary (Figure 3), and (b) with $\beta = 25\%$ adversary (Figure 6), each for 2500 s on five AWS EC2 `c5a.8xlarge` instances in each of ten AWS regions[4], 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 3) as well as under the 25%-attack (Figure 6) 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 8 and 7 *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 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 ag-

[4] `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`

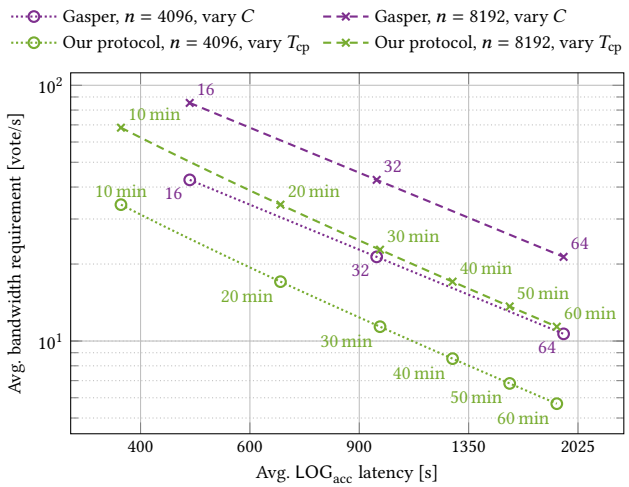

Figure 9: For fixed n, the average latency of $\mathsf{LOG}_{\mathrm{acc}}$ for Gasper and our protocol (here for $k_{\mathrm{cp}} = 6$) increases with the number C of slots per epoch and with T_{cp}, respectively, while the bandwidth required for votes reduces proportionally. Our protocol offers a better 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 $n = 4096, C = 32$).

gregation) and the average latency of the accountable ledger (see Figure 9), 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 9, 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 9 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 7 and 8 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 Appendix I suggests that even without adversarial but with merely random network delay, Gasper is susceptible to a balancing by an adversary controlling $O(\sqrt{C/n})$ of nodes. As a result, 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.

6 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 [30], or those based on Proof-of-Space (PoSpace), *e.g.*, Chia [14]. 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. Thus, security of Π_{lc} is primarily maintained by the miners, while security of Π_{acc} and the associated protocol Π_{bft} are primarily maintained by the validators. While the 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, the two example permissionless LC protocols we focus on. Then, the accountable ledger $\mathsf{LOG}_{\mathrm{acc}}$ and the available ledger $\mathsf{LOG}_{\mathrm{da}}$ satisfy the following properties:

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 $\mathsf{LOG}_{\mathrm{acc}}$ can provide an accountable safety resilience of $n - 2f + 2$ at all times, and it 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 6. However, accountability gadgets for permissionless LC protocols have further

requirements. First, under **P1**, liveness of $\mathsf{LOG}_{\mathrm{acc}}$ requires $\beta < \beta^*$. Otherwise, the large fraction of adversarial miners might stall Π_{lc} and hence $\mathsf{LOG}_{\mathrm{da}}$ might not become live after the network partition, which was argued to be a necessary condition for the 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 (line 9 of Algorithm 2), thus causing safety violations on $\mathsf{LOG}_{\mathrm{da}}$.

For Bitcoin, proof of the security properties above is given in Appendices G and H. For Chia, security can be proven via an extension of the method of blocktree partitioning [17] to the setting of checkpoint-respecting LCs. Further details can be found in Appendix H.5. Finally, note that the reduced threshold $\beta^* = 1/e$ if Chia is used as the permissionless LC protocol is due to nothing-at-stake attacks [17], since randomness is updated per each block in Chia. Other embodiments of PoSpace can provide different β^*.

ACKNOWLEDGMENTS

JN, ENT, and DT gratefully acknowledge funding from 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.

REFERENCES

[1] 2020. *Ethereum 2.0 networking specification.* https://github.com/ethereum/eth2.0-specs/blob/dev/specs/phase0/p2p-interface.md
[2] I. Abraham, D. Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *Symposium on Security and Privacy (S&P '20)*. IEEE, 106–118.
[3] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *Conference on Computer and Communications Security (CCS '18)*. ACM, 913–930.
[4] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2020. Consensus Redux: Distributed Ledgers in the Face of Adversarial Supremacy. IACR Cryptology ePrint Archive, Report 2020/1021.
[5] Vitalik Buterin. 2020. *Proposal for mitigation against balancing attacks to LMD GHOST.* https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation
[6] Vitalik Buterin and Virgil Griffith. 2019. Casper the Friendly Finality Gadget. arXiv:1710.09437
[7] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. 2020. Combining GHOST and Casper. arXiv:2003.03052
[8] Vitalik Buterin and Alistair Stewart. 2018. *Beacon chain Casper mini-spec (comments #17, #19).* https://ethresear.ch/t/beacon-chain-casper-mini-spec/2760/17
[9] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, 173–186.
[10] Benjamin Y. Chan and Elaine Shi. 2020. Streamlet: Textbook Streamlined Blockchains. In *Advances in Financial Technologies (AFT '20)*. ACM, 1–11.
[11] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. 2018. ALGORAND AGREEMENT: Super Fast and Partition Resilient Byzantine Agreement. IACR Cryptology ePrint Archive, Report 2018/377.
[12] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science* 777 (2019), 155–183.
[13] P. Civit, Seth Gilbert, and Vincent Gramoli. 2019. Polygraph: Accountable Byzantine Agreement. IACR Cryptology ePrint Archive, Report 2019/587.
[14] Bram Cohen and Krzysztof Pietrzak. 2019. The Chia Network Blockchain. https://www.chia.net/assets/ChiaGreenPaper.pdf.
[15] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *Financial Cryptography and Data Security (FC '19)*. Springer, 23–41.
[16] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*. Springer, 66–98.
[17] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2020. Everything is a Race and Nakamoto Always Wins. In *Conference on Computer and Communications Security (CCS '20)*. ACM, 859–878.
[18] Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Nielsen, and Daniel Tschudi. 2020. Afgjort: A Partially Synchronous Finality Layer for Blockchains. In *Conference on Security and Cryptography for Networks (SCN '20)*. 24–44.
[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (April 1988), 288–323.
[20] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.
[21] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*. Springer, 281–310.
[22] Yue Guo, Rafael Pass, and Elaine Shi. 2019. Synchronous, with a Chance of Partition Tolerance. In *CRYPTO 2019*. Springer, 499–529.
[23] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2007. PeerReview: Practical Accountability for Distributed Systems. *SIGOPS Oper. Syst. Rev.* 41, 6 (Oct. 2007), 175–188.
[24] Andreas Haeberlen and Petr Kuznetsov. 2009. The Fault Detection Problem. In *International Conference on Principles of Distributed Systems (OPODIS '09)*.
[25] Sreeram Kannan, Kartik Nayak, Peiyao Sheng, Pramod Viswanath, and Gerui Wang. 2020. BFT Protocol Forensics. arXiv:2010.06785
[26] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO 2017*. Springer, 357–388.
[27] Andrew Lewis-Pye and Tim Roughgarden. 2020. Resource Pools and the CAP Theorem. arXiv:2006.10698
[28] Libra Association. 2020. Libra White Paper. https://libra.org/en-US/white-paper/.
[29] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible Byzantine Fault Tolerance. In *Conference on Computer and Communications Security (CCS '19)*. ACM, 1041–1053.
[30] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf.
[31] Ryuya Nakamura. 2019. *Analysis of bouncing attack on FFG.* https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113
[32] Ryuya Nakamura. 2019. *Prevention of bouncing attack on FFG.* https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114
[33] Joachim Neu, Ertem Nusret Tas, and David Tse. 2020. *A balancing attack on Gasper, the current candidate for Eth2's beacon chain.* https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079
[34] Joachim Neu, Ertem Nusret Tas, and David Tse. 2020. Snap-and-Chat Protocols: System Aspects. arXiv:2010.10447
[35] Joachim Neu, Ertem Nusret Tas, and David Tse. 2021. Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma. In *Symposium on Security and Privacy (S&P '21)*. IEEE. arXiv:2009.04987 Forthcoming.
[36] Rafael Pass and Elaine Shi. 2017. Rethinking Large-Scale Consensus. In *Computer Security Foundations Symposium (CSF '17)*. IEEE, 115–129.
[37] Rafael Pass and Elaine Shi. 2017. The Sleepy Model of Consensus. In *ASIACRYPT 2017*. Springer, 380–409.
[38] Alejandro Ranchal-Pedrosa and Vincent Gramoli. 2020. Blockchain Is Dead, Long Live Blockchain! Accountable State Machine Replication for Longlasting Blockchain. arXiv:2007.10541
[39] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. 2021. Blockchain CAP Theorem Allows User-Dependent Adaptivity and Finality. In *Financial Cryptography and Data Security (FC '21)*.
[40] Alistair Stewart and Eleftherios Kokoris-Kogia. 2020. GRANDPA: a Byzantine Finality Gadget. arXiv:2007.01560
[41] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Symposium on Principles of Distributed Computing (PODC '19)*. ACM, 347–356.

A PROOF OF THE ACCOUNTABLE SAFETY-LIVENESS RESILIENCE TRADEOFF FOR SMR PROTOCOLS

Proof. For the sake of contradiction, suppose there exists a protocol Π that provides f_{l}-liveness for some constant $f_{\mathrm{l}} \leq \lceil n/2 \rceil$ and f_{a}-accountable safety for $f_{\mathrm{a}} = n - 2f_{\mathrm{l}} + 3$. Then, $\mathcal{J}$ outputs at least $n - 2f_{\mathrm{l}} + 3$ adversary nodes when there is a safety violation

and never outputs an honest node.

Suppose there are n nodes in $\mathcal{Z}$. Let P, Q and R denote disjoint sets consisting of $f_l - 1$, $f_l - 1$ and $n - 2f_l + 2 > 0$ nodes respectively. Given these sets, consider the following worlds:

World 1: $\mathcal{Z}$ inputs tx_1 to all of the nodes. Nodes in P and R are honest and the nodes in Q are adversary. Nodes in Q do not communicate with the nodes in P and R and do not output any transaction. Since $|P \cup R| = n - (f_l - 1)$, via f_l-liveness, nodes in P and R eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$.

World 2: $\mathcal{Z}$ inputs tx_2 to all of the nodes. Nodes in Q and R are honest and the nodes in P are adversary. Nodes in P do not communicate with the nodes in Q and R and do not output any transaction. As $|Q \cup R| = n - (f_l - 1)$, via f_l-liveness, nodes in Q and R eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$.

World 3: $\mathcal{Z}$ inputs tx_1 to the nodes in P, tx_2 to the nodes in Q, and both transactions to the nodes in R. Nodes in P are honest, nodes in Q and R are adversary. Nodes in Q do not send any message to any of the nodes in P. Nodes in R perform a split-brain attack where one brain interacts with P as if the input were tx_1 and it is not receiving any message from Q. Also, separately, nodes in Q and the other brain of R start with input tx_2 and communicate with each other exactly as in world 2.

Since worlds 1 and 3 are indistinguishable for the nodes in P, they along with the first brain of R eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Since Q and the second brain of R behave the same as in world 2, they eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$. Thus, there is a safety violation, in which case $\mathcal{J}$ outputs at least $f_a = n - 2f_l + 3$ adversarial nodes from the set $Q \cup R$. Hence, it outputs at least one node from the set Q.

World 4: $\mathcal{Z}$ inputs tx_1 to the nodes in P, tx_2 to the nodes in Q, and both transactions to the nodes in R. Nodes in Q are honest, nodes in P and R are adversary. Nodes in P do not send any message to any of the nodes in Q. Nodes in R perform a split-brain attack where one brain interacts with Q as if the input were tx_2 and it is not receiving any message from P. Also, separately, nodes in P and the other brain of R start with input tx_1 and communicate with each other exactly as in world 1.

Since worlds 2 and 4 are indistinguishable for the nodes in Q, they eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$ along with the first brain of R. Since P and the second brain of R behave the same as in world 1, they eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Thus, there is a safety violation, in which case $\mathcal{J}$ outputs at least $n = n - 2f_l + 3$ adversarial nodes. Note that worlds 3 and 4 are indistinguishable in the perspective of the adjudication function $\mathcal{J}$. Then, $\mathcal{J}$ would again output at least one node from the set Q (which is honest) with non-negligible probability. However, this violates the assumption that $\mathcal{J}$ never outputs an honest node, implying a contradiction with the definition of $\mathcal{J}$.

Next, we prove that $f_a = 0$ for any $f_l > \lceil n/2 \rceil$. Let P and Q denote disjoint sets of nodes with sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ respectively. Given these sets, consider the following worlds:

World 1: $\mathcal{Z}$ inputs tx_1 to the nodes in P and $\mathsf{tx}_2 \neq \mathsf{tx}_1$ to the nodes in Q. Nodes in P are honest and the nodes in Q are adversary. Nodes in Q do not communicate with the nodes in P and behave like honest nodes that do not hear from the nodes in P. Since $|P|, |Q| \geq n - (f_l - 1)$, via f_l-liveness, nodes in P and Q eventually generate the sets of evidences W_1 and W_2 such that $C(W_1) = [\mathsf{tx}_1]$ and $C(W_2) = [\mathsf{tx}_2]$. Thus, there is a safety violation and upon receiving W_1 and W_2, $\mathcal{J}$ outputs at least one node from the set Q.

World 2: Again, $\mathcal{Z}$ inputs tx_1 to the nodes in P and $\mathsf{tx}_2 \neq \mathsf{tx}_1$ to the nodes in Q. Nodes in Q are honest and the nodes in P are adversary. Nodes in P do not communicate with the nodes in Q and emulate the honest nodes in P within world 1. Since the nodes in Q within world 1 emulate the nodes in Q within world 2 and the nodes in P within world 2 emulate the nodes in P within world 1, worlds 1 and 2 are indistinguishable from the perspective of the adjudication function $\mathcal{J}$. In this case, $\mathcal{J}$ again outputs a node in Q with non-negligible probability. This violates the assumption that $\mathcal{J}$ never outputs an honest node, implying a contradiction. □

B PROOF OF THE SAFETY-LIVENESS RESILIENCE TRADEOFF FOR SMR PROTOCOLS

Proof. For the sake of contradiction, assume that there exists an SMR protocol Π that provides f_l-liveness for some constant $f_l \leq \lceil n/2 \rceil$ and f_s-safety for $f_s = n - f_l + 2$. Then, the protocol should be safe when there are $n - f_l + 1$ adversarial nodes. Suppose there are n nodes in $\mathcal{Z}$. Let P, Q and R denote disjoint sets consisting of $f_l - 1$, $f_l - 1$ and $n - 2f_l + 2 > 0$ nodes respectively. Next, consider the following worlds with three clients c_1, c_2 and c_3:

World 1: $\mathcal{Z}$ inputs tx_1 to all nodes. Nodes in P and R are honest and the nodes in Q are adversarial. There is only one client c_1. Nodes in Q do not communicate with the nodes in P and R; they also do not respond to c_1. Since $|P \cup R| = n - (f_l - 1)$, via f_l liveness, nodes in P and R eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Thus, c_1 outputs $[\mathsf{tx}_1]$ as the ledger.

World 2: $\mathcal{Z}$ inputs tx_2 to all nodes. Nodes in Q and R are honest and the nodes in P are adversarial. There is only one client c_2. Nodes in P do not communicate with the nodes in Q and R; they also do not respond to c_2. Since $|Q \cup R| = n - (f_l - 1)$, via f_l liveness, nodes in Q and R eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$. Thus, c_2 outputs $[\mathsf{tx}_2]$ as the ledger.

World 3: $\mathcal{Z}$ inputs tx_1 to the nodes in P, tx_2 to the nodes in Q, and both transactions to the nodes in R. Nodes in P are honest, nodes in Q and R are adversarial. There are two clients this time, c_1 and c_3. Nodes in Q do not send any message to any of the nodes in P; they also do not respond to c_1.

Nodes in R perform a split-brain attack where one brain interacts with P as if the input were tx_1 and it is not receiving any message from Q. Also, separately, nodes in Q and the other brain of R start with input tx_2 and communicate with each other exactly as in world 2. The first brain of R responds to c_1 and the second brain of R responds to c_3.

Since worlds 1 and 3 are indistinguishable for c_1 and the nodes in P, these nodes along with the first brain of R eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$, and c_1 outputs $[\mathsf{tx}_1]$. Since Q and the second brain of R behave the same as in world 2, they eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$. Observe that the client c_3 receives both sets of evidences W_1 and W_2.

World 4: $\mathcal{Z}$ inputs tx_1 to the nodes in P, tx_2 to the nodes in Q, and both transactions to the nodes in R. Nodes in Q are honest, nodes in P and R are adversarial. There are again two clients this time, c_2 and c_3. Nodes in P do not send any message to any of the nodes in Q; they also do not respond to c_2.

Nodes in R perform a split-brain attack where one brain interacts with Q as if the input were tx_2 and it is not receiving any message from P. Also, separately, nodes in P and the other brain of R start with input tx_1 and communicate with each other exactly as in world 1. The first brain of R responds to c_2 and the second brain of R responds to c_3.

Since worlds 2 and 4 are indistinguishable for c_2 and the nodes in Q, these nodes along with the first brain of R eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$, and c_2 outputs $[\mathsf{tx}_2]$. Since P and the second brain of R behave the same as in world 1, they eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Observe that the client c_3 receives both sets of evidences W_1 and W_2.

Finally, notice that the worlds 3 and 4 are indistinguishable for the client c_3. Thus, if it outputs the ledger $[\mathsf{tx}_1]$ (or $[\mathsf{tx}_2]$) in world 3, then it should also output the ledger $[\mathsf{tx}_1]$ in world 4 with non-negligible probability. Suppose it outputs $[\mathsf{tx}_1]$ ($[\mathsf{tx}_2]$) in both worlds. However, this implies a safety violation for $f = |P \cup R| = n - f_{\mathsf{l}} + 1$ many adversary nodes since in this case, two different clients, c_2 (c_1) and c_3 would output conflicting ledgers in world 4 (3). This is a contradiction with the assumption that $f_{\mathsf{s}} = n - f_{\mathsf{l}} + 2$.

Next, we observe that no SMR protocol Π provides f_{l}-liveness for $f_{\mathsf{l}} > \lceil n/2 \rceil$. For the sake of contradiction, assume the contrary and let P and Q denote disjoint sets of nodes with sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ respectively. Given these sets, consider the following worlds:

World 1: $\mathcal{Z}$ inputs tx_1 to the nodes in P and $\mathsf{tx}_2 \neq \mathsf{tx}_1$ to the nodes in Q. There is a single client c_3. Nodes in P are honest and the nodes in Q are adversarial. Nodes in Q do not send any message to any of the nodes in P; at the same time, they behave like honest nodes that do not receive any messages from the nodes in P. Since $|P|, |Q| \geq n - (f_{\mathsf{l}} - 1)$, via f_{l}-liveness, nodes in P and Q eventually generate sets of evidences W_1 and W_2 such that $C(W_1) = [\mathsf{tx}_1]$ and $C(W_2) = [\mathsf{tx}_2]$. Note that a client c_3 receives both sets of evidences W_1 and W_2.

World 2: $\mathcal{Z}$ inputs tx_1 to the nodes in P and $\mathsf{tx}_2 \neq \mathsf{tx}_1$ to the nodes in Q. Again, there is a single client c_3. Nodes in Q are honest and the nodes in P are adversarial. Nodes in P do not send any message to any of the nodes in Q; at the same time, they emulate the honest nodes in P . Since the nodes in Q within world 1 emulate the honest nodes in Q within world 2 and the nodes in P within world 2 emulate the honest nodes in P within world 1, P and Q generate sets of evidences W_1 and W_2 such that $C(W_1) = [\mathsf{tx}_1]$ and $C(W_2) = [\mathsf{tx}_2]$. c_3 again receives both sets of evidences W_1 and W_2.

Observe that worlds 1 and 2 are indistinguishable from the perspective of c_3. Thus, if it outputs $C(W_1) = [\mathsf{tx}_1]$ ($C(W_2) = [\mathsf{tx}_2]$) in world 1, it should also output $[\mathsf{tx}_1]$ ($[\mathsf{tx}_2]$) in world 2 with non-negligible probability. However, in this case, liveness will not satisfied in world 2 (1) since the transaction tx_2 (tx_1) received by honest nodes in world (2) would not appear in the ledger outputted by the client. This is a contradiction with the statement that Π provides f_{l}-liveness for $f_{\mathsf{l}} > \lceil n/2 \rceil$. □

C PROOF OF THE SAFETY-LIVENESS RESILIENCE TRADEOFF FOR PARTIALLY SYNCHRONOUS SMR PROTOCOLS

Proof. For the sake of contradiction, assume that there exists a partially synchronous SMR protocol Π that provides f_{l}-liveness for some constant $f_{\mathsf{l}} \leq \lceil n/2 \rceil$ and f_{s}-safety for $f_{\mathsf{s}} = n - 2f_{\mathsf{l}} + 3$ under $(\mathcal{A}_{\mathrm{p}}, \mathcal{Z}_{\mathrm{p}})$. Then, the protocol should be safe when there are $n - 2f_{\mathsf{l}} + 2$ adversarial nodes. We know from the proof of Theorem 3 that no SMR protocol can provide f_{l}-liveness for any $f_{\mathsf{l}} > \lceil n/2 \rceil$.

Suppose there are n nodes in $\mathcal{Z}$. Let P, Q and R denote disjoint sets consisting of $f_{\mathsf{l}} - 1$, $f_{\mathsf{l}} - 1$ and $n - 2f_{\mathsf{l}} + 2 > 0$ nodes respectively. Next, consider the following worlds, where each node also acts as a client:

World 1: $\mathcal{Z}$ inputs tx_1 to all nodes. Nodes in P and R are honest; nodes in Q are adversarial and do not communicate with the nodes in P and R. Since $|P \cup R| = n - (f_{\mathsf{l}} - 1)$, via f_{l}-liveness, nodes in P and R eventually generate a set of evidences W_1 and output $[\mathsf{tx}_1] = C(\mathsf{tx}_1)$.

World 2: $\mathcal{Z}$ inputs tx_2 to all nodes. Nodes in Q and R are honest; the nodes in P are adversarial and do not communicate with the nodes in Q and R. Since $|Q \cup R| = n - (f_{\mathsf{l}} - 1)$, via f_{l}-liveness, nodes in Q and R eventually generate a set of evidences W_2 and output $[\mathsf{tx}_2] = C(\mathsf{tx}_2)$.

World 3: $\mathcal{Z}$ inputs tx_1 to the nodes in P, tx_2 to the nodes in Q, and both transactions to the nodes in R. Nodes in P and Q are honest, nodes in R are adversary. Nodes in P and Q cannot communicate with each other due to a network partition before GST. Nodes in R perform a split-brain attack where one brain interacts with P as if the input were tx_1 and it is not receiving any message from Q. The other brain interacts with Q as if the input were tx_2 and it is not receiving any message from P.

Since worlds 1 and 3 are indistinguishable for the nodes in P, they output $[\mathsf{tx}_1]$ along with the first brain of R. Since worlds 2 and 3 are indistinguishable for the nodes in Q, they output $[\mathsf{tx}_2]$ along with the second brain of R. Thus, honest nodes output conflicting transactions, implying that there is a safety violation when there are $|R| = n - 2f_{\mathsf{l}} + 2$ many adversary nodes. This is a contradiction with the assumption that $f_{\mathsf{s}} = n - 2f_{\mathsf{l}} + 3$. □

D PROOF OF THE SAFETY-LIVENESS RESILIENCE TRADEOFF FOR DYNAMICALLY AVAILABLE SMR PROTOCOLS

Proof. For the sake of contradiction, assume that there exists a dynamically available SMR protocol Π that provides β_{l}-liveness and β_{s}-safety for some constant $\beta_{\mathsf{l}} < 1/2$ and $\beta_{\mathsf{s}} = 1/2 + \epsilon$ under $(\mathcal{A}_{\mathrm{da}}, \mathcal{Z}_{\mathrm{da}})$, where $\epsilon > 0$ can be chosen arbitrarily small. Then, the protocol should be safe when half of the awake nodes are adversarial. We know from the proof of Theorem 3 that no SMR protocol can provide β_{l}-liveness for any $\beta_{\mathsf{l}} > 1/2$.

Suppose there are n nodes in $\mathcal{A}$. Let P and Q denote disjoint sets, each consisting of $\lfloor n/2 \rfloor$ nodes. Next, consider the following worlds with three clients c_1, c_2 and c_3:

World 1: $\mathcal{Z}$ wakes up the nodes in P and inputs tx_1 to them. There is only one client, c_1. All the nodes in P are honest. Via β_{l}-

liveness, nodes in P eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Thus, c_1 outputs $[\mathsf{tx}_1]$ as the ledger.

World 2: $\mathcal{Z}$ wakes up the nodes in Q and inputs $\mathsf{tx}_2 \neq \mathsf{tx}_1$ to them. There is only one client, c_2. All the nodes in Q are honest. Via β_{l}-liveness, nodes in Q eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$. Thus, c_2 outputs $[\mathsf{tx}_2]$ as the ledger.

World 3: $\mathcal{Z}$ wakes up all of the nodes in $P \cup Q$ and inputs tx_1 to the nodes in P and tx_2 to the nodes in Q. There are two clients c_1 and c_3. Nodes in P are honest. Nodes in Q are adversarial and do not communicate with the nodes in P and the client c_1. Since the worlds 1 and 3 are indistinguishable for the nodes in P and c_1, nodes eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$, and c_1 outputs $[\mathsf{tx}_1]$. Nodes in Q simulate the execution in world 2, thus they eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$. Note that c_3 receives both W_1 and W_2.

World 4: $\mathcal{Z}$ wakes up all of the nodes in $P \cup Q$ and inputs tx_1 to the nodes in P and tx_2 to the nodes in Q. There are two clients c_2 and c_3. Nodes in Q are honest. Nodes in P are adversarial and do not communicate with the nodes in Q and the client c_2. Since the worlds 2 and 4 are indistinguishable for the nodes in Q and c_2, nodes eventually generate a set of evidences W_2 such that $C(W_2) = [\mathsf{tx}_2]$, and c_2 outputs $[\mathsf{tx}_2]$. Nodes in P simulate the execution in world 1, thus they eventually generate a set of evidences W_1 such that $C(W_1) = [\mathsf{tx}_1]$. Note that c_3 receives both W_1 and W_2.

Finally, notice that the worlds 3 and 4 are indistinguishable for c_3 with non-negligible probability. Thus, if it outputs the ledger $[\mathsf{tx}_1]$ ($[\mathsf{tx}_2]$) in world 3 (world 4), then it should also output the ledger $[\mathsf{tx}_1]$ ($[\mathsf{tx}_2]$) in world 4 (world 3). However, this implies a safety violation for $\beta = 1/2$ (half of the nodes are adversarial in worlds 3 and 4) since two different clients, c_2 (c_1) and c_3, would have outputted ledgers with conflicting transactions in world 4 (world 3). This is a contradiction with the assumption that $\beta_{\mathsf{s}} = 1/2$. □

E PROTOCOL EXAMPLES

In this section, we give examples of protocols that achieve the points on the optimal safety-liveness and accountable safety-liveness resilience tradeoffs presented in Figure 1.

Synchronous Protocols: We first observe that any point in the shaded regions of Figure 1 (i) and (iv), *i.e.* $(f_{\mathsf{l}}, f_{\mathsf{s}})$ such that $f_{\mathsf{l}} \leq \lceil n/2 \rceil$, $f_{\mathsf{s}} \leq n - f_{\mathsf{l}} + 1$ and $f_{\mathsf{a}} \leq n - 2f_{\mathsf{l}} + 2$, can be achieved by Sync Streamlet [10] and Sync HotStuff [2] with a quorum size of $n - f_{\mathsf{l}} + 1$ under $(\mathcal{A}_{\mathsf{s}}, \mathcal{Z}_{\mathsf{s}})$. For any such point $(f_{\mathsf{l}}, f_{\mathsf{s}})$, f_{s}-safety and f_{l}-liveness of the protocols follow directly from the security proofs in the respective papers.

To show the accountability of Sync Streamlet, we first observe that if a block is confirmed in the view of an honest node in Sync Streamlet, then that block would also be confirmed according to the confirmation rule of partially synchronous Streamlet. In particular, Sync Streamlet's confirmation rule requires 4 adjacent blocks from consecutive epochs to be at the tip of a *notarized* chain (with $n - f + 1$ votes on each block), whereas partially synchronous Streamlet requires only 3 such blocks. Hence, a safety violation on Sync Streamlet implies the existence of two conflicting blocks that would have been confirmed by partially synchronous Streamlet as well. Then, via [34, Theorem 1] which shows $n - 2f + 2$-accountable-safety of partially synchronous Streamlet with a quorum size of $n - f + 1$, we know that at least $n - 2f + 2$ nodes could be identified as adversarial nodes in the event of a safety violation on Sync Streamlet. Hence, Sync Streamlet with a quorum size of $n - f + 1$ satisfies $n - 2f + 2$-accountable-safety.

We complete the discussion on synchronous protocols with the accountability of Sync HotStuff:

Theorem 7. *For any $f \leq \lceil n/2 \rceil$, Sync HotStuff with a quorum size of $n - f + 1$ satisfies $n - 2f + 2$-accountable-safety.*

Proof. We first recall from [2] that if a block B received a quorum certificate of $n - f + 1$ votes from view v, it is denoted by $C_v(B)$ and called a certified block. Certified blocks are ranked first by views and then by heights *i.e.*, (i) blocks certified in a higher view has higher rank, and (ii) for blocks certified in the same view, a higher height gives a higher rank.

Next, suppose there is a safety violation with two conflicting blocks $C_v(B)$ and $C_{v'}(B')$ with quorum certificates from views v and v' respectively. Without loss of generality, suppose $C_{v'}(B')$ is the lowest ranked block that conflicts with $C_v(B)$ and has rank greater than or equal to it. If $v = v'$, via a quorum intersection argument, at least $n - 2f + 2$ nodes must have voted for equivocating blocks in the same view, thus can be identified as adversarial nodes by an adjudication function $\mathcal{J}$.

On the other hand, if $v < v'$, for the quorum certificate of $C_{v'}(B')$ to exist, at least $n - f + 1$ nodes must have voted for B' in view v' upon receiving $\langle \text{newview}, v^*, C_{v^*}(B') \rangle$, where $v^* < v'$ or $\langle \text{propose}, B', v', C_{v'}(B'') \rangle$ for some parent B'' of B'. Now, since $v < v'$, B'' cannot be in the prefix of B, and has to conflict with B. Thus, the latter case contradicts with the assumption that $C_{v'}(B')$ is the lowest ranked block that conflicts with $C_v(B)$, thus could not have happened. For the former case, similarly, $C_{v^*}(B')$ must rank lower than $C_v(B)$ since a higher ranking $C_{v^*}(B')$ would contradict with the definition of $C_{v'}(B')$ as the lowest ranked block that conflicts with $C_v(B)$ and has rank greater than or equal to it.

As $v < v'$, by assumption, there is no equivocating block in view v, implying that every honest node that voted on B in view v must have committed B in view v. Moreover, by [2, Lemma 1(ii)], if an honest node commits B, it should lock on a certified block with rank equal to or higher than that of $C_v(B)$ before entering view $v + 1 \leq v'$. Together, these observations imply that no honest node could have voted for $C_{v^*}(B')$ in view v' after voting for B in view v. However, via a quorum intersection arguement, at least $n - 2f + 2$ nodes must have voted for both B in view v and $C_{v^*}(B')$ in view v', *i.e.*, there are at least $n - 2f + 2$ double-votes in the quorum certificates of $C_{v'}(B')$ and $C_v(B)$. Hence, at least $n - 2f + 2$ nodes can be identified as adversarial by $\mathcal{J}$. □

Partially Synchronous Protocols: Our second observation is that any point in the shaded regions of Figure 1 (ii) and (v), *i.e.*, $(f_{\mathsf{l}}, f_{\mathsf{s}})$ tuples such that $f_{\mathsf{l}} \leq \lceil n/2 \rceil$, $f_{\mathsf{s}}, f_{\mathsf{a}} \leq n - 2f_{\mathsf{l}} + 2$, can be achieved by Streamlet or HotStuff [10, 41] with a quorum size of $n - f_{\mathsf{l}} + 1$ under $(\mathcal{A}_{\mathsf{p}}, \mathcal{Z}_{\mathsf{p}})$. For any such point $(f_{\mathsf{l}}, f_{\mathsf{s}})$, f_{s}-safety and f_{l}-liveness of the protocols again follow directly from the security proofs in the respective papers. Accountability results for Streamlet and HotStuff are proven by [34, Theorem 1] and [25, Theorems 5.1, 5.2]

Dynamically Available Protocols: Finally, we know that Nakamoto-style LC protocols [26, 30, 37] can operate at any $(\beta_{\mathsf{l}}, \beta_{\mathsf{s}})$ point

within the shaded region $\beta_l, \beta_s \leq 1/2$ on Figure 1 (iii), concluding the achievability for Figure 1.

F PROOF OF NON-ACCOUNTABILITY OF THE CHECKPOINTED LONGEST CHAIN PROTOCOL

In this section, we show that the checkpointed longest chain protocol presented in [39] does not provide accountable safety. The checkpointing protocol used by [39] is a slight modification of the Algorand BA protocol from [11]. Thus, the attack below on the accountability of the Algorand BA in [39] is very similar to the one described for Algorand BBA* [12] in [25, Appendix C.3].

Theorem 8. *Algorand BA [11] 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 the 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. □

G SECURITY PROOFS FOR THE ACCOUNTABILITY GADGETS

G.1 Theorem Statement and Notation

In this section, we consider an accountability gadget Π_{acc} instantiated with a BFT protocol Π_{bft} that provides $n-2f+2$-accountable-safety at all times, and f-liveness after $\max(\text{GST}, \text{GAT})$ under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$. (Recall that it is always possible to find such a BFT protocol as Π_{bft}; see Appendix E.) 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 $n-f+1$ and f 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 [37] 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 LOG_{acc} and LOG_{da} outputted by Π_{acc} and Π_{lc} satisfy Theorem 6 repeated below:

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

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

Before proceeding with the proofs, we formalize the concept of *security after a certain time* (We write $\text{LOG} \preceq \text{LOG}'$ 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 $\text{LOG}_i^t \preceq \text{LOG}_j^{t'}$ or $\text{LOG}_j^{t'} \preceq \text{LOG}_i^t$.
- **Liveness:** If a transaction is received by an awake honest node at some time $t \geq T$, then, for any time $t' \geq t + T_{\text{confirm}}$ and honest node j that is awake at time t', the transaction will be included in $\text{LOG}_j^{t'}$.

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.

G.2 Accountable Safety Resilience

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

Proposition 1. *Suppose the number of adversarial nodes is less than $n - 2f + 2$. 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 LOG_{bft} when the number of adversarial nodes is less than $n - 2f + 2$.

Theorem 9 (Accountable Safety of LOG_{acc})**.** LOG_{acc} *provides $n - 2f + 2$-accountable-safety.*

Proof. To show that LOG_{acc} provides $n - 2f + 2$-accountable-safety, we construct an adjudication protocol $\mathcal{J}$, which in the case of a safety violation on LOG_{acc}, outputs at least $n - 2f + 2$ nodes as adversarial and never outputs an honest node. For this purpose, suppose there is a safety violation on LOG_{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 $\text{LOG}_{\text{bft},s}^i$, there are accept votes $\langle \text{accept}, c', b_1 \rangle_k$ from more than $n - f$ nodes for the proposal b_1 and iteration c'. Similarly, within $\text{LOG}_{\text{bft},t}^j$, there are accept votes $\langle \text{accept}, c, b_2 \rangle_k$ from more than $n - f$ nodes for the proposal b_2 and iteration c. Thus, more than $2(n - f + 1) - n = n - 2f + 2$ nodes voted both $\langle \text{accept}, c', b_1 \rangle_k$ and $\langle \text{accept}, c, b_1 \rangle_k$. Let S denote the set of these nodes.

Next, consider the following two cases:

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

(ii) Suppose there is no safety violation on LOG_{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 $\langle \text{accept}, c, b_2 \rangle_k$ 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 $\langle \text{accept}, c', b_1 \rangle_k$ and $\langle \text{accept}, c, b_2 \rangle_k$, 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 LOG_{bft}, $\text{LOG}_{\text{bft},i}^s \preceq \text{LOG}_{\text{bft},j}^t$ or $\text{LOG}_{\text{bft},j}^t \preceq \text{LOG}_{\text{bft},i}^s$. In either case, all of the accept votes $\langle \text{accept}, c', b_1 \rangle_k$ and $\langle \text{accept}, c, b_2 \rangle_k$ are within the longer of $\text{LOG}_{\text{bft},i}^s$ and $\text{LOG}_{\text{bft},j}^t$, 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 $n - 2f + 2$ nodes as the set of nodes that have irrefutably violated the protocol. □

G.3 Liveness Resilience

We next focus on the liveness of LOG_{acc}.

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

Theorem 10 (Liveness of LOG_{acc})**.** *Suppose* Π_{lc} *is secure (safe and live) after some slot* $T \geq \max(\text{GST}, \text{GAT}) + \Delta + T_{\text{cp}}$. *Then,* LOG_{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 LOG_{bft}, and by implication LOG_{acc} will not be live. Thus, to show f-liveness of LOG_{acc}, we assume that there are less than f adversarial nodes and prove that LOG_{acc} satisfies liveness. In this case, again via Proposition 2, we know that LOG_{bft} satisfies liveness with transaction confirmation time T_{confirm} after $\max(\text{GST}, \text{GAT})$, 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 $\max(\text{GAT}, \text{GST})$. (Let $c' = 0$ if there does not exist such an iteration.) Since the honest nodes receive every message delivered before $\max(\text{GAT}, \text{GST})$ by slot $\max(\text{GAT}, \text{GST}) + \Delta$, all honest nodes would have entered iteration $c' + 1$ by slot $\max(\text{GAT}, \text{GST}) + \Delta$. Then, either all honest nodes are within iteration $c' + 1$ at slot $\max(\text{GAT}, \text{GST}) + \Delta + T_{\text{cp}}$ or an honest nodes have seen a block checkpointed for iteration $c' + 1$ by slot $\max(\text{GAT}, \text{GST}) + \Delta + T_{\text{cp}}$. In either case, entrance times of the honest nodes to subsequent iterations have become synchronized by slot $\max(\text{GAT}, \text{GST}) + \Delta + T_{\text{cp}}$: If an honest node enters an iteration $c > c'$ at slot $t \geq \max(\text{GAT}, \text{GST}) + \Delta + T_{\text{cp}}$, every honest node enters iteration c' by slot $t + \Delta$.

Suppose iteration $c > c'$ has an honest leader $\text{L}^{(c)}$, which sends a proposal $\hat{b}_c$ at slot $t > T + T_{\text{cp}}$. Note that $\hat{b}_c$ is received by every honest node by slot $t + \Delta$. Since the entrance times of nodes are synchronized by $T \geq \max(\text{GST}, \text{GAT}) + \Delta + T_{\text{cp}}$, 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 $\langle \text{accept}, c, \hat{b}_c \rangle_k$ for $\hat{b}_c$ by slot $t+\Delta$, all of which appear within $\mathsf{LOG}_{\text{bft}}$ in the view of every honest node by slot $t+\Delta+T_{\text{confirm}}$. Thus, $\hat{b}_c$ becomes checkpointed in the view of every honest node by slot $t+\Delta+T_{\text{confirm}}$. (Here, we assume that T_{to} was chosen large enough for $T_{\text{to}} > \Delta + T_{\text{confirm}}$ 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 $t - T_{\text{cp}} > T$, during the period $[t - T_{\text{cp}}, t]$, Π_{lc} satisfies the chain growth and quality properties (see Appendix H). 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 the it from before iteration c. (As a corollary, $\mathrm{L}^{(c)}$ extends all of the previous checkpoints seen by i and all other honest nodes.) This implies that $\hat{b}_c$ contains at least one fresh honest block that enters $\mathsf{LOG}_{\text{acc}}$ by slot $t+\Delta+T_{\text{confirm}}$.

Next, we show that an adversarial leader cannot make an iteration last longer than $\Delta + T_{\text{to}} + T_{\text{confirm}}$ for any honest node. Indeed, if an honest node i enters an iteration c at slot t, by slot $t+\Delta+T_{\text{to}}$, 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 $\langle \text{reject}, c \rangle_k$ from at least $n-f+1 \geq f$ 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 $((f+1)/n)^m$, 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 10 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 11 (Security of Π_{lc})**.** *If $p < (n-2f-2)/(2\Delta n(n-f-1))$ and there are $f-1$ (or less) adversarial nodes, for each sufficiently large T_{cp}, there exists a constant $\mathrm{C} > 0$ such that for any GST and GAT specified by $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$, $\Pi_{\text{lc}}(p)$ is secure after $\mathrm{C}(\max(\mathsf{GST}, \mathsf{GAT}) + \sigma)$, with transaction confirmation time σ, except with probability $e^{-\Omega(\sqrt{\sigma})}$.*

Proof of the theorem is given in Section H and relies on a combination of the method outlined in [39, Appendix C] with the concept of strong pivots from [37].

Finally, since $f \leq \lceil n/2 \rceil$, we can always find a p such that $p < (n-2f-2)/(2\Delta n(n-f-1))$. Then, given Theorems 11 and 10 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 $\mathrm{C}(\max(\mathsf{GST}, \mathsf{GAT}) + \sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$.

G.4 Recency and Gap Properties

Proof of Theorem 11 requires the accountability gadget Π_{acc} to satisfy two main properties first introduced in [39]: *recency* and *gap* properties.

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

Proposition 3 (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 3 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 [39], 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 $[t - T_{\text{rec}}, t]$. 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 $T_{\text{rec}} = \Delta + T_{\text{to}} + T_{\text{confirm}}$.*

Proof. We have seen in the proof of Theorem 10 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 $t-\Delta+T_{\text{to}}+T_{\text{confirm}}$. Thus, more than $n-f+1$ nodes must have voted $\langle \text{accept}, c, b \rangle_k$ by time t. Let j denote an honest nodes which voted $\langle \text{accept}, c, b \rangle_j$. Note that j would vote only after it sees the proposal for iteration c, i.e after slot $t - T_{\text{rec}} = t - \Delta + T_{\text{to}} + T_{\text{confirm}}$. Hence, b should have been on the checkpoint-respecting LC of node j at some slot within $[t - T_{\text{rec}}, t]$. This concludes the proof that every checkpoint block proposed after $\max(\mathsf{GST}, \mathsf{GAT})$ is T_{rec}-recent. □

H SECURITY PROOF FOR THE CHECKPOINT-RESPECTING LONGEST CHAINS

In this section, we prove Theorem 11, which states that the security of Π_{lc} is restored after $\max(\mathsf{GST}, \mathsf{GAT})$ under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$ provided that the election probability p of each node is sufficiently small. Via [35, Appendix C], we know that the Sleepy consensus protocol [37] regains its safety and liveness within $O(\max(\mathsf{GST}, \mathsf{GAT}))$ slots under $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$. On a similar note, [4] 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, [39] is the first work to

show the recovery of security for checkpoint-respecting LC protocols. As argued in [39], 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 [39] that relies on [21] 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 [3, 16, 37] 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 [39] by using the concept of strong pivots from [37].

In the proof below, we follow the same notation as [37]. 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 $f-1$ 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 $\beta = p(f-1)$ 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 $\alpha = p(n-f+1) > \beta$ as $f \leq \lceil n/2 \rceil$. For the convergence opportunities, we adopt the definition given in [37, 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 $\max(\mathsf{GST}, \mathsf{GAT}) + T_{\mathrm{rec}} < t_1^* \leq t_2^* \leq \ldots \leq$ be the slots at which new blocks $b_1^*, b_2^*, \ldots$ 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 [39] 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.

H.1 Chain Growth

Definition 7 ([37, 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 [39] to lower bound the chain growth in terms of convergence opportunities that lie within the inter-checkpoint intervals.

Lemma 2 ([39, 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 [39, 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 ([39, 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 [39, 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 $t_2 - t_1 = \tau \geq 2\Delta$, $\mathrm{C}(I \cap [t_1 + \Delta, t_2 - \Delta]) \geq k$.

H.2 Chain Quality

Definition 8 ([37, 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 $b_1, \ldots, b_{1/\mu}$ 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 quality$(\mu, 1/\mu) = 0$, $\mathrm{A}([s,t]) \geq 1/\mu$ and $\mathrm{A}([s,t]) \geq \mathrm{C}(I \cap [s+\Delta, t-\Delta]) - 1$. □

H.3 Common Prefix

Definition 9 ([37, Section 3.2.3])**.** Predicate prefix(τ) 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 [37, 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 $\mathrm{A}([t_0, t_1]) < \mathrm{C}(I \cap [t_0, t_1])$ 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 [37] about the prefix property. In this context, following proposition and lemma extend [37, Fact 4, Lemma 5]:

Proposition 4 (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. Let $b_{1,i}$ and $b_{1,j}$ from slots $t_{1,i}$ and $t_{1,j}$ be the last honest blocks on the chains $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ respectively that correspond to convergence opportunities within $(0,t] \cap I$. Similarly, let $b_{2,i}$ and $b_{2,j}$ from slots $t_{2,i}$ and $t_{2,j}$ be the first honest blocks on the chains $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ respectively that correspond to convergence opportunities within $[t, \infty) \cap I$.

Now, consider the following two cases: First, $t \in I$ is a convergence opportunity such that some honest block b^* at height ℓ^* was produced at t. 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^*. Then, via Proposition 4, $b_{1,i} = b_{1,j} = b_{2,i} = b_{2,j} = b^*$, and thus $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ could not have diverged at slot t.

Next, suppose t is not a convergence opportunity. In this case, by the definition of a checkpoint-strong pivot, within $\mathsf{ch}_i^{r_1}$, there cannot be any adversarial block between $b_{1,i}$ and $b_{2,i}$. Then, there also cannot be any convergence opportunity in $(t_{1,i}, t_{2,i}) \cap I$, since otherwise there would be another honest block in $\mathsf{ch}_i^{r_1}$ from a convergence opportunity t' such that either $t' \in (0,t] \cap I$, $t' > t_{1,i}$ or $t' \in [t, \infty) \cap I$, $t' < t_{2,i}$. Hence, $t_{1,i}$ and $t_{2,i}$ must be the two convergence opportunities within I closest in time to t on either side of t. As the same reasoning applies to $\mathsf{ch}_j^{r_2}$, we can conclude that $t_{2,i} = t_{2,j}$, which implies $b_{2,i} = b_{2,j}$. Hence, $\mathsf{ch}_i^{r_1}$ and $\mathsf{ch}_j^{r_2}$ could not have diverged at slot t. □

H.4 Probabilistic Analysis

To lower bound the number of convergence opportunities, we can use the following observation from [39, Proposition 4] which relies on Proposition 3 (gap property): If $t \geq s \geq \max(\mathsf{GST}, \mathsf{GAT})$,

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

Combining this expression with [37, Lemma 2, Corollary 2, Fact 2] yields the following proposition:

Proposition 5. *For any $\epsilon > 0$, there exists an ϵ' such that given $t_2 \geq t_1 \geq \max(\mathsf{GST}, \mathsf{GAT})$, $t \triangleq t_2 - t_1$,*

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

$$\mathbb{P}[\mathrm{A}([t_1, t_2]) > (1+\epsilon)\beta t] < \exp(-\epsilon^2 \beta t/3).$$

We also note that for any given $p < (n - 2f + 2)/(2\Delta n(n-f+1))$ and sufficiently large T_{cp}, there exists a constant $\epsilon > 0$ such that

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

Next, we define T as the minimum slot $t \geq \max(\mathsf{GST}, \mathsf{GAT})$ such that $\mathrm{C}([0,t] \cap I) = \mathrm{A}([0,t])$. 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 1 to hold, combining [35, Propositions 2,3,4] with Proposition 5, we can assert the following proposition bounding T:

Proposition 6. *There exists a constant* $\mathbf{C}$ *such that for any given security parameter* σ*, and* GST, GAT *specified by* $(\mathcal{A}_{\text{pda}}, \mathcal{Z}_{\text{pda}})$, $T \leq \mathbf{C}(\max(\text{GST}, \text{GAT}) + \sigma)$ *except with probability* $e^{-\Omega(\sigma)}$.

Using Proposition 6, we can complete the proof of Theorem 11:

Proof. Using Proposition 5 and Lemma 3, we can assert that for any given $\epsilon > 0$, growth(σ, k) is satisfied after $\max(\text{GST}, \text{GAT})$ except with probability $e^{-\Omega(\sigma)}$, where $k = g_0\sigma$ for

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

Similarly, using Lemma 4, Proposition 5 and Proposition 6, we can assert that for any given $\epsilon > 0$, quality$(\mu, 1/\mu)$ is satisfied after slot $\mathbf{C}(\max(\text{GST}, \text{GAT}) + \sigma)$, except with probability $e^{-\Omega(\sigma)}$, where

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

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 [37, Theorem 7] to show that prefix(σ) is satisfied after slot $\mathbf{C}(\max(\text{GST}, \text{GAT}) + \sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$. Then, using [37, Lemma 1], we conclude that Π_{lc} is secure with $T_{\text{confirm}} = O(\sigma/g_0)$ after slot $\mathbf{C}(\max(\text{GST}, \text{GAT}) + \sigma)$ except with probability $e^{-\Omega(\sqrt{\sigma})}$. □

H.5 Security Argument for Chia

While the sections above prove Theorem 6 for PoS, and by implication PoW protocols, security of Chia [14] does not immediately follow from the analysis of checkpoint-strong-pivots due to nothing-at-stake attacks [17], 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 [17] 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 H.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 [17, 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 [17, Section 4.2] goes through provided that $\beta < 1/e$ and the parameters p and $1/T_{\text{cp}}$ are sufficiently small. Details of this analysis is left as future work.

I ATTACKING GASPER WITHOUT ADVERSARIAL NETWORK DELAY

I.1 Motivation

Earlier works [8, 31–33, 35] have described balancing-type attacks against variants of the GHOST fork choice rule used in Gasper. In particular, the attack described in [33, 35] uses adversarial network delay to show that Gasper is not secure in traditional (partially) synchronous networks. While adversarial network delay (up to some delay bound) is a widely employed assumption in the consensus literature, there is disagreement whether it is appropriate for Internet-scale open-participation consensus. As a result, past attacks are often seen as impractical and have not been mitigated: "Note that this attack does depend on networking assumptions that are highly contrived in practice (the attacker having fine-grained control over latencies of individual validators), [...]" [5]

We show how the attack of [33, 35] can be modified and implemented so that an adversary controlling 15% of stake can stall Gasper *without requiring adversarial network delay*. To this end, we show through experiments that aggregate properties of many individually random message propagation processes (*e.g.*, 'within time T this transmission is received by fraction x of nodes') in real-world Internet-scale peer-to-peer gossip networks are sufficiently predictable to give the adversary the required control over how many validators see which adversarial messages when. None of the adversarial actions are slashable protocol violations.

I.2 High-Level Idea

Recall that the *balancing attack* [33, 35] consists of two steps: First, adversarial block proposers initiate two competing chains – call them Left and Right. Then, a handful of adversarial votes per slot, released under carefully chosen circumstances, suffice to steer honest validators' votes so as to keep the system in a tie between the two chains and consequently stall consensus.

Assume, w.l.o.g., that when viewing Left and Right with equal number of votes, the protocol's tie-break favors Left over Right. If the adversary manages to deliver a withheld adversarial vote for Right from an earlier slot to roughly one half of honest validators for the current slot i, before validators submit their votes for slot i, while the other half does not receive said vote before casting their votes, then roughly half of honest validators (those who have received the sway vote 'in time') see Right as leading and will vote for it in slot i, while the other half (those who see the sway vote 'late' and hence at the time of voting see a tie which they break in favor of Left) will vote for Left in slot i (see Figure 10).

Idealizing the above as voting according to a coin flip for each validator, roughly $m/2$ of m honest validators per slot would vote Left and Right, respectively, with a gap of $O(\sqrt{m})$ (*cf.* variance of a binomially distributed random variable). So, $O(1/\sqrt{m})$ adversarial fraction of stake would suffice to rebalance the vote to a tie and keep the system in limbo. In Section I.4 we provide evidence from real-world propagation delay measurements in a replica of Ethereum 2's gossip network [1] to support the hypothesis that the adversary can indeed reliably determine the time T_{delay} it takes for approximately half of nodes to receive a message broadcast by the adversary.

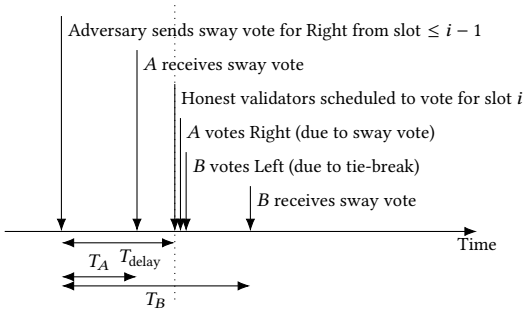

Figure 10: Assuming a tie between two chains Left and Right, with tie-break favoring Left. The adversary releases a sway vote for Right from a slot $< i$ at time T_{delay} before the point in time at which honest validators vote in slot i according to the protocol. The parameter T_{delay} is chosen such that roughly half of honest validators (such as A) receive the sway vote *before* they submit their vote (and hence vote Right, as Right *now* has more votes *in their view*), and the other half of honest validators (such as B) receive the sway vote *after* they submit their vote for (and hence vote Left, as the tie-break *still* favors Left *in their view*).

I.3 Detailed Description

First we describe the attack for a given T_{delay}, then we describe how to obtain T_{delay}. Our simulation[5] using the gossip network propagation model obtained in Section I.4 provides further details.

First, the adversary waits for an opportune epoch to launch the attack. An epoch is opportune if the block proposers in slot 0 and 1 are adversarial (this can be strengthened). Due to the random committee selection in Gasper, this happens with probability β^2 for any given epoch, so that the adversary needs to wait on average $1/\beta^2$ epochs until it can launch the attack. In the following, assume epoch 0 is opportune. The adversarial proposers of slots 0 and 1 propose conflicting new chains 'Left' and 'Right', respectively. Note that this is not a slashable protocol violation. Both withhold their proposals so that none of slot 0 or 1 honest validators vote for either block. The adversary releases the blocks to the network after slot 1. We assume w.l.o.g. that the tie between Left and Right (recall that no vote has been cast for either so far) is broken in favor of Left.

Time T_{delay} before honest validators in slot 2 vote, the adversary releases a vote for Right from an adversarial committee member of slot 1 (so called *sway vote*, see Figure 10). If T_{delay} is tuned well to the network propagation behavior *at large*, then roughly one half of honest committee members of slot 2 see the sway vote before they cast their vote, and thus view Right as leading (due to the sway vote) and will vote for it, and the other half see the sway vote only after they cast their vote, and thus view Left as leading (due to the tie-break) and will vote for it. Once the adversary has observed the outcome of the vote, which now should be a split up to an $O(\sqrt{m})$ gap, the adversary uses its slot 2 committee members (which stipulates the adversarial fraction $O(1/\sqrt{m})$ required for this attack) as well as slot 0 and 1 committee members to rebalance the vote to a tie. As the tie is restored, the adversary can use the same strategy in the following slot, and so forth.

Note that the adversary can observe the outcome of a vote and learns how many honest committee members saw Left and Right leading, respectively. The adversary can use this information to improve its estimate of T_{delay}. We show in Section I.4 that the optimal T_{delay} can be reliably localized even using grid search.

[5]Source code: https://github.com/tse-group/gasper-gossip-attack

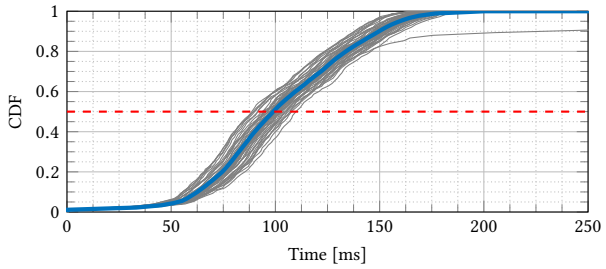

Figure 11: Fraction of participants in the peer-to-peer gossip network who have received a message broadcast by node 0 at time 0 by the given time (50 sample messages in gray, mean over all samples in blue). Median (dashed red) at ≈ 100 ms.

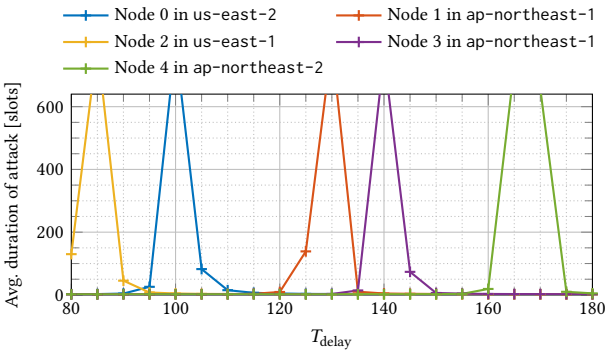

Figure 12: Using the propagation delay measurements to model network propagation, we simulated our attack for fixed $\beta = 0.15$, varying T_{delay}, and five different positions of the adversary in the network, and plot the resulting average duration of the liveness interruption (cut off at 800 slots horizon). Observe that the peak for node 0 fits well to the median observed in Figure 11. The curves are smooth and allow for easy and reliable localization of the optimal T_{delay}.

I.4 Experimental Evaluation

To understand whether the network propagation delay distribution is sufficiently well-behaved for an adversary to reproducibly broadcast messages so that they arrive at roughly half of nodes by a fixed deadline, we replicated the gossip network of Ethereum 2 [1] and measured the network propagation delay of test 'ping' packets from a designated sender to all nodes. The implementation in the Rust programming language used `libp2p`'s `Gossipsub` protocol and implementation, as is used in Ethereum 2 [1].

The gossip network comprised 750 nodes, each on an AWS EC2 `m6g.medium` instance (with 50 instances each in all 15 AWS regions that supported `m6g.medium` as of 21-April-2021). Each node initiated a connection with ten randomly chosen peers. The five nodes

with lowest instance ID were designated as senders and continuously broadcasted beacon messages with inter-transmission times uniformly distributed between zero and five seconds over a period of 20 minutes, logging the time when each message was broadcast. All nodes logged the time when a message was first received.

The network propagation delay was determined for each message and each receiving node. The respective CDFs, *i.e.*, what fraction of nodes have received a given message by a certain delay, is plotted as an example for a sample of messages from the first designated sender (node 0) in Figure 11 (together with the average CDF of all messages originating at node 0). (CDFs for the other four designated senders are omitted for brevity here. They show similar behavior, just slightly shifted in time.) It is apparent from the CDFs that depending on the location of the node (nodes 0, 1, 2, 3, 4 happened to be located in `us-east-2`, `ap-northeast-1`, `us-east-1`, `ap-northeast-1`, `ap-northeast-2`, respectively) both geographically as well as within the peer-to-peer network topology, the median of the average CDF varies, but considering messages originating at a fixed sender, the fraction of validators reached by the median of the average CDF is fairly concentrated around $1/2$. This suggests that the adversary can indeed determine T_{delay} so that with little dispersion honest validators get split in two halves.

We simulated the attack for $\beta = 0.15, m = 128$, using the network propagation delay samples to model random network delay.[6] Assigning the simulated adversary to one of the five designated senders for all of the attack, whenever the adversary broadcasts a sway vote, the propagation delays to the honest committee members of the given slot are sampled (without replacement) from the delays of one randomly drawn message of that designated sender.

To determine the optimal T_{delay}, we performed a grid search (with 5 ms step size) and for each T_{delay} simulated ten attacks in opportune epochs and recorded (see Figure 12) how long the adversary was able to stall liveness (terminating at a horizon of 800 slots). It is apparent that for the adversary in the position of each of the five designated senders of the measurement experiment, different T_{delay} are optimal. The optimal T_{delay} correspond well with the median of the average CDF (*cf.* Figure 11). As the curves are smooth and have a single distinct peak of width ≈ 5 ms, the adversary can locate the optimal T_{delay} well. In particular, even with T_{delay} approximating the optimal value only up to 10 ms, the adversary can stall liveness for dozens of slots. Recall that none of the adversarial actions are slashable protocol violations, so the adversary can refine T_{delay} iteratively and launch this attack over and over.

[6] Source code: https://github.com/tse-group/gasper-gossip-attack