Low-Latency Dynamically Available Total Order Broadcast

SRAVYA YANDAMURI, Duke University, USA and Common Prefix, EE
NIBESH SHRESTHA, Supra Research, USA
LUCA ZANOLINI, Ethereum Foundation, UK
KARTIK NAYAK, Duke University, USA

This work addresses the problem of Byzantine Fault-Tolerant (BFT) Total-Order Broadcast (TOB) in a dynamically available setting, where parties can transition between online and offline states without knowing the number of active parties. Dynamically available protocols are known to not be secure if parties commit at the speed of the network. In line with this result, existing dynamically available protocols rely on a synchronous network assumption, which means their latency remains tied to the pessimistic network delay Δ, even when the actual network delay is $\delta << \Delta$. This raises the question of whether a dynamically available BFT TOB protocol can maintain safety and liveness under synchrony while committing blocks with latency closer to the actual network delay. We answer this question affirmatively by designing the first dynamically available BFT TOB protocol that can commit blocks at the rate of 1 block per $O(\Delta_{ideal})$ time in expectation, where $\Delta_{ideal} < 2\delta$.

1 INTRODUCTION

Byzantine fault-tolerant (BFT) Total-Order Broadcast (TOB) enables a group of parties to agree on a sequence of client transactions despite the presence of a bounded number of Byzantine (malicious) parties. The traditional model assumes a permissioned network with a fixed set of parties known to each other. Bitcoin [21] introduced the first protocol operating in the *dynamically available* model [16], where parties can go online or offline freely, and the total number of active parties is unknown. Pass and Shi later formalized this concept as the sleepy model [23]. Dynamic availability is a crucial property for blockchain protocols, ensuring both safety (parties do not decide conflicting sequences of transactions) and liveness (all valid client transactions are eventually decided) even when an external adversary forces a large fraction of parties offline or introduces a surge of new parties.

There is a long line of work in improving the latency of dynamically available protocols [2, 11, 13, 25]. [1] Initial works using the longest-chain approach incurred latency that depended on the security parameter and the actual level of participation [1, 4, 23]. Several works, including Goyal et al. [13], Prism [2], parallel chains [11], and OHIE [25], removed the dependency on one of these parameters but not the other. Finally, MR [20] and MMR [17, 18] designed protocols with $O(\Delta)$ latency where Δ is a pessimistic network delay. However, all of these protocols only work under a synchronous [9] network assumption. While it is conceivable that the network is synchronous under a very large value of Δ (e.g., 10 mins), it may be the case that the maximum bound on the actual network delay δ is much lower. At the same time, relying on synchrony implies that the latency is a function of this pessimistic parameter Δ. On the other hand, while partially synchronous [10] and asynchronous protocols can be designed to allow parties to commit at the speed of the unknown actual network delay, dynamically available protocols are known to not be secure under partial synchrony or asynchrony [16]. Thus, we have the following natural question:

[1] We define latency as the amount of time it takes for all of the awake honest parties to commit a new block. When a party commits a block at height h, it also commits any uncommitted blocks it extends.

Authors' addresses: Sravya Yandamuri, sravya@commonprefix.com, Duke University, USA and Common Prefix, EE; Nibesh Shrestha, nibeshrestha2@gmail.com, Supra Research, USA; Luca Zanolini, luca.zanolini@ethereum.org, Ethereum Foundation, UK; Kartik Nayak, kartik@cs.duke.edu, Duke University, USA.

Can we obtain the best of both worlds, i.e., be secure under the synchrony assumption yet commit in time that is closer to the actual network delay?

We answer this question affirmatively in this work. Our core contribution is the design of a dynamically available Total-Order Broadcast protocol which outputs k logs by running k dynamically available TOB protocols with monotonically increasing delays $\Delta_1, \ldots, \Delta_k$, where it is guaranteed that Δ_k is at least the actual network delay. Any client with a correct assumption of the network delay enjoys consistency with all other clients which also have a correct assumption on the network delay because all logs created in the protocols with delays that hold have safety. Our protocol also ensures that all logs created with delays that hold eventually have latency of $O(\Delta_{ideal})$ in expectation for $\Delta_{ideal} < 2\delta$.

We work within the extended sleepy model of Malkhi et al. [18], referred to as the (T_f, T_b, α)-sleepy model, where the total number of corrupt parties during interval $[t - T_f, t + T_b]$ is less than α times the number of awake parties at time t. When $T_f = \infty$, the protocol does not allow the number of corrupt parties to decrease. Our solution relies on running multiple instances of the MMR protocol [18] with pessimistic delay parameters $\Delta_1, \ldots, \Delta_k$ where $\Delta_{i+1} = 2\Delta_i$ and $\Delta_k = \Delta$. In particular, we obtain the following result,

Theorem 1.1. *Let $\Delta_1, \ldots, \Delta_k$ be a series of delays such that $\forall i \Delta_i = 2\Delta_{i-1}$ and $\Delta_k = \Delta \geq \delta$, where δ is the actual network delay. There is a Total-Order Broadcast protocol in the $(\infty, 20\Delta, \frac{1}{2} - \epsilon)$-sleepy model for $0 < \epsilon < 1/2$ such that for all clients with correct assumptions on the network delay, blocks are eventually committed with a latency of $\max(O(\delta), O(\Delta_1))$ time in expectation.*

Although we achieve latency $\max(O(\delta), O(\Delta_1))$ what we achieve is a fixed parameter larger than δ and thus the protocol is not optimistically responsive. Due to this we don't fall into the trap of lower bounds or impossibility results. In fact, what we achieve is progress at the speed of $O(\Delta_{ideal})$, where Δ_{ideal} is the lowest of the k delays that holds. That $\Delta_{ideal} < 2\delta$ when $\delta > \frac{\Delta_1}{2}$ is directly related to our choice of $\Delta_{i+1} = 2\Delta_i$ in the protocol design. In comparison, the state-of-the-art dynamically available protocols incur a latency of $O(\Delta)$ in expectation. Finally, observe that the assumptions of the sleepy model are made with respect to Δ; that is, instances of the MMR protocol with different delay parameters Δ_i assume a bounded adversary relative to the delay Δ.

1.1 Approach and Key Challenges

Our core idea is to run multiple instances of the expected constant round dynamically available protocol (such as [17]), each based on different assumptions for the upper bound of network delay, denoted as $\Delta_1, \ldots, \Delta_i, \Delta_{i+1}, \ldots, \Delta_k$, where $\forall i \Delta_{i+1} = 2\Delta_i$ and $\Delta_k = \Delta$, i.e., the pessimistic network delay. Allowing all instances to propose new blocks simultaneously would lead to conflicting blockchains across different instances. Thus, only one instance is ever allowed to propose new blocks at a time. Initially, this responsibility lies with the instance that has the smallest delay bound, Δ_1. When a block B proposed in a lower-delay instance (e.g., Δ_i) is confirmed in that instance, π_i, it is then passed to the next higher instance, π_{i+1}, for reconfirmation. However, if the lower delay bound (such as Δ_1) does not hold, the protocol may stop receiving new input blocks, or parties may produce inconsistent outputs in that instance. In such cases, the responsibility for proposing new blocks shifts to the next higher instance, and parties stop engaging in the failed instance and all lower instances.

When taking the above approach, we want to ensure that blocks are committed with a latency of $O(\Delta_{ideal})$ where Δ_{ideal} is the smallest parameter larger than δ.

The protocol outputs all k logs, and each client may decide which log to believe. As long as they listen to a log corresponding to a delay that holds (i.e. for $\pi_{j \geq ideal}$), they will agree with the eventual truth. The crux of how this protocol achieves the desired latency is that the parties

propose *new* blocks in some instance $\pi_{j \leq ideal}$, and blocks are committed every $O(1)$ time steps in that sub-protocol in expectation (in the following paragraphs we discuss how we achieve this). Blocks committed in lower sub-protocols are confirmed in higher sub-protocols, and when a party commits a block it commits all uncommitted blocks that it extends. As a result, even if a client only believes in the log of π_k, a single commit in π_k incorporates multiple blocks that were committed in the lower sub-protocols, and the amortized latency for the log of π_k is the same.

To achieve the desired latency, we need to address the following key challenges. First, parties should never propose blocks in instances past π_{ideal}. Second, if the parties are proposing in instances π_j for $j < ideal$, they should still make sufficient progress (with the desired latency) or move to higher instances. Finally, we need to ensure that new parties joining the protocol have sufficient state to not stall the protocol.

Challenge 1: Ensuring blocks are not proposed in instances past π_{ideal}. Since not all Δ_j hold, it is necessary that the parties notice when sufficient progress is not being made in the sub-protocol corresponding to their estimate of the lowest delay that holds. Thus, if a party observes lack of progress or forks in the chain, then they must attempt to move to a higher sub-protocol. We need to, however, ensure that parties do not *incorrectly* attempt to move to a higher sub-protocol. In particular, in our protocol, the parties can only propose and vote for new blocks in a single sub-protocol at a time (otherwise, this would lead to conflicting chains). So even if a party is proposing and voting for new blocks in π_{ideal}, it may not see progress if all of the other awake honest parties are not also proposing and voting for new blocks in π_{ideal}, as the leader's proposal is not guaranteed to succeed if all of the awake honest parties do not vote for it. Thus, it is necessary that upon updating their estimate of Δ_{ideal}, parties only start to check that progress is made once they are sure that all of the other awake honest parties have also updated their estimate to Δ_{ideal}. Toward this end, our protocol uses a beacon protocol, which the parties use to locally decide when to increase their estimate of Δ_{ideal}. The beacon protocol is run with a delay of Δ_k, which is guaranteed to hold. Furthermore, it has the property that when one party updates their estimate of Δ_{ideal} to some Δ_j due to outputting j from the beacon protocol, all others will also output j and update their estimates accordingly within $3\Delta_k$. With this knowledge, the parties may safely begin expecting, and checking for, progress in π_j after a $3\Delta_k$ grace period elapses once they update their estimate and ensure that they do not move past π_{ideal}.

Challenge 2: Ensuring progress at the rate of Δ_{ideal}. A second challenge in designing such a protocol is that it is possible that a subset of the honest parties may see evidence that a given Δ_j does not hold, while others may not. For example, it is possible for some honest parties to see progress in $\pi_{j<ideal}$, while other honest parties see no progress. Or, an honest party might see two conflicting blocks decided in π_j, but given that the protocol has fluctuating participation, it may not be able to produce a proof to convince other honest parties that two conflicting blocks were certified. This can potentially ensure that parties are stuck on $\pi_{j<ideal}$ and not making sufficient progress. To handle this challenge, we ensure that if the honest parties *do not* eventually converge on Δ_{ideal}, then this is because progress is being made *at least* at the speed of Δ_{ideal}, and consistent chains are being created in all $\pi_{j \geq ideal}$. To achieve this, we have parties check that a block decided in π_j must be decided in π_{j+1} sufficiently quickly; otherwise this is evidence that Δ_j does not hold, and they attempt to increase their estimate. If $j < ideal$, and an honest party does *not* meet the condition to attempt to increase its estimate, if we set each delay to be double the previous delay, then the parties will make progress *at least* at a rate of 1 block per $O(\Delta_{ideal})$.

Challenge 3: Ensuring newly joining parties do not stall the protocol. A third challenge in designing our protocol relates to parties joining the network. As described in the preceding

paragraphs, there are various criteria that the parties use to detect whether their current estimate of Δ_{ideal} is wrong. The liveness of the protocol requires that if progress is not made in π_j, the parties update their estimates and try to start proposing and voting for new blocks in π_{j+1}. In a network without fluctuating participation, once all of the honest parties notice that Δ_j does not hold, they can all indicate this in the beacon protocol and subsequently update their estimates. However, when parties may join the network at any moment, we must be careful that parties joining at critical moments do not prevent the parties from increasing their estimates when needed, resulting in the protocol losing liveness. This could happen because a party must observe a lack of progress over a sufficiently long time period, and only at the end of that time period can it be sure that its current estimate does not hold. What is the duration of this sufficiently long period? Our protocol runs in *epochs* where in each *epoch*, if Δ_j holds and all parties are in π_j, a block will be committed with probability $\frac{1}{2} + \epsilon$ for $0 < \epsilon < 1/2$. Thus, we can set the duration to be a 2κ epoch period, where κ is a security parameter such that in expectation, κ blocks are committed every 2κ epochs and with high probability, a block is committed within 2κ epochs. If a newly joined party waits for a 2κ epoch time period to elapse without seeing progress before attempting to increase its estimate of Δ_{ideal}, once it nears the end of this period, *another* honest party may join the protocol, starting another waiting cycle. In this way, the network could never reach a state in which all of the awake honest parties are convinced that progress is not being made, and this could prevent them from ever succeeding in increasing their estimates.

To overcome this challenge, we present a joining protocol in which parties verify the amount of progress that has been made in the protocol up to the time they joined. Upon learning an estimate of the lowest delay that holds according the current knowledge of the awake honest parties, a joining party determines when the first parties to adopt this estimate did so, and verifies if sufficiently many blocks were committed since that point. We ensure that if a party joins the network with the estimate that $j = ideal$ and it is *not* convinced that its estimate is wrong, this is because an average of one block per $O(\Delta_j)$ time steps have been committed since the time at which the first honest parties to set their estimates to Δ_j did so. Otherwise, the party joins the network with its input to the beacon protocol already indicating that it wants to increase its estimate. We overcome these challenges and present a joining protocol that ensures that new parties joining the protocol do not prevent liveness.

This work is structured as follows. In Section 2, we provide a detailed technical overview of our solution. In Section 3, we present the necessary preliminaries. In Section 4, we present the dynamically available TOB protocol with an assumption on the state adopted by parties joining the network. We remove this assumption in Section 5, where we show the joining protocol. We present the proofs for the full protocol in Appendix B.

2 TECHNICAL OVERVIEW

As described in the introduction, our work takes the approach of running one instance of the expected constant round dynamically available protocol of MMR [18], one for each value of Δ_i. Initially, parties estimate that the network delay is Δ_1, considering it to be the actual delay Δ_{ideal}. They adjust this estimate to Δ_2 only when sufficient evidence suggests otherwise. Such evidence typically indicates possible network asynchrony or partitioning, which could lead to inconsistent outputs, in line with the known impossibility result presented in [15]. Eventually, without knowing Δ_{ideal}, we need to ensure that the current estimate $\Delta_j < \Delta_{ideal}$ makes sufficient progress, or the parties will change their estimate to Δ_{ideal}.

Protocol with a single Δ that holds. To begin with, we outline the scenario where there is a single delay Δ that holds. The protocol progresses through a series of *epochs*, with each epoch having a designated leader responsible for proposing a new block. The protocol uses a Lock-Commit paradigm; in this case, once a party decides a block (in a chain)[2], all honest parties *lock* onto the same block. The lock ensures safety across epochs: parties are restricted to voting only for blocks consistent with their current locks. This ensures that once a party decides on a block, no other party can subsequently decide on a conflicting block. To maintain the protocol's liveness, it is crucial that honest leaders propose blocks that all honest parties will *vote* for — specifically, blocks that extend the current locks of all honest parties. This ensures that the protocol continues to make progress while preserving safety.

To instantiate the above outline in the dynamically available setting, we adopt the approach of MMR [18] which relies on Graded Agreement (GA)[3]. In a GA, each party inputs a block and outputs a set of blocks with associated *grades* of 0 or 1. GA provides us with the following properties: (i) (graded delivery) if an honest party outputs a block with grade 1, all honest parties output the same block with any grade, (ii) (integrity) if an honest party outputs a block with any grade, then at least one honest party has input the block or a block extending it, and (iii) (validity) the highest block that every honest party's input extends will be output by all honest parties with grade 1.

Each epoch starts with two consecutive instances of Graded Agreement (GA) where parties choose their inputs to the second GA from their outputs of the first GA. Ultimately, this block of two GA's is used for parties to update their locks (to ensure that honest parties do not have conflicting locks) and choose their candidate blocks which they will extend or propose to ensure that parties propose blocks that all honest parties will vote for. To the first GA instance of the epoch, which we denote as GA_0, each party inputs their current lock. By choosing their input in this way, a party ensures that safety remains intact in this protocol, i.e., when one honest party decides a block, no honest party decides a conflicting block. Each party then selects the highest grade 1 output from GA_0 that does not conflict with any other outputs as their input to GA_1, the second Graded Agreement instance. The protocol ensures that such an output is guaranteed to exist by the validity and integrity of GA. Furthermore, the graded delivery property of GA ensures that the inputs to (and outputs from) GA_1 of the honest parties do not conflict. The output from GA_1 is used by honest parties to update their locks. As mentioned before, when an honest party decides on a block, all honest parties lock on it. The validity property of GA ensures that the highest block that every honest party's input extends will be output by all honest parties with grade 1. The integrity property of GA ensures that any block output by an honest party must have been input by an honest party (or a block extending it must have been). In this way, the parties choosing to input their locks to GA_0 preserves safety because if all honest parties were locked on a given block at the beginning of the epoch, their locks after the update would continue to extend that block. This mechanism also ensures that the locks of all honest parties are consistent, a crucial requirement for liveness, as parties will only vote for blocks that do not conflict with their current locks.

After outputting from GA_1, in addition to updating their lock, a party selects their *candidate* block as the highest output with any grade. Due to the graded delivery property of GA and the fact that the outputs from GA_1 do not conflict, this process ensures that the candidate blocks chosen by all honest parties are extensions of the locks held by all honest parties. Subsequently, each party generates a new block proposal extending their candidate and submits it to the Graded Proposal Election (GPE) instance (Definition 3.2). Owing to the validity property of GPE, with a probability

[2]Since each block uniquely determines its corresponding log, deciding a block B is equivalent to deciding on the log it represents.

[3]See Definition 3.1 in Section 3.1

of $1/2 + \epsilon$, all honest parties will decide on the block proposed by one of the honest parties. If any party decides a block B, all honest parties lock on that block, ensuring safety and liveness.

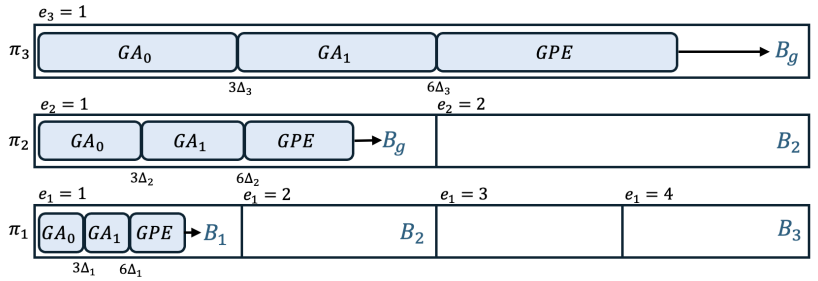

Fig. 1. A depiction of an execution of the protocol from time $t = 0$ to $10\Delta_3$. Each sub-protocol is executed with an increasing delay, where $\Delta_i = 2\Delta_{i-1}$. Because Δ_1 holds, new blocks are only decided in π_1 and blocks decided in π_1 may be used as input to π_2, and subsequently decided there. Similarly, blocks decided in π_i may be used as input, and decided in, π_{i+1}. In epoch 3 of π_1, no block is decided due to a dishonest leader of GPE.

Towards improving the latency to $O(\Delta_{ideal})$. What has been described so far outlines a protocol that functions effectively when a single delay bound, Δ, holds. However, our goal is to ensure that the protocol makes progress faster at the rate of $\Delta_{ideal} < 2\delta$ without knowing δ. The goal is to dynamically adjust to the correct delay, enabling the system to progress efficiently while maintaining safety, despite the uncertainty about which delay governs the current network conditions. Initially, parties assume that the network delay is Δ_1, considering it to be the actual delay Δ_{ideal}. They adjust this estimate to Δ_2 only when sufficient evidence suggests otherwise. Such evidence typically indicates possible network asynchrony or partitioning, which could lead to inconsistent outputs, in line with the known impossibility result presented in [15]. Eventually, either one of two possibilities occurs: (1) the parties converge on the actual network delay, Δ_{ideal}, which corresponds to Δ_k in the worst-case scenario; or (2) if the parties do *not* converge on Δ_{ideal}, this is because progress is being made *at least* at the desired rate of one block per $O(\Delta_{ideal})$ time. Regardless of which case occurs, parties never definitively know which delay is Δ_{ideal}, but the protocol ensures that progress is made at this speed eventually. Importantly, our protocol design ensures that no party increments their estimate past *ideal*, ensuring that progress is made at the speed of Δ_{ideal} and that the logs for all delays that hold do not conflict.

Our high level approach is as follows. The protocol consists of k sub-protocols π_i for $1 \leq i \leq k$, each associated with a delay parameter Δ_i. We set the delays such that $\Delta_i = \frac{\Delta_{i+1}}{2}$ and $\Delta_k = \Delta$. Parties execute all k sub-protocols simultaneously, continuously updating their estimate of the lowest delay that holds. Once a party has seen evidence that Δ_j does not hold, a party ceases participation in protocols $\pi_{1,\ldots,j}$. Of course, having k sub-protocols implies that honest parties output k logs. So how would parties choose which log is the *right one*? Our solution is to have each party maintain an *estimate* of the delay that they *believe* is the lowest delay that holds. Parties only vote for and propose *new* blocks in the single sub-protocol that is being executed at the delay which they believe is the lowest that holds. In the sub-protocols with higher delays, they only decide on blocks which were already decided in lower sub-protocols.

To determine whether to increment their estimate of the delay, parties rely on three key criteria. The first criterion checks for the *lack of progress in the current sub-protocol*. Within the sub-protocol, since honest leaders are chosen by the GPE instance with a probability of $1/2 + \epsilon$, we cannot

guarantee progress in a given epoch. However, because the leader selection events across instances are independent, we can expect to commit κ blocks within 2κ epochs with high probability. This check is triggered only after all parties have moved to the current sub-protocol (ensuring which requires additional care as discussed in challenge 1 in the previous section). The second criterion checks for the *lack of confirmation from sub-protocols with higher delay*. This check ensures that higher sub-protocols, and in particular π_{ideal}, are reconfirming sufficient blocks from the current sub-protocol. Since any $\Delta_{j'} > \Delta_j$ is a stronger assumption, lack of reconfirmations in any higher sub-protocol $\pi_{j'}$ implies insufficient progress in π_j. The last criterion checks for *consistency violations using higher sub-protocols*. Again, by a similar reasoning as the previous one, a sub-protocol with a larger delay is more authoritative. Thus, if, at any point, there is a conflict between what was confirmed in the current sub-protocol and what is reconfirmed in a sub-protocol with a larger delay, the party will decide to move to the next sub-protocol.

Finally, even if all honest parties have observed evidence to move to a higher sub-protocol, how do they agree to do so? Observe that this evidence suggests that the current estimate of delay is incorrect; thus, executing any protocol in this instance may not have any guarantees. Thus, the parties rely on a beacon protocol π_{beacon} that continuously runs GAs sequentially with parameter $\Delta_k = \Delta$. The beacon protocol ensures that if some party decides to update its estimate at a given time, all honest parties will update their estimate within another $3\Delta_k$ time.

Allowing parties to join the protocol correctly. A dynamically available protocol allows parties to join the protocol freely. From a modeling standpoint, we can classify parties in the protocol as being in one of three states: awake (i.e., actively participating parties), asleep (i.e., not participating), and recovering (i.e., a state where they receive sufficient state to be awake). Thus, when a party joins, it first enters the recovering state, then engages in a *joining protocol* to learn the necessary state required to participate before transitioning to the awake state. A natural question is, how does the party learn this necessary state in the joining protocol?

A naïve approach is for joining parties to start with their estimate $est = 1$, i.e., assume delay Δ_1, (like the initial awake parties) and obtain the necessary information to participate in each sub-protocol (i.e., a log and a lock). However, the protocol's liveness depends on parties incrementing est when progress stalls in the sub-protocol proposing new blocks. In particular, we need all awake honest parties to propose and vote for blocks in the same sub-protocol. Thus, if new parties continuously join with $est = 1$, the protocol may *never* advance beyond π_1, even when Δ_1 does not hold. This is because parties rely on the beacon protocol to move to higher sub-protocols, and the GA invocation in the beacon protocol is guaranteed to suggest moving only if all honest parties input an intent to move. Since a joining party does not have access to this information, if it indicates a lack of this intent, the parties may not be able to move to higher sub-protocols. Thus, if parties keep joining the network, the parties may never move past $est = 1$. On the other hand, if the joining party suggests an intent to move incorrectly (i.e., when the estimate is correct), this may eventually cause parties to move beyond π_{ideal}.

A better approach is for joining parties to first obtain the outcome of an iteration of the beacon protocol before joining. This allows them to determine the appropriate est value to start with. However, the same issue persists. The already awake parties may have noticed insufficient progress in the current sub-protocol which a joining party would not know. Just as it is about to set its intent to move, another party might join, stalling progress again. Thus, to prevent perpetual stalling at critical moments, parties must only join without intent to move if sufficient progress has been made in the current sub-protocol up to their joining time.

This leads to our solution, which relies on the following key idea: A party joining with a given est with a lack of intent to move should do so only after verifying that sufficient progress has been

made since the time that parties first adopted this value of est. To achieve this, the joining party must not only learn est from π_{beacon} but also obtain t^*_{est}, which represents the time when the first honest parties that adopted this value of est did so. Additionally, the joining party must acquire a log for each sub-protocol that matches the log of some already awake honest party, preventing the adversary from falsely inflating the perceived progress. Using t^*_{est} and these logs, the joining party can verify whether enough blocks have been committed in π_{est} without triggering criterion 1.[4] Moreover, it can determine if those blocks were committed in higher sub-protocols as expected to avoid triggering criterion 2, and whether the logs conflict with the ones in higher sub-protocols triggering criterion 3. Using this approach, our protocol ensures that the joining party assumes the approximate state of some awake honest party after recovering.

3 MODEL AND DEFINITIONS

We consider a system with n *parties*, denoted as $\mathcal{P} = \{p_1, \ldots, p_n\}$, connected through pairwise reliable, authenticated point-to-point channels. The system operates under an adaptive adversary capable of corrupting parties at any point during execution. Byzantine parties may exhibit arbitrary behavior, while honest parties strictly follow the protocol as specified. κ is a security parameter.

Cryptographic assumptions. We assume the use of a Public Key Infrastructure (PKI) and digital signatures. A message m sent by party p_i is digitally signed with p_i's private key, denoted as $\langle m \rangle_i$. Additionally, we assume a collision-free hash function, with $H(x)$ representing the invocation of the hash function H on input x. Lastly, we assume a verifiable random function (VRF). Each party p_i with its secret key can evaluate $(\rho, \pi) \leftarrow \mathsf{VRF_i}(\mu)$ on any input μ. The output is a deterministic pseudorandom value ρ along with a proof π. Using π and party p_i's public key, anyone can verify whether ρ is the correct evaluation of $\mathsf{VRF_i}$ on input μ.

Time. Our protocols operate in discrete rounds, where each round is represented by a value in $\mathbb{N}$. Protocol execution begins at round 1. We assume that all parties have perfectly synchronized clocks. Our protocol operates in the synchronous network model, in which all messages arrive within δ time, where $\delta \leq \Delta_k$ and δ is unknown.[5]

Sleepiness. Parties can be in one of three states during the protocol: *awake*, *asleep*, or *recovering*. Parties that are awake actively participate in the protocol, while parties that are asleep neither participate nor receive any messages. A party enters the recovering state when they are engaged in a *joining protocol*[6]. During the recovering state, the party receives all messages sent from the time they begin the joining protocol at time t, and they continue to receive messages until they transition to the asleep state. Importantly, we do not assume that messages are queued for a party while they are asleep.

Our protocol operates in the extended sleepy model of [18], which differs from the original sleepy model of [23]. In this extended model, the participation of corrupt parties may grow proportionally to the overall participation of the network, with a certain delay. Let T_b and T_f be non-negative integers and let $\alpha \geq 1$ denote a predetermined failure ratio of honest to Byzantine parties. Our protocol works in the (T_f, T_b, α)-sleepy model [18] for $T_f = \infty$, $T_b = 20\Delta (= 20\Delta_k)$, and $\alpha = \frac{1}{2} - \epsilon$. Let n_t denote the total number of parties awake at time t, and let $\mathcal{F}_t$ be the total number of corrupt

[4]The criteria are described earlier in this section and in Figure 2.

[5]k is chosen so that Δ_k is a conservative bound ensuring synchrony with high confidence. Its concrete value depends on the system. In blockchain systems, where external coordination is possible, a few hours may suffice. We leave Δ_k abstract as only the conditions $\Delta_{ideal} < \Delta_k$ and synchrony from Δ_k onward matter for our results.

[6]A joining protocol enables parties that were previously offline to retrieve messages sent during their absence. The duration of this protocol typically depends on the volume of data to be recovered. For this discussion, we consider the joining protocol in abstract terms, without delving into its specific mechanics.

parties awake at time t. Then:

$$f(t, T_f, T_b) = |\bigcup_{t-T_f \leq \tau \leq t+T_b} \mathcal{F}_\tau|. \quad (1)$$

An execution of the protocol is *admissible* in the (T_f, T_b, α)-sleepy model if for all $t \geq 0$ it holds that $f(t, T_f, T_b) < \alpha n_t$. To put it simply, the total number of Byzantine parties awake at any time during the time interval from $t - T_f$ to $t + T_b$ must be less than α fraction of the total number of parties awake at time t.

3.1 Graded Agreement, Graded Proposal Election, and Total-Order Broadcast

Logs and blocks. A *log* is defined as a finite sequence of *blocks*, represented by $\Lambda = [B_0, B_1, B_2, \ldots, B_k]$, where each block B_i contains a set of *transactions* and references the preceding block. For this discussion, we assume the existence of an external transaction pool from which honest parties retrieve transactions.[7] Each block includes a hash pointing to the previous block, creating a hash chain. The initial block in this chain, which lacks a reference to any other block, is called the *genesis block* and is denoted as B_0. The *height* of a block is defined by its position in the chain, measured as the distance from the genesis block. Any block B_k specifies a unique chain $B_0, \ldots, B_k$, which corresponds to a distinct log.

Two logs are considered *conflicting* if neither is a prefix of the other. Likewise, two blocks B and B' are said to conflict if the logs they uniquely define are conflicting. A log Λ is described as an *extension* of Λ' if and only if Λ' is a prefix of Λ. If $\Lambda' = \Lambda$, then Λ' and Λ are prefixes of each other. We use the notation $\Lambda[:-c]$ to indicate the last c blocks in the log Λ. We assume the existence of a function max() which takes as input two logs and returns the longer of the two. If both logs have the same length, it returns an arbitrary one. In addition we assume the existence of a function diff() which takes as input two logs and returns the blocks that are only in the longer log and not in shorter log. We make no assumptions on what it returns if neither log is a prefix of the other.

Definition 3.1 (Graded Agreement). In the *Graded Agreement* (GA) primitive, each party holds a block B, and outputs a set of blocks with associated *grades*, either 0 or 1. We represent the output as a set $\{(B, g)\}$. The log that party p_i submits to the GA is referred to as the *input of party* p_i. We require the Graded Agreement primitive to satisfy the following properties.[8]

Graded delivery. If an honest party outputs $(B, 1)$, then all honest parties output $(B, *)$.

Integrity. If an honest party outputs $(B, *)$, then at least one honest party has input B' extending B.

Validity. Let B be the highest block that every honest party's input extends. Then all honest parties output $(B, 1)$.

Note that there is no consistency property, so a single party may output multiple conflicting blocks.

Definition 3.2 (Graded Proposal Election). In graded proposal election (GPE), each party *proposes* their own block B and the parties *elect* a single block with grades. Each party is equipped with a function predicate() that takes as input a block B and outputs 0 or 1. At the end of the protocol, each party *outputs* a single pair (B, g) of a block B (or $B = \perp$) and a grade $g \in \{0, 1\}$ with the following guarantees:

Consistency. If two honest parties output $(B, *)$ and $(B', *)$ for $B, B' \neq \perp$, then $B = B'$.

[7] These transactions are validated against a set of predetermined rules before being compiled into blocks.

[8] It is important to note that there are several versions of Graded Agreement, each with distinct properties. The protocol presented in this work is based on the versions with the properties specified in [18].

Graded delivery. If an honest party outputs $(B, 1)$, then all honest parties output $(B, *)$.
Validity. If all honest parties input blocks that satisfy predicate() for all honest parties, then with probability $\frac{1}{2} + \epsilon$, all honest parties output $(B, 1)$ where B is the input of an honest party.
Integrity. If an honest party outputs $(B, *)$, then B satisfies predicate() for at least one honest party.

Throughout this paper, we generally treat Graded Agreement and Graded Proposal Election as black-box protocols; however we mention a few details regarding GPE because they come up later. At a high level, an instance of GPE consists of election of a leader via VRF, and the leader is responsible for proposing a block. The $\frac{1}{2} + \epsilon$ probability in the validity property of GPE comes from the probability that the elected leader is honest. Note that this is also the reason for the $\frac{1}{2} - \epsilon$ resilience of our protocol.

The GA protocol we utilize requires 3Δ time, while the GPE protocol takes 4Δ time [18]. Our focus is on the specific properties these protocols satisfy, and we build our solution upon those properties.

Definition 3.3 (Total-Order Broadcast [7]). A Total-Order Broadcast (TOB) protocol ensures that all the honest parties *decide on* the same log Λ. A protocol for BFT TOB satisfies the following properties.

Safety. If two honest parties deliver logs Λ_1 and Λ_2, then Λ_1 and Λ_2 do not conflict.
Liveness. The honest parties continue committing blocks.

We define a TOB protocol as *dynamically available* if it operates as a TOB protocol within the (T_f, T_b, α)-sleepy model.

4 DYNAMICALLY AVAILABLE TOTAL-ORDER BROADCAST

In this section, first provide a high-level overview of a protocol for dynamically available Total-Order Broadcast. In the following subsection, we present the pseudo-code for the protocol and provide a detailed explanation of its execution, highlighting how the protocol operates step by step under various conditions.

4.1 Total-Order Broadcast

Towards using multiple Δ_i's. Synchronous protocols require that *all* messages sent by honest parties are delivered within some fixed known time Δ after they are sent by the sender. This model has the advantage of being able to tolerate up to one-half Byzantine faults. In practice, ensuring the delivery requirement implies that the parameter Δ needs to be chosen pessimistically and it can be arbitrarily larger than the actual network delay δ. On the other hand, partially synchronous and asynchronous protocols can tolerate asynchrony in the network, but protocols under these models tolerate only one-third Byzantine faults.

Recent research in distributed consensus has focused on developing Total-Order Broadcast protocols within various adaptations of the sleepy model [23]. This has led to the creation of a series of dynamically available Total-Order Broadcast protocols [6–8, 12, 18, 20], which are capable of tolerating fluctuations in the participation levels of parties. These protocols must be synchronous [15][16, Theorem 7.1], meaning they rely on a known, system-wide upper bound Δ for communication delay and clock skew, while assuming that local computation time is negligible. This is, in practice, a significant limitation because the actual network delay may in practice be significantly shorter than Δ.

In this work, we aim to commit at a latency that is closer to the actual network delay. Our approach inolves attempting to run the protocol for multiple delay values $\Delta_1, \Delta_2, \ldots, \Delta_k$, where $\Delta_{i+1} = 2\Delta_i$

and $\Delta_k = \Delta$. The smallest such delay that "holds" is denoted as Δ_{ideal}, where $\Delta_1 \leq \Delta_{ideal} \leq \Delta_k$. In other words, $\Delta_{ideal-1} < \delta \leq \Delta_{ideal}$. Adopting this approach allows us to design protocols that obtain a latency better than the synchronous model (since Δ_{ideal} may be less than Δ). At the same time, since the protocol uses $\Delta_k = \Delta$ that always holds, it allows us to circumvent impossibility results pertaining to dynamic availability and partial synchrony [16].

The protocol. In Figures 2 and 3 we present our protocol for dynamically available TOB.

We start by describing the portion of the protocol presented in Figure 2. In this section we assume that joining parties adopt the exact state of some honest party that was already awake prior to it joining. We remove this assumption in Section 5.

Figure 2 depicts the sub-protocol that is run k times concurrently, where instance π_j is executed with delay Δ_j. Throughout the course of the protocol, a party maintains two important variables: *est* and *move*. The variable *est* is used to indicate the delay that the party believes is Δ_{ideal}. In protocol π_{est}, the party proposes new blocks and votes for new block proposals; it does not vote for new block proposals in any other sub-protocol. If a party sees evidence that Δ_{est} does not hold during the course of the protocol, they set $move := 1$. In addition to these variables, a party maintains for each j a timestamp t_j^*. Upon setting $est = j$, a party sets t_j^* to the current timestamp t_{now}. The purpose of this timestamp is that parties use it to detect whether they are seeing sufficient progress in π_{est}. We discuss in more detail how it is used later.

The sub-protocol π_j proceeds in epochs of length $10\Delta_j$. Each epoch begins with two instances of GA (Definition 3.1), as in the approach of [18]. To the first instance of GA, GA_0, the party inputs a block: either their lock from that sub-protocol, lock_j, or their highest decided block from the previous sub-protocol, decide_{j-1} if it extends lock_j. The latter option is a key difference in our protocol from that of [18]. The reason why parties input decide_{j-1} if it extends lock_j is for the liveness and prefix properties: a party must continue to grow all of its logs, and its logs constructed with higher delays must be prefixes of those constructed in lower delays. The set $\mathsf{out}_{0,j}$ contains all of the party's outputs from GA_0, meaning that it is a set of tuples of blocks and their corresponding grades. The party takes the block from $\mathsf{out}_{0,j}$ with the greatest height that is output with grade 1 such that no conflicting blocks were output (with any grade) to be its input to the second GA. From the outputs of the second GA, the party determines its candidate block, B_{cand} and updates lock_j. In particular, it sets lock_j equal to the block with the greatest height output with grade 1, and it sets B_{cand} equal to the block with the greatest height (of any grade) that it output. As described earlier, this approach enables the parties to have consistent locks, and for the parties to choose candidate blocks that extend the locks of all honest parties.

The next component of the sub-protocol is an instance of GPE (Definition 3.2). If a party has $est = j$, they create a new block proposal extending B_{cand} containing all the new transactions in their buffer. Otherwise, they propose B_{cand}. The parties execute GPE with their block proposal, as well as a specification for how they evaluate the function predicate on blocks within the GPE. Specifically, if $est = j$, the party only requires that the proposed block extends lock_j. In other words, the party accepts proposals of newly constructed blocks. Otherwise, the block must also have been output by the party from GA_0, meaning that this block was previously constructed and input to GA_0 by some honest party. The purpose of choosing block proposals and evaluating predicate on proposed blocks in this way is that it ensures that only blocks that were already committed in π_{low}, where *low* is the current value of *est* of the honest parties, will be committed in $\pi_{j>low}$. This is necessary because a conflict between Λ_{low} and Λ_j for $j > low$ is seen as evidence that Δ_{low} does not hold, and the parties will attempt to move *est* past *low* if this happens. If the parties move *est* past *ideal*, progress is not ensured at the speed of Δ_{ideal}. Furthermore, such a conflict violates the prefix property, and potentially the safety property as well.

Global variables:

B_0 := Genesis block
$est := 1$
$move := 0$
$t^*_1 := 0$
$t^*_{j \in 2,\ldots,k} := \infty$

Subprotocol π_j:

$\mathsf{lock}_j := \mathsf{decide}_j := B_0$
for epochs $e_j = 0, 1, 2, 3, \ldots$ **do**
  at $10e_j\Delta_j$ **do** ▷ Run $\mathsf{GA}_{0,j}$
    if $est < j$ and decide_{j-1} extends lock_j **then** $\mathsf{inp}_{0,j} := \mathsf{decide}_{j-1}$
    else $\mathsf{inp}_{0,j} := \mathsf{lock}_j$
    $\mathsf{out}_{0,j} := \mathsf{GA}_{0,j}(\mathsf{inp}_{0,j})$ ▷ $\mathsf{out}_{0,j}$ is a set of tuples (B^*, g) denoting a block and its grade
    B_{GA0} := the block with the greatest height output with grade 1 such that it does not conflict with any block output from $\mathsf{out}_{0,j}$
  at $(10e_j + 3)\Delta_j$ **do** ▷ Run $\mathsf{GA}_{1,j}$
    $\mathsf{out}_{1,j} := \mathsf{GA}_{1,j}(B_{GA0})$
    lock_j := block with the greatest height output with grade 1 from $\mathsf{out}_{1,j}$
    B_{cand} := block with the greatest height output from $\mathsf{out}_{1,j}$
  at $(10e_j + 6)\Delta_j$ **do** ▷ Run GPE
    if $est = j$ **then**
      $B_{\mathsf{proposal}} := (txns, H(B_{\mathsf{cand}}))$ ▷ Create a new block proposal
      $(B_{\mathsf{proposed}}, g) := \mathsf{GPE}_j(B_{\mathsf{proposal}}, \mathsf{predicate}(B) = \text{true if } B \text{ extends } \mathsf{lock}_j)$
    else
      $B_{\mathsf{proposal}} := B_{\mathsf{cand}}$
      $(B_{\mathsf{proposed}}, g) := \mathsf{GPE}_j(B_{\mathsf{proposal}}, \mathsf{predicate}(B) = \text{true if } B \text{ extends } \mathsf{lock}_j$ and a block extending B was output from $\mathsf{GA}_{0,j})$
    if $B_{\mathsf{proposed}} \neq \perp$ **then**
      $\mathsf{lock}_j := B_{\mathsf{proposed}}$
      if $g = 1$ **then** $\mathsf{decide}_j := B_{\mathsf{proposed}}$ (if haven't decided it already)
  at any time **do**
    upon deciding block B in π_j **do** set $e_{B,j} := e_j$ and $ts_{B,j} := ts_{now}$
    if any of the criteria for moving up is met **then**
      set $move := 1$

Criteria for moving to higher sub-protocols:

(1) **No progress in π_{est}.** Beginning at time $t^*_{est} + 3\Delta_k$ or later, a 2κ epoch period elapses in π_{est} during which fewer than κ blocks are decided
(2) **No confirmations in higher sub-protocols.** At time $t^*_{est} + 3\Delta_k$ or later for any j' s.t. $est \leq j' < k$ a block B is decided in $\pi_{j'}$ and more than 2κ full epochs of $\pi_{j'+1}$ elapse after $ts_{B,j'}$ without B having been decided in $\pi_{j'+1}$
(3) **Observing equivocations in higher sub-protocols.** $\mathsf{lock}_{j'}$ conflicts with the chain constructed in π_j for some $j' > j$ or there is a fork in the chain constructed in π_j for some $j \geq est$

Fig. 2. Protocol π_j for party P_i.

1: **at** time $t = 3\Delta_k, 6\Delta_k, 9\Delta_k, \ldots$ **do**
2: out := $\mathrm{GA}_{beacon}(est + move)$
3: let $v = val$ s.t. $(val, *) \in$ out and $\forall (val', grade') \in$ out, $val \geq val'$
4: let $v_1 = val$ s.t. $(val, 1) \in$ out and $\forall (val', 1) \in$ out, $val \geq val'$
5: **if** $v_1 > est$ **then**
6: cease participation in all sub-protocols π_j for $j < v_1$
7: set $est := v_1$, $move := 0$, $t^*_{est} := ts_{now}$
8: **if** $v > est$ **then**
9: set $move := v - est$

Fig. 3. Protocol π_{beacon} for party P_i run with delay Δ_k.

Once the GPE outputs, if the party outputs a block with grade 1, it commits the block. The party also updates lock_j if it output a block (with any grade) from GPE. Note that GPE has a uniqueness property, which ensures that there is a single block $B \neq \perp$ that the honest parties may output from a single instance of GPE. Due to the graded delivery property of GPE, this step ensures that if an honest party commits a block, all of the awake honest parties lock on it. If the delay corresponding to the sub-protocol holds, the fact that when one party decides a block all of the awake parties lock on it ensures that honest parties do not have conflicting chains.

Participation criteria. There are also certain instructions that a party executes at any point during the sub-protocol. If a party meets any of the criteria for moving up, i.e. it sees evidence that Δ_j does not hold, it sets $move := 1$.

We now describe these criteria in further detail. Criterion 1 captures that once all honest parties have $est = j$ for the same value of j, if Δ_j holds, the parties should commit κ new blocks every 2κ epochs. As we describe later, it takes $3\Delta_k$ time once one party sets $est = j$ for the rest of the parties to have $est = j$. For this reason, a party expects progress in π_{est} from time $t^*_{est} + 3\Delta_k$ onward. Criterion 2 captures the fact that a block committed in π_j should be committed in π_{j+1} in a timely manner. This captures the liveness property: that $\Lambda_j \forall j \geq ideal$ should continue to grow throughout the duration of the protocol. This property also ensures a more nuanced property. It is possible that Δ_j does not hold, and because of this only a single honest party sees progress in π_j. As we later show, it is necessary that *all* of the awake honest parties have $move := 1$ in order for them to increment est. Due to Criterion 2, if a party stops the network from converging on Δ_{ideal}, this is because a party is seeing progress at an *even lower delay*, and this progress is reflected in all the higher logs as well. It ensures that if the parties don't eventually converge on the lowest delay that holds, it is because progress is made at an even faster rate than it would have been if they did converge on Δ_{ideal}. Finally, Criterion 3 captures that seeing a conflict between Λ_j and $\Lambda_{j'>j}$ indicates that Δ_j does not hold. The protocol ensures that only blocks committed in lower sub-protocols may be committed or locked on in higher sub-protocols, so a violation of this property is grounds to set $move := 1$.

The π_{beacon} protocol. Next, we describe the protocol π_{beacon} in Figure 3. This protocol, executed with a delay of Δ_k, dictates when a party locally decides to increase its value of est. The protocol consists only of GAs executed at intervals of $3\Delta_k$. To each GA_{beacon}, a party inputs the sum of its value of est and its value of $move$. Within GA_{beacon}, the numbers 1 through k form a chain. That is, 2 extends 1; 3 extends 2; k extends $k-1$; and so on. From the outputs, out, of GA_{beacon}, the party takes the highest value output (with any grade), v, and the highest output with grade 1, v_1. If $v_1 > est$, the party ceases participation in π_{est}, sets $est = v_1$, resets $move := 0$, and sets $t^*_{est} := t_{now}$, the current timestamp. Then, if $v > est$, the party only sets $move := v - est$. The graded delivery

property of GA ensures that once one honest party sets $est = j$, all of the honest parties have $est = j$ within $3\Delta_k$ time. This property is crucial for our protocol, because progress in π_{est} is ensured only once *all* awake honest parties have *est* equal to that value. Since a party knows that after it has updated its value of *est*, all of the other honest parties will do so within $3\Delta_k$ time, it can expect to see progress as soon as that grace period has passed, as reflected in criterion 1 for moving up.

We now present the analysis of the protocol. In these proofs, we assume that when parties join the protocol, and changes their status to *awake* at time t, they assume the state of another honest party that was awake prior to and including time t. In Section 5, we show how to do away with this assumption with our joining protocol. We start by proving that the protocol satisfies a property called *conditional safety*. Conditional safety is safety within a sub-protocol π_j (no two honest parties decide different blocks at the same height) conditioned on two points: (1) Δ_j holds and (2) no honest party has $est > j$. Note that the first condition is equivalent to saying that $j \geq ideal$, and we use the two notions interchangeably.

Definition 4.1 (Conditional Safety). If Δ_j holds and no honest party has $est > j$, then if an honest party decides block B in protocol π_j, no honest party decides a block B' conflicting with B in π_j.

We start our protocol analysis with a proof of conditional safety because the safety and prefix properties of our TOB protocol requires that no honest party has incremented *est* past *ideal*. This is because honest parties propose new blocks in the GPE of π_{est} and evaluate predicate(B) = true for new blocks in π_{est}. As a result, if some honest party has $est > ideal$ the protocol does not prohibit a conflict between Λ_j for $j > ideal$ and Λ_{ideal}. One of the conditions by which a party decides that they wish to move to a higher sub-protocol is if they see a fork in π_{est} or if the lock in a higher sub-protocol conflicts with the chain they are constructing in π_{est} (condition 3). If conditional safety does not hold, then honest parties may increment *est* past *ideal*. If an honest party has $est > ideal$, then the protocol does not guarantee that blocks are decided at a latency of $O(\Delta_{ideal})$. Since the safety and prefix properties of the protocol and the fact that the property that no honest party increments *est* past *ideal* are interdependent, we start by proving a weaker safety property, conditional safety. The proof of conditional safety serves as a stepping stone to proving that no party increments *est* past *ideal*, which enables us to prove the that the protocol satisfies the properties of TOB and that progress is made at the speed of Δ_{ideal}.

We start with two helpful lemmas that are stepping stones to proving that conditional safety holds. In Lemma 4.2, we show that blocks are decided directly in lower sub-protocols before they are locked on (and as a consequence, also decided), in higher sub-protocols. This is an important step in proving that progress occurs at the speed of Δ_{ideal} and that the safety and prefix properties are preserved. If this property does not hold, Λ_j and $\Lambda_{j'}$ for $j, j' \geq ideal$ could conflict, violating these properties.

Lemma 4.2. *If $j \geq ideal$, and all honest parties have $est \leq j$, then for all $j' > j$, $\mathsf{lock}_{j'}$ held by any honest party was decided by an honest party in π_j.*

Proof. We prove this lemma by induction on j, demonstrating that the statement holds for $j' = j + 1$. It is evident that the statement holds at the beginning of the protocol. Now, assuming it holds at the start of epoch $e_{j'} \geq 0$ of $\pi_{j'}$, we aim to prove that it continues to hold at the start of epoch $e_{j'} + 1$.

At the start of epoch $e_{j'}$, honest parties input to $\mathsf{GA}_{0,j'}$ the highest block they decided in π_j (if it extends $\mathsf{lock}_{j'}$) or $\mathsf{lock}_{j'}$. All honest parties input a block that was decided by an honest party in π_j. This follows directly from the inductive assumption. The honest parties then update $\mathsf{lock}_{j'}$ to the highest grade 1 output from $\mathsf{GA}_{1,j'}$ (line 15). The lemma holds for these lock updates because of

the integrity property of GA and the fact that honest parties chose their inputs to $\mathsf{GA}_{1,j'}$ from their outputs from $\mathsf{GA}_{0,j'}$.

A necessary condition for honest parties (all having $est < j'$ by the lemma statement) to evaluate predicate to true on a block in GPE of $\pi_{j'}$ is that they output it (or a block extending it) with any grade from $\mathsf{GA}_{0,j'}$. This implies that if the leader's proposal succeeds, i.e., some honest parties output a non-$\perp$ block B from GPE, then some honest party decided the block B in π_j (by the integrity property of GA). The lemma therefore follows for lock updates after GPE outputs (line 25).

We have thus proven the statement for $j' = j + 1$. The lemma therefore holds for every two consecutive pairs of sub-protocols where both are at least j.

Next, we address general $j \geq ideal$ and $j' > j$. To do so, we will use the fact that we proved the statement for $j' = j + 1$ already. If an honest party sets $\mathsf{lock}_{j'} = B$ in $\pi_{j'}$ at time t, then some honest party decided it in $\pi_{j'-1}$ prior to time t, which follows from the fact that the lemma holds for $j' = j + 1$. Since an honest party decided B in $\pi_{j'-1}$, they must have locked on it as well (line 25). It follows from the fact that the lemma holds for $j' = j + 1$ that some honest party must have decided B in $\pi_{j'-2}$ prior to this. Using this line of reasoning, we see that the lemma holds for general $j \geq ideal$ and $j' > j$. □

Lemma 4.3. *If an honest party decides block B in epoch e_j of π_j at time t, Δ_j holds, and all honest parties have $est \leq j$, then all honest parties awake at any time $t' \geq t$ have* lock_j *extending B from time t' onwards.*

Proof. Note that all honest parties awake at time t update lock_j to B in epoch e_j due to graded delivery of GPE (line 25). We therefore only need to prove that no honest party locks on a block that doesn't extend B in any epoch after e_j.

We use a proof by induction to show that the lemma holds for all epochs after epoch e_j. We already showed that the lemma holds at the end of epoch e_j, as a base case. Assume that it holds at the end of some epoch e'_j. We show that it holds at the end of epoch $e'_j + 1$.

In epoch $e'_j + 1$ of π_j, all awake honest parties input to $\mathsf{GA}_{0,j}$ their lock_j or their highest decided block from π_{j-1} if it extends lock_j. In either case, they input a block extending B. It follows from integrity and validity that all awake honest parties input a block extending B to $\mathsf{GA}_{1,j}$. By integrity, no block output by an honest party from $\mathsf{GA}_{0,j}$ or $\mathsf{GA}_{1,j}$ may conflict with B, and by validity, the lock of all awake honest parties on line 15 extends B. By validity of GPE, if an honest party updates their lock due to outputting a block from GPE, then the block must extend B. It follows that all honest parties that are awake at the end of epoch $e'_j + 1$ have lock_j extending B. □

We are now ready to prove that the protocol satisfies conditional safety.

Lemma 4.4 (Conditional Safety). *If no honest party has $est > j$ and Δ_j holds, then if an honest party decides block B in protocol π_j, no honest party decides a block B' conflicting with B in π_j.*

Proof. Suppose an honest party h decides block B in protocol π_j in epoch e_j. We need to show that an honest party h' cannot decide on a conflicting block B' in π_j at any time. Assume that h' decides block B' conflicting with B in epoch e'_j; we will show that this implies a contradiction.

First, note that it cannot be the case that $e'_j = e_j$. This is because of the consistency property of the execution of GPE in epoch e_j. Without loss of generality, let us assume $e_j < e'_j$. Also, since we have $est \leq j$, by Lemma 4.3 all honest parties awake at any time $t' \geq t$ have lock_j extending B from time t' onwards. A necessary condition for parties to decide on a leader's proposal in GPE (i.e., evaluate predicate = *true* for the proposal) is if it extends their current lock. Thus, since all honest parties have lock_j extending B, no party would decide on a block conflicting with it (by the validity property of GPE). Thus, h' will not decide on B' in any epoch e'. □

Lemma 4.5. *If honest party p_i sets $est = j$ at time t, all honest parties awake at any time $t' \geq t + 3\Delta_k$ have $est \geq j$ by time t'.*

Proof. By graded delivery, all honest parties awake at time t output $(j, *)$ from GA at time t. The parties that output $(j, 0)$ set $move := j' - est$, where $j' \geq j$ (since they may also output a value higher than j). All honest parties that input to the subsequent GA do so with input at least j. By validity of GA, every awake honest party outputs $(j'', 1)$ for some value $j'' \geq j$ from the next GA and updates their value of est accordingly. The parties output from this GA at time $t + 3\Delta_k$, thus proving the lemma. □

Lemma 4.6. *If Δ_j holds and no honest party has $est > j$, then B_{cand} of every honest party during epoch e_j awake at time $(10e_j + 6)\Delta_j$ extends lock_j of all honest parties awake at that time during epoch e_j from time $(10e_j + 6)\Delta_j$ onward, and a block extending B_{cand} was output by all honest parties awake at any point during epoch e_j from time $(10e_j + 3)\Delta_j$ onward from $\text{GA}_{0,j}$ of that epoch.*

Proof. Because honest parties input to $\text{GA}_{1,j}$ their highest grade 1 output from $\text{GA}_{0,j}$ such that there were no conflicting outputs, it must be the case that none of the outputs of $\text{GA}_{1,j}$ of honest parties conflict during epoch e_j. This follows from the graded delivery and integrity properties of GA. Parties set their lock_j to be the highest grade 1 output from $\text{GA}_{1,j}$. The graded delivery property of GA ensures that all honest parties output the lock_j of all other honest parties, at least with grade 0. Moreover, B_{cand} is chosen to be the block with the greatest height from the output of $\text{GA}_{0,j}$. Since, the outputs of $\text{GA}_{0,j}$ do not conflict, B_{cand} set by all honest parties awake at time $(10e_j + 6)\Delta_j$ must either be equal to lock_j or a block that extends lock_j for all honest parties that are awake at that time. This proves the first part of the lemma.

To prove that B_{cand} of all honest parties at time $(10e_j + 6)\Delta_j$ was output from $\text{GA}_{0,j}$ of the same epoch by all honest parties that were awake at time $(10e_j + 3)\Delta_j$ onwards, recall that parties choose as B_{cand} their highest output of any grade from $\text{GA}_{1,j}$. The integrity property implies that some honest party must have input a block extending B_{cand} to $\text{GA}_{1,j}$ at time $(10e_j + 3)\Delta_j$. Honest parties choose B_{cand} their highest grade 1 output from $\text{GA}_{0,j}$ (with no conflicting outputs). The graded delivery property ensures that if an honest party outputs a block with grade 1, all honest parties output it with at least grade 0.

□

Lemma 4.7. *If Δ_j holds, no honest party has $est > j$, $j < k$, and all honest parties awake at time t have decided B (or a block extending it) in π_j by time t, then all honest parties awake at any point at or after t have lock_{j+1} equal to a block extending B for all epochs of π_{j+1} beginning at or after time t.*

Proof. By conditional safety (Lemma 4.4), no honest party decides a block conflicting with B in π_j. By conditional safety and Lemma 4.2, it follows that no honest party ever has lock_{j+1} conflicting with B. From the way that parties choose their inputs to $\text{GA}_{0,j+1}$ on lines 9- 10, it follows that all awake honest parties input to $\text{GA}_{0,j+1}$ of π_{j+1} B or a block extending it for all epochs beginning at or after time t.

In every epoch of π_{j+1} starting after time t, by the validity and integrity properties of GA, no honest party outputs a block conflicting with B from $\text{GA}_{0,j+1}$ and all honest parties awake at the time of outputting, output B or a block extending it with grade 1. As a result, all awake honest parties input a block extending B to $\text{GA}_{1,j+1}$. The lemma therefore follows for lock updates prior to GPE due to the integrity and validity properties of GA. Because all awake honest parties are locked on blocks extending B, and by the way honest parties evaluate predicate(), the lemma follows for lock updates after GPE due to integrity. □

In the following lemma, we show that for every 2κ epoch period where all honest parties awake at any point during that time period have $est = j$ for some $j \geq ideal$, there are at least κ epochs during that period with high probability s.t. all honest parties awake by the end of that epoch have decided a new block. In addition, we show that for all $j' > j$, during the time period where all awake honest parties have $est = j$, there is at least 1 epoch out of every κ in which a new block is decided by all honest parties awake by the end of that epoch with high probability. Later, we use this lemma to show that once there is a time t s.t. all honest parties awake at any point at or after t have $est = ideal$, they continuously decide sufficiently many new blocks in π_{ideal} which are also confirmed in higher sub-protocols.

Lemma 4.8. *If there are 2κ consecutive epochs in π_j where all honest parties awake at any point during that time period have $est = j$ and Δ_j holds, then during at least κ of those epochs all honest parties awake by the end of that epoch decide B, where B is the input of an honest party, with probability $1 - e^{-2\kappa\epsilon^2}$. In addition, during the time in which all awake honest parties have $est = j$, $\forall j' > j$ there is at least one epoch out of every κ in which all of the honest parties awake by the end of that epoch output B, where B is the input of an honest party, with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$.*

Proof. We start by proving the first statement. By Lemma 4.6, and line 19, in each such epoch of π_j, the inputs of all honest parties to GPE satisfy *predicate* for all honest parties that are awake at any point during the GPE. As such, in expectation, a block that is the input of an honest party to GPE is decided every $\frac{1}{\frac{1}{2}+\epsilon}$ epochs in expectation by the validity property of GPE by all honest parties awake by the end of that epoch. With the application of a Chernoff Bound, it follows that with probability $1 - e^{-2\kappa\epsilon^2}$, this occurs at least κ epochs out of every 2κ.

We now address the second statement in the lemma. Again, by Lemma 4.6, the inputs of all awake honest parties to GPE satisfy *predicate* for all honest parties awake during GPE in every epoch where all honest parties have $est = j$. The remainder of the lemma therefore follows from a simple Chernoff Bound. □

In the next lemma, we show that once one honest party decides a block in $\pi_{j \geq ideal}$, all honest parties that are awake κ epochs later will have decided it as well with high probability.

Lemma 4.9. *If an honest party p_i decides block B directly in epoch e_j of π_j, Δ_j holds, and all honest parties have $est < j$ then all honest parties that are awake at the end of epoch $e_j + \kappa$ will have decided B in π_j with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$.*

Proof. Because p_i decided B directly in π_j in epoch e_j, it follows from Lemma 4.3 that all honest parties awake at time t' after the end of epoch e_j have lock_j equal to a block extending B from time t' onward. By conditional safety (Lemma 4.4), no honest party decides a block conflicting with B in π_j. Furthermore, by Lemma 4.6 and the fact that all honest parties have $est < j$, in all subsequent epochs the blocks proposed by honest parties satisfy *predicate* for all awake honest parties (and therefore extend B).

From the validity property of GPE, it follows that with probability $\frac{1}{2} + \epsilon$ all awake honest parties output $(B', 1)$ from GPE (and therefore decide it) where B' is the input of an honest party and therefore extending B in each such epoch. It follows with the application of a Chernoff Bound that this occurs within κ epochs with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. □

Lemma 4.10. *If an honest party sets $est := j$ at time t and no honest party ever has $est + move > j$, then all honest parties awake at time $t' \geq t + 3\Delta_k$ onward have $est = j$ by time t'.*

Proof. This follows from the graded delivery and integrity properties of GA . □

We are now ready to prove that with overwhelming probability, no honest party ever sets $est > ideal$ for the duration of the protocol, which is crucial to ensuring that progress is made at the speed of $O(\Delta_{ideal})$.

Lemma 4.11. *No honest party ever sets $est > ideal$ except for probability $1 - \text{negl}(\kappa)$.*

Proof. From the protocol in Figure 3, an honest party must output a value from GA_B that is greater than $ideal$ in order to set est past $ideal$. By the integrity property of GA, this implies that some honest party input a value $j > ideal$ to GA. To do so, that party must already have $est > ideal$ or $est \leq ideal$ and $move \geq 1$.

Now, we show a proof by contradiction. Let p be the first party to set $est > ideal$ (if multiple did at the same time, choose one arbitrarily without loss of generality), and let t be the time at which they did so. This implies two possibilities prior to time t due to the integrity property of GA: (1) some honest party p' had $est = ideal$ and set $move := 1$ due to meeting one of the criteria for moving up or (2) some honest party had $est \leq ideal$ and $move > ideal - est$ due to outputting a value $v > ideal$ from GA_B in π_{beacon}.

We start by addressing the first case, and check if it is possible for some party p' to have had $est = ideal$ and set $move := 1$ due to meeting one of the criteria for moving up. Without loss of generality, let p' be the first party to have $est = ideal$ and $move := 1$. We go through each possible criteria for moving up to check if it's possible for p' to meet one of them at some time t^* prior to time t, assuming that none of the other criteria are met prior to time t^*. If none of the criteria are met in the absence of any of the other criteria already having been met, it follows that possibility (1) could not have happened.

We start with criterion 1 for moving up, and let $t' \leq t^* < t$ be the time at which p' set $est = ideal$. All honest parties awake at time $t' + 3\Delta_k$ onwards must have had $est = ideal$ by time $t' + 3\Delta_k$ by Lemma 4.10. By Lemma 4.8 there are κ epochs out of every 2κ epochs from time $t' + 3\Delta_k$ onwards in which all honest parties that are awake by the end of that epoch decide a new block proposed by an honest leader with probability $1 - e^{-2\kappa\epsilon^2}$, and this holds for every 2κ epoch period after that (as long as no honest party increments est past $ideal$ for any other reason).

Again, let $t' \leq t^* < t$ be the time at which p' set $est = ideal$. All honest parties awake at time $t' + 3\Delta_k$ onward must have had $est = ideal$ by time $t' + 3\Delta_k$ by Lemma 4.10 (prior to which this criterion could not have been met for p'). By Lemma 4.9, when an honest party decides a block in π_j for $j \geq ideal$ and all awake honest parties have $est = ideal$, all honest parties awake by the end of the subsequent κ epochs will have decided that block with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. By Lemma 4.7, once all honest parties awake at a certain time t decide a block in $\pi_{j \geq ideal}$, all honest parties awake at that point onward are locked on that block in π_{j+1} for all subsequent epochs beginning at time t onward. By Lemma 4.6 and Lemma 4.8, all honest parties awake at the end of the κ subsequent epochs of π_{j+1} will have decided that block with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. It follows from the fact that $\Delta_j = 2\Delta_{j-1}\ \forall 1 < j \leq k$ that any block decided in protocol $\pi_{j'}$ for $ideal \leq j < k$ all honest parties awake at the end of the subsequent 2κ epochs of $\pi_{j'+1}$ will have decided that block with high probability, in which case the criterion cannot be met in the absence of any of the other criterion having been met.

Finally, we address criterion 3. We start by checking whether the first part of the statement is possible: $\text{lock}_{j'}$ conflicts with the chain constructed in π_j for some $j \geq ideal$ and $j' > j$. Conditional safety (Lemma 4.4) applies for π_j since no honest party has $est > ideal$, and Lemma 4.2 ensures that $\text{lock}_{j'}$ was decided by an honest party in π_j. It follows that this part of the criterion cannot be met. The second part of the statement, that there is a fork in the chain constructed in π_j for $j \geq ideal$, also cannot be met due to conditional safety.

It therefore follows that option (1) could not have happened.

Next, we address (2): the possibility that p' had $est \leq ideal$ and $move > ideal - est$ due to outputting a value $v > ideal$ from GA in π_{beacon} prior to time t. For this to happen, p' must have output $v \geq ideal + 1$ from an instance of GA in π_{beacon} prior to time t. By the integrity property of GA, some honest party p'' must have input $v' \geq v$ to GA prior to time t. Without loss of generality, assume that p'' is the first party to have done so prior to time t. As already addressed when exploring possibility (1), it is not possible for p'' to have done so due to having $est = ideal$ and $move = 1$ from meeting the criteria for moving up with high probability. By the definition of p'', possibility 2 also cannot be because it would imply that a party input $v'' \geq v$ to GA prior to p'' doing so. We have arrived at a contradiction. □

In the following two lemmas, we prove that the protocol satisfies the properties of TOB.

Lemma 4.12 (Liveness). *Blocks containing transactions continue to be appended to $\Lambda_j \forall j \geq ideal$.*

Proof. If all awake honest parties have $est = j$ for $j \geq ideal$, the lemma follows from Lemma 4.8 and the fact that when an honest party decides a block in a sub-protocol π_j, they decide it to all $\Lambda_{j'}$ for $j' < j$. If all honest parties have $est < ideal$, this implies that some honest party p_i never meets any of the criteria for moving up past some sub-protocol π_j for $j < ideal$. This implies, by criterion 1 for moving up, that p_i sees κ new blocks decided every 2κ epochs to Λ_j. Furthermore, by criterion 2, every block decided by p_i to $\Lambda_{j'}$ is decided within the following 2κ epochs of $\pi_{j'+1}$ to $\Lambda_{j'+1} \forall j \leq j' < k$. For p_i, the fact that they don't meet criterion 3 for moving up implies that $\Lambda_{j'}$ and $\Lambda_{j''}$ do not conflict for any $j' \geq j$ and $j'' \geq j$. This follows from the fact that if they decide a block in any sub-protocol, they also update their lock for that sub-protocol so that it extends that block. It follows that every block that p_i decides to Λ_j, it also eventually decides to $\Lambda_{ideal,\ldots,k}$. □

Lemma 4.13 (Safety). *If two honest parties deliver logs $\Lambda_{j'}$ and $\Lambda'_{j'}$, then $\Lambda_{j'}$ and $\Lambda'_{j'}$ do not conflict $\forall \Delta_{j'} \geq \Delta_{ideal}$.*

Proof. The lemma follows from conditional safety (Lemma 4.4) and the fact that no honest party ever has $est > ideal$ (Lemma 4.11). □

Finally, we prove that blocks are decided at the desired rate in expectation.

Lemma 4.14. *Blocks are eventually decided at a rate of 1 block per $O(\Delta_{ideal})$ time in expectation in all $\pi_{j \geq ideal}$.*

Proof. To prove the lemma statement, we show that in all π_j for $ideal \leq j \leq k$, $2^{j-ideal}\kappa$ blocks are decided every 2κ epochs. Because $\Delta_j = \frac{\Delta_{j+1}}{2}$ $\forall j < k$, this implies the lemma statement.

First, we prove the claim that new blocks are decided at a rate of κ blocks every 2κ epochs in π_{ideal}. By Lemma 4.11, we know that all honest parties have $est \leq ideal$ for the duration of the protocol. If there eventually comes a time that all awake honest parties have $est = ideal$, the claim follows from Lemma 4.8. If this does not eventually happen, validity of GA implies that for every iteration of π_{beacon}, some honest party inputs $est + move < ideal$. Eventually, by validity of GPE, there must be some $j < ideal$ such that some honest party (note that it need not be the same one each time) inputs j to each iteration of π_{beacon} for the remainder of the protocol. Otherwise, at some point, all the awake honest parties *would* reach $est = ideal$.

Let $t^*_{1,j}$ be the earliest time at which an iteration of π_{beacon} output $(j, 1)$ for an honest party. By Lemma 4.5, all honest parties awake at time $t^*_{2,j} = t^*_{1,j} + 3\Delta_k$ must have output $(j, 1)$ by that time. Note that no honest party could output $(j', 1)$ for $j' > j$ from π_{beacon}, as this would contradict the definition of j by Lemma 4.5.

Let t be some time during the course of the protocol s.t. $t > t^*_{2,j}$. Let p be the party that input j to the latest instance of π_{beacon} that completed at or prior to time t with $move = 0$. This implies

that p never met any of the criteria for moving up since time $t_j^* \leq t_{2,j}^*$ (during which time it may have been awake or based on the state it acquired when joining the protocol). That is, p decides κ new blocks for every 2κ epochs of π_j from time $t_j^* + 3\Delta_k$ onward. Furthermore, every block it decides to $\Lambda_{j'}$, it decides within the subsequent 2κ epochs of $\pi_{j'+1}$ to $\Lambda_{j'+1}$. By Lemma 4.9, and the fact that $\Delta_{j+1} = 2\Delta_j \forall j < k$, it follows that in all π_j for $ideal \leq j \leq k$, all honest parties awake at the end of every 2κ epoch period will have decided κ new blocks with probability $1 - \mathsf{negl}(\kappa)$. □

Corollary 4.15. *If there exists some time t s.t. all honest parties awake at time t have $est = ideal$, then beginning at time t onward, all honest parties awake at the end of every 2κ epoch period will have decided κ new blocks and any block decided by an honest party in π_j for $ideal \leq j < k$ is decided by all honest parties in π_{j+1} that are awake at the end of the subsequent 2κ epochs of π_{j+1} with high probability.*

5 JOINING PARTIES

In this section, we present our joining protocol which enables us to remove the assumption that joining parties adopt the state of an already-awake honest party. In order to participate in the protocol, joining parties must learn must learn a log Λ_j and a lock lock_j for each sub-protocol π_j. As described in the introduction, joining parties need to also learn a value of *est* and *move* prior to beginning participation in the protocol so that they don't hinder liveness.

In the following text, we describe how the joining party obtains all of the necessary information through the joining protocol. Figure 4 presents the joining protocol executed by all awake parties and the joining party, p_{join}. The joining party obtains the following pieces of information:

- A For each sub-protocol π_j, a log Λ_j and a lock lock_j that captures the current state of the awake honest parties.
- B A value of *est* and *move* to begin with.
- C If it should set $move := 1$ prior to joining the protocol due to any of the criteria for moving up having already been met.

(A) For each sub-protocol π_j, p_{join} learns a log Λ_j and a lock lock_j that captures the current state of the awake honest parties.

Lines 5- 11 depict a series of graded agreements (GA) running continuously at $3\Delta_k$ intervals with a delay of Δ_k by all awake parties. Separate instances of $\mathsf{GA}_{high,j}$ and $\mathsf{GA}_{lock,j}$ are executed concurrently for each of the k sub-protocols. While only awake parties participate in these GA instances and send messages, p_{join} records their outputs as specified in line 7. The outputs of these GA's are used by p_{join} to obtain item A.

Lines 12- 20 depict actions done by p_{join}. On lines 18- 20, p_{join} uses the information that it learns from $\mathsf{GA}_{high,j}$ and $\mathsf{GA}_{lock,j}$ to adopt its log and lock for each sub-protocol (for learning item A). It uses the highest block that it outputs from each of these GAs. The integrity and validity properties of GA ensure that the conditional safety property of all $\pi_{j \geq ideal}$ continues to hold with the status adopted by p_{join}. p_{join}'s log corresponding to such sub-protocols will not conflict with the logs of any other honest parties. In addition, each of p_{join}'s logs for each such sub-protocol are a prefix of the log of some honest party awake at time ts, and the log of some honest party awake at time ts is a prefix of p_{join}'s log for each sub-protocol. These properties ensure that properties related to the safety of the main protocol remain intact. Furthermore, if all of the awake parties have decided a block in a given sub-protocol π_j, p_{join} will also have decided that block. The purpose of $\mathsf{GA}_{lock,j}$ is for p_{join} to learn which lock, lock_j, to adopt for each sub-protocol π_j. The validity property ensures that p_{join} chooses a lock that is at least as high as the highest common ancestor of the lock of all of the honest parties at the time at which the GA instances start. Since the main protocol has the

//*awake* parties:
1: **at** any time **do**
2: **upon** updating est to j for the first time **do**
3: set $\text{backlog}_{est} := height(\text{decide}_j)$ s.t.:
4: $j \geq est$ and $\forall j' > j$: decide_j does not conflict with $\Lambda_{j'}$ and $height(\text{decide}_j) \geq height(\text{decide}_{j'})$
5: **at** time $t = 3\Delta_k, 6\Delta_k, 9\Delta_k, \ldots$ **do**
6: **for** $j = 1, \ldots, k$ **do**
7: $\text{out}_{high,j} := \text{GA}_{high,j}(\text{decide}_j)$, $\text{out}_{lock,j} := \text{GA}_{lock,j}(\text{lock}_j)$ ▷ p_{join} stores output
8: $\text{out}_{backlog,j} := \text{GA}_{backlog,j}$ (backlog_j **if** $j \leq est$ **else** $height(\text{decide}_{j'})$) s.t.:
9: $j' \geq j$ and $\forall j'' > j'$: $\text{decide}_{j'}$ does not conflict with $\Lambda_{j''}$ and $height(\text{decide}_{j'}) \geq height(\text{decide}_{j''})$
10: **upon** receiving message $\langle \text{recovering}, p_{join}, ts \rangle_{join}$ from party p_{join} **do**
11: **at** time $t = ts + 3\Delta_k$ **do** send to p_{join} $\langle \text{status}, est, t^*_{est} \rangle_i$

//p_{join}:
12: set status to *recovering* and begin storing all messages received and the time they were received
13: send $\langle \text{recovering}, p_{join}, ts \rangle_{join}$ to all parties, where ts is the timestamp of the start of the next full iteration of π_{beacon}
14: **at** time $t = ts + 3\Delta_k$ **do**
15: set est and $move$ according to the outputs of GA in the iteration of π_{beacon} starting at time ts according to lines 7 and 9
16: **if** $\exists j \geq est$ s.t. there are any two conflicting outputs (of any grade) in $\text{out}_{high,j}$ **then**
17: set $move := 1$
18: set $B_{high,j \in [est,\ldots,k]} :=$ the highest block in $\text{out}_{high,j}$ (with any grade)
19: set $\text{lock}_{j \in [est,\ldots,k]} :=$ the highest block in $\text{out}_{lock,j}$ (with any grade)
20: **for** $j \geq est$ **do** set $\text{decide}_j := B_{high,j}$
21: **at** time $t = ts + 4\Delta_k$ **do**
22: let P be the list of parties from whom status messages were received
23: let M_{est} be the list of values of t^*_{est} received from every party $p_i \in P$ (for parties that sent a lower est than p_{join}'s, use t_{now}, the current timestamp)
24: $t^*_{est} := \text{median}(M_{est})$
25: set backlog_{est} := the highest block in $\text{out}_{backlog,est}$ (with any grade)
26: $min\_blocks_{est} := \max(\lfloor \frac{ts - (t^*_{est} + 3\Delta_k)}{20\kappa\Delta_{est}} \rfloor \kappa, 0)$
27: **for** $j = est + 1, \ldots, k$ **do**
28: set $min\_blocks_j := \max(min\_blocks_{j-1} - 2^{j - est}\kappa, 0)$
29: **if** $\exists j \geq est$ s.t. $ts > t^*_{est} + 3\Delta_k + \sum_{i=est}^{j} 20\kappa\Delta_i$ and $min\_blocks_j + \text{backlog}_{est} > |\Lambda_j|$ **then**
30: set $move := 1$
31: **for** block $B \in \Lambda_{est}[\text{backlog}_{est} + min\_blocks_{est} + 1 :]$ **do**
32: set $ts_{B,j} := ts$
33: **for** $j = est + 1, \ldots, k$ **do**
34: **for** block $B \in \Lambda_j$ **do** set $ts_{B,j} := ts$
35: **upon** completion of an epoch of π_k since setting status to *recovering* **do**:
36: update state based on simulation of sub-protocols $\pi_{est,\ldots,k}$ and π_{beacon} using stored messages
37: begin checking criterion 1 for moving up from time $t^*_{est} + 3\Delta_k + 20\kappa\Delta_{est} \lfloor \frac{ts - (t^*_{est} + 3\Delta_k)}{20\kappa\Delta_{est}} \rfloor$ onward
38: begin checking criterion 2 for moving up for blocks in $\text{diff}(\Lambda_j, \Lambda_{j+1}) \forall est \leq j < k$
39: begin checking criterion 3 for moving up
40: set status to *awake*

Fig. 4. Joining Protocol for *awake* parties p_i and *recovering* party p_{join} executed with delay of Δ_k

property that if an honest party decides a block in sub-protocol π_j, all awake honest parties lock on it (assuming that Δ_j holds), this ensures that p_{join} does not disrupt the safety of any sub-protocol (or the larger protocol), as its lock incorporates any block decided by any honest party.

(B) p_{join} learns a value of est and $move$ to begin with.

By listening to an instance of π_{beacon}, p_{join} sets its values of est and $move$ for the first time (line 15).

(C) p_{join} learns if it should set $move := 1$ prior to joining the protocol due to any of the criteria for moving up having already been met.

Recall that there are 3 criteria for moving up which cause a party to set $move := 1$:

(1) **No progress in π_{est}.** Beginning at time $t^*_{est} + 3\Delta_k$ or later, a 2κ epoch period elapses in π_{est} during which fewer than κ blocks are decided
(2) **No confirmations in higher sub-protocols.** At time $t^*_{est} + 3\Delta_k$ or later for any j' s.t. $est \leq j' < k$ a block B is decided in $\pi_{j'}$ and more than 2κ full epochs of $\pi_{j'+1}$ elapse after $ts_{B,j'}$ without B having been decided in $\pi_{j'+1}$
(3) **Observing equivocations in higher sub-protocols.** $\text{lock}_{j'}$ conflicts with the chain constructed in π_j for some $j' > j$ or there is a fork in the chain constructed in π_j for some $j \geq est$

For detecting criterion 1, p_{join} must detect if Λ_{est} is long enough to reflect κ blocks having been committed every 2κ epochs since the time that parties first updated their values of est to this value. For this, it must learn the length of the logs that parties had prior to updating their values of est to this value, which we refer to as a backlog. Additionally, p_{join} must learn the time at which parties first set est equal to this value, t^*_{est}.

On line 4, awake parties update their backlogs each time they update their value of est. $backlog_j$ represents the blocks that the party already committed to Λ_j prior to setting $est = j$. They update $backlog_j$ to the height of the head of any of their chains $\Lambda_{j' \geq j}$ such that the block does not conflict with any chains $\Lambda_{j'' > j'}$. The reason why they set $backlog_j$ this way (rather than just setting it equal to the length of Λ_j) is that if a party commits a block in $\pi_{j' > j}$, some party must have already committed it in π_j if Δ_j holds. This helps ensure that p_{join} learns the largest backlog possible. p_{join} learns its backlogs via $\text{GA}_{backlog,j}$. The awake parties input to $\text{GA}_{backlog,j}$ $backlog_j$ if $j \leq est$, and $height(\text{decide}_{j'})$ otherwise, where $j' \geq j$ and $\text{decide}_{j'}$ does not conflict with $\Lambda_{j'' \forall j'' \geq j'}$. Note that we assume for $\text{GA}_{backlog,j}$ that the set of non-negative integers form a chain, where $i + 1$ extends i. p_{join} sets $backlog_{est}$ based on the output of $\text{GA}_{backlog,est}$ on line 25.

p_{join} learns t^*_{est} as follows. On line 13, p_{join} sends to all of the awake parties a message $\langle \text{recovering}, p_{join}, ts \rangle_{join}$. In this message, p_{join} includes a timestamp, ts, of the start of the next full iteration of π_{beacon}. The inclusion of ts is necessary because at the completion of the instance of π_{beacon} beginning at ts, parties must send to p_{join} information relevant to the value of est that they ended the instance with. Lines 10-11 illustrate the actions of awake parties upon receiving the message $\langle \text{recovering}, p_{join}, ts \rangle_{join}$ from p_{join}. After completing one full iteration of π_{beacon} starting at time ts, the awake party p_i sends p_{join} a message $\langle \text{status}, est, t^*_{est} \rangle_i$, which includes (1) its current value of est and (2) t^*_{est}, the time of its last est update. On lines 21- 24, p_{join} uses the messages it received from the awake parties to calculate t^*_{est}. It compiles the values of t^*_{est} reported by each of the responding parties that reported having the same value of est, adding t_{now} (the current timestamp) for each party that sent a different value of est. It sets t^*_{est} to the median of these values. The purpose of calculating t^*_{est} is that p_{join} must verify that sufficient progress has been made since the honest parties originally adopted this value of est; otherwise we run into the liveness problem described in the introduction. Note also that if an honest party has a greater value of est than p_{join}'s, these calculations are redundant, because the graded delivery and validity properties of GA ensure that p_{join} will update its values of est and $move$ with the next iteration of π_{beacon}. If an honest party has a lower value of est, the current timestamp is a correct lower bound for the time at which that party will update est to this value. This method of calculating t^*_{est} ensures that (when this calculation is necessary rather than redundant), t^*_{est} is no lower than the time at which

the first honest party to adopt this value of est did so, and is at most $3\Delta_k$ higher than that. Since there is up to a $3\Delta_k$ latency in the time between two honest parties that are awake adopting the same value of est (by Lemma 4.5), p_{join} adopts a "valid" value for t^*_{est}.

p_{join} uses t^*_{est} and $backlog_j$ for $j \geq est$ to do calculations to detect if criteria 1 or 2 for setting $move := 1$ have been met. That is, it should see the equivalent of κ blocks committed every 2κ epochs in π_{est} from time $t^*_{est} + 3\Delta_k$ onward, and all but the most recent blocks in Λ_j should have been decided in Λ_{j+1} for $est \leq j < k$. To check this, p_{join} calculates $min\_blocks_{j \in est,\ldots,k}$, the minimum number of blocks it should have committed to $\Lambda_{j \in est,\ldots,k}$ when creating its logs on line 20 if criteria 1 and 2 for moving up have *not* been met. min_blocks_{est} is calculated as κ blocks for every 2κ epoch period after $t^*_{est} + 3\Delta_k$. p_{join} sets $min\_blocks_j := \max(min\_blocks_{j-1} - 2^{j-est}\kappa, 0)$. This is an approximation to reflect criterion 2: a block decided in π_j should be decided in π_{j+1} within the subsequent 2κ epochs. On line 30, if fewer than $min\_blocks_j + backlog_{est}$ blocks were decided to any log on line 20, p_{join} sets $move := 1$.

On line 34, p_{join} sets a commit time $t_{B,j}$ for blocks B committed to Λ_j in excess of $min\_blocks_{j+1} + backlog_{est}$. These are blocks it has not yet committed to Λ_{j+1}, and it needs to ensure that these blocks are committed in π_{j+1} within subsequent 2κ epochs after $t_{B,j}$, as per criterion 2. While $t_{B,j}$ is only guaranteed to be lower bounded by the actual time at which an awake honest party committed B in π_j, it is sufficient for ensuring that progress is made at the desired amortized rate. This is because up to the end of the last full 2κ epoch period prior to ts, sufficiently many blocks have been decided from p_{join}'s perspective. If p_{join} lets these blocks in excess of min_blocks_j contribute to the next 2κ epoch period (which must contain timestamp ts), even if they were actually decided by an honest party prior to this period, because the previous periods already saw enough progress without these blocks, the amortized rate is as needed. Furthermore, the blocks that p_{join} decides due to messages received since the time it set its state to recovering *will* contain the actual timestamp of when they were decided by an honest party. Deciding a block B implies deciding all blocks that it extends, so p_{join} deciding κ blocks every 2κ epochs in π_{est} and not meeting criteria 2 for moving up is sufficient.

For criterion 3, on lines 16- 17, p_{join} detects evidence of a fork in the logs constructed by any of the honest parties for $\pi_{j \geq est}$, and if so, uses this as evidence that est does not hold and sets $move := 1$.

Lastly, before setting its status to awake, p_{join} must update its state based on all messages it received while in the recovering state. Upon setting its status to recovering when it first starts the joining protocol (line 12), p_{join} begins storing all messages that it receives and the time at which they were received. Upon completion of an entire epoch of π_k since beginning the joining protocol, p_{join} updates its entire state (except for updates to $move$ due to meeting any criteria for moving up) by simulating the protocol upon the state that it has adopted thus far on line 36. Note that this may include deciding blocks, updating locks, and updating est and $move$ (due to π_{beacon}). It is important to note that simulation of the protocol is only to learn what may have happened since p_{join} started the joining protocol; p_{join} does send any messages in the simulation; it just calculates changes to its state based on the saved messages. On line 37, p_{join} starts checking if any of the criteria for moving up are met, from a specific point in time: $t^*_{est} + 3\Delta_k + 20\kappa\Delta_{est} \lfloor \frac{ts - (t^*_{est} + 3\Delta_k)}{20\kappa\Delta_{est}} \rfloor$ onward. This time is calculated to begin immediately after the end of the last 2κ epoch period which was covered by min_blocks_{est} (if there was one), and therefore already verified to pass the check. On line 38, p_{join} begins checking criterion 2 for moving up with the blocks that have been decided to a log Λ_j but not to the log of the sub-protocol immediately higher, Λ_{j+1}. On 40, p_{join} begins checking criterion 3. Finally, p_{join} sets its status to *awake*.

The backlog While we cannot ensure that the backlog learned by p_{join} is equal to the largest backlog of any honest party, it is not very far off. Intuitively, the reason is that if an honest party has committed many blocks to Λ_j prior to setting $est = j$, and it continues to believe that $est = ideal$ (meaning it does not set $move := 1$), then the blocks that it commits in π_j, it also commits in all higher sub-protocols, including in π_{ideal}. If one honest party commits a block in π_{ideal}, all of the awake honest parties commit it within the κ subsequent epochs. Because of the way honest parties set backlog_j and decide what to input to $\text{GA}_{backlog,j}$, it follows that the backlog learned by a joining party cannot be very far off from the backlog held by any other honest party that is seeing progress in π_j. We formalize this intuition in Appendix B when analyzing the full protocol.

5.1 Recovery Protocol Analysis

We now present the analysis of the recovery protocol. Our analysis of the full TOB protocol may be found in Appendix B. Our approach in this section is that we first show that the state adopted by p_{join} is *safety-preserving*. That is, for all sub-protocols in which the delay holds, p_{join} adopts a chain that includes the highest decided block at time ts of some honest party that is awake at time ts and that its lock for each such sub-protocol is at least the highest common ancestor of the lock at time ts of all honest parties awake at time ts.

Lemma 5.1. *Let $\Lambda_{j,ts,join}$ be the chain adopted by p_{join} on line 20 for sub-protocol π_j and $\text{lock}_{j,ts,join}$ be the lock adopted for the same sub-protocol by p_{join} on line 19. Similarly, let $\Lambda_{j,ts,\alpha}$ be Λ_j for party p_α awake at time ts. The following guarantees hold for all $j \geq ideal$ if no two awake honest parties have conflicting logs for sub-protocol π_j at time ts:*

(1) $\Lambda_{j,ts,join}$ is a prefix of $\Lambda_{j,ts,\alpha}$ for some honest party p_α awake at time ts, and $\Lambda_{j,ts,\beta}$ of some honest party p_β awake at time ts is a prefix of $\Lambda_{j,ts,join}$.

(2) $\text{lock}_{j,ts,join}$ extends the highest common ancestor of lock_j for all honest parties awake at time ts.

Proof. We start by proving the first of the two guarantees. First, we show that $\Lambda_{j,ts,join}$ is a prefix of $\Lambda_{j,ts,\alpha}$ for some honest party p_α awake at time ts. This follows from the integrity property of GA, which applies to $\text{GA}_{high,j}$. Next, we show that $\Lambda_{j,ts,\beta}$ of some honest party p_β awake at time ts is a prefix of $\Lambda_{j,ts,join}$. Since no two honest parties have conflicting chains Λ_j for sub-protocol π_j at time ts, the validity property of GA ensures that the highest decided block for some honest party at time ts in sub-protocol π_j must be in $\text{out}_{j,high}$ for p_{join}, from which the statement follows. The second guarantee follows from the validity property of GA. □

From this lemma and the properties proved for the main protocol in the previous section, we derive the following corollaries which are analogous to Lemmas 4.3, 4.4, and Lemma 4.7 for the state adopted by p_{join}.

Corollary 5.2. *If an honest party directly decides block B in epoch e of π_j, Δ_j holds, and all honest parties have $est \leq j$, then p_{join} is locked on a block B' extending B for all subsequent epochs of π_j from time $ts + 3\Delta_k$ onwards.*

Corollary 5.3. *If no honest party has $est > j$ and Δ_j holds, then if an honest party decides block B in protocol π_j, p_{join} never decides a block B' conflicting with B.*

Corollary 5.4. *If Δ_j holds, no honest party has $est > j$, $j < k$, and all honest parties awake at time t have decided B (or a block extending it) in π_j by time t, then from time $max(t, ts + 3\Delta_k)$ onward, p_{join} has lock_{j+1} equal to a block extending B for all epochs of π_{j+1}.*

In the following lemma we show that the value t^*_{est} adopted by p_{est} is sufficiently close to the value of t^*_{est} held by the first honest party to adopt that value of est.

Lemma 5.5. *Let $t^*_{1,j}$ be the first time at which the GA in π_{beacon} output $(j, 1)$ for an honest party. If no honest party has est $> j$ by time $ts + 3\Delta_k$, then $t^*_{1,j} \leq t^*_j \leq t^*_{1,j} + 3\Delta_k$, where t^*_j is the value set by p_{join} on line 24.*

Proof. If $t^*_{1,j}$ is at or prior to ts, the result follows from the graded delivery property of GA (which ensures that all honest parties that were awake at the time had $est = j$ by time $t^*_{1,j} + 3\Delta_k$) and the fact that p_{join} takes the median of the values it receives in all of the messages, a majority of which must be honest parties. Otherwise, the result follows from the fact that $t^*_{1,k=j}$ must be equal to t^*_j because no honest party has set $est = j$ prior to p_{join} doing so. □

In Appendix B, we prove that the full protocol satisfies the properties of TOB and that blocks are eventually committed at the desired rate.

5.2 Tolerating Backward Simulation

To determine T_b, we start by enumerating the cases in which parties use information from previously executed instances of GA and GPE for each sub-protocol below:

(1) As input to GA_0 the parties need the highest committed block from the previous sub-protocol and the lock from the current sub-protocol.
(2) As input to GA_1, parties need their output from GA_0.
(3) As input to, and for evaluating the predicate within, GPE, parties need their outputs from GA_0 and GA_1.

For item 2, parties use as input to GA_1 their output from GA_0 immediately after it is produced. For item 3, parties use their outputs from GA_0 and GA_1 throughout the instance of GPE (in the same epoch). This information is generated at most $7\Delta_j$ earlier for sub-protocol π_j. For item 1, parties use information that may have been generated epochs ago. While the lock can be generated, at the earliest, in the previous epoch, the highest committed block of the lower sub-protocol could have been generated long ago. However, the joining protocol ensures that when a party sets their status to *awake*, they have this information if necessary. If a party was *awake* at the time of committing their highest committed block in π_{j-1}, they have this information when it's needed in all subsequent epochs of π_j If the party was not *awake*, then they learned the highest committed block of some honest party in π_{j-1} and lock_j from $\mathsf{GA}_{high,j}$ and $\mathsf{GA}_{lock,j}$, respectively. Because they listen for an entire epoch of π_k (and all epochs of the lower sub-protocols that elapse in that time) prior to setting their status to *awake*, if this information changed since the point at which those instances of $\mathsf{GA}_{high,j}$ and $\mathsf{GA}_{lock,j}$ began, the joining party is able to compute the information itself. Since a party doesn't join the protocol in the middle of an epoch, they have the necessary information for participating in GPE because they were awake during the instances of GA_0 and GA_1 in which it was calculated and were able to calculate it themselves.

Since parties wait for the duration of an entire epoch of π_k before setting their status to *awake*, it is necessary that the adversarial fraction grows at a delay that accounts for this. It follows that if $T_b = 20\Delta_k$, that is, the adversarial fraction grows proportionate to the overall participation at a delay of $20\Delta_k$.

6 ACKNOWLEDGMENT

This work is funded in part by an academic grant from the Ethereum Foundation. We would like to thank Francesco D'Amato and Aditya Asgaonkar for their inputs at the early stages of this work.

REFERENCES

[1] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.

[2] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 585–602, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Repeated agreement is cheap! on weak accountability and multishot byzantine agreement. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 15–27, 2025.

[4] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 23–41. Springer, 2019.

[5] Francesco D'Amato, Giuliano Losa, and Luca Zanolini. Asynchrony-resilient sleepy total-order broadcast protocols. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 247–256. ACM, 2024.

[6] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on ethereum?! In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part I*, volume 14744 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2024.

[7] Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. Tob-svd: Total-order broadcast with single-vote decisions in the sleepy model, 2024.

[8] Francesco D'Amato and Luca Zanolini. Recent latest message driven GHOST: balancing dynamic availability with asynchrony resilience. In *37th IEEE Computer Security Foundations Symposium, CSF 2024, Enschede, Netherlands, July 8-12, 2024*, pages 127–142. IEEE, 2024.

[9] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[10] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[11] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptol. ePrint Arch.*, 2018:1119, 2018.

[12] Eli Gafni and Giuliano Losa. Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 41:1–41:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[13] Vipul Goyal, Hanjun Li, and Justin Raizes. Instant block confirmation in the sleepy model. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II*, page 65–83, Berlin, Heidelberg, 2021. Springer-Verlag.

[14] Daniel Kane, Andreas Fackler, Adam Gagol, and Damian Straszak. Highway: Efficient consensus with flexible finality. *arXiv preprint arXiv:2101.02159*, 2021.

[15] Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting. In Foteini Baldimtsi and Christian Cachin, editors, *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023, Revised Selected Papers, Part I*, volume 13950 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2023.

[16] Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *CoRR*, abs/2304.14701, 2023.

[17] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine consensus under fully fluctuating participation. *IACR Cryptol. ePrint Arch.*, page 1448, 2022.

[18] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy bft. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 490–503, 2023.

[19] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1041–1053, 2019.

[20] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2295–2308, 2022.

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.

[22] Joachim Neu, Srivatsan Sridhar, Lei Yang, and David Tse. Optimal flexible consensus and its application to ethereum. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3885–3903. IEEE, 2024.

[23] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 380–409. Springer, 2017.

[24] Zhuolun Xiang, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Strengthened fault tolerance in byzantine fault tolerant replication. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 205–215. IEEE, 2021.

[25] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105, 2020.

A RELATED WORK

In recent years, research on distributed consensus has evolved to address the challenges posed by large-scale systems such as blockchain networks. One of the primary challenges is *dynamic participation*, where parties in a Total-Order Broadcast (TOB) protocol may go offline and return at any point during execution. This concept was first introduced by Bitcoin [21], which allowed parties to join the network at any time while assuming that a majority of the total computational power (hash rate) remains honest. Such protocols are called dynamically available.

Building on this idea, Pass and Shi [23] formally characterized dynamic participation through the sleepy model, where protocol execution occurs in discrete rounds, allowing parties to wake up or go offline freely. Their model assumes that in each round, a majority of online parties are honest. They also proposed a longest-chain consensus protocol tailored for this setting, inspiring further research in the field.

Later works aimed to improve latency and adaptability under dynamic participation. Momose and Ren [20] laid the foundation for deterministically safe, dynamically available TOB protocols. They proposed a protocol with constant latency in the Sleepy Model, leveraging a *Graded Agreement* primitive and introducing the *time-shifted quorum* technique to handle fluctuating participation levels. This technique redefines quorum configurations dynamically to address challenges in forwarding quorum certificates. However, the protocol incurred a relatively high latency of 16Δ, limiting its practicality.

Malkhi, Momose, and Ren [18] improved upon this by proposing a deterministically safe TOB protocol with a best-case latency of 4Δ, using Graded Agreement and a new *Graded Proposal Election* primitive while maintaining minority corruption tolerance. Their earlier work [17], partially detailed in Appendix A of [18], presented TOB protocols achieving 3Δ latency with 1/3 fault tolerance and 2Δ latency with 1/4 fault tolerance. In parallel, Gafni and Losa [12] introduced two TOB protocols within the sleepy model that also tolerate minority corruption. The first guarantees deterministic safety and probabilistic liveness with constant expected latency, while the second provides both deterministic safety and liveness.

D'Amato *et al.* [7] presented TOB-SDV, another deterministically safe TOB protocol in the sleepy model that tolerates up to 50% adversarial corruption. This protocol improves upon prior work by requiring only a single voting phase per decision in the best case and achieving lower expected latency than existing approaches with the same optimal adversarial resilience.

Beyond deterministic safety, research has also explored *probabilistically safe* protocols, i.e., those that guarantee safety with overwhelming probability under dynamic availability. Examples include Goldfish [6] and RLMD-GHOST [8], which apply chain-based strategies rather than quorum-based decision-making.

A key distinction between the protocols discussed above lies in the nature of their safety guarantees. Deterministically safe protocols ensure that no two honest parties ever deliver conflicting values. In contrast, probabilistically safe protocols guarantee safety with overwhelming probability: forks may occur but become unlikely over time. This distinction is particularly relevant in the context of permissionless networks. While probabilistically safe protocols can, in principle, be

designed to operate without knowledge of the parties set, the same is not true for deterministically safe protocols, which inherently rely on quorums and thus require a known set of parties.

All the protocols discussed above assume network synchrony. This stems from an impossibility result showing that dynamically available protocols cannot be achieved in partially synchronous or asynchronous networks [16]. A few works attempt to circumvent this limitation [5, 6, 8] by introducing the notion of messages with expiration periods. D'Amato and Zanolini [8], and later D'Amato, Losa, and Zanolini [5] propose mechanisms for tolerating bounded periods of asynchrony in dynamically available TOB protocols ensuring safety probabilistically and deterministically, respectively. These approaches enable the selection of a small known synchrony bound, allowing the protocol to achieve low latency under normal conditions, while remaining resilient to occasional asynchrony of a larger bounded duration π.

Their approach differs from ours in that it assumes a fixed network delay, except during explicitly bounded periods of asynchrony, and progresses in discrete time steps. In contrast, our protocol operates over varying message delays and gradually stabilizes to the actual network delay.

While their protocol maintains prefix safety, safety can be violated during periods where the delay exceeds the bound π, as in our model when exceeding Δ. The key distinction lies in how adversarial resilience is handled. Their protocol relies on message expiration to cope with asynchrony, which reduces resilience: since messages from honest-but-asleep parties remain valid until expiry, an adversary can exploit these outdated votes. As a result, all such asleep-but-unexpired parties must effectively be treated as Byzantine.

In our approach, adversarial resilience is solely determined by the pessimistic bound Δ, without needing to discount asleep parties, thus achieving stronger resilience in dynamic settings.

On the topic of acheiving latency better than that of the pessimistic assumption on the network delay, recent work by Civit, Dzulfikar, Gilbert, Guerraoui, Komatovic, and Vidigueira have taken the approach of detecting and eliminating malicious parties [3]; however, this work is not in the dynamically available setting.

In flexible consensus, as in our work, different users express different beliefs over the network delay and get a secure protocol with latency commensurate to that network delay *if* their assumption is correct [14, 19, 22, 24]. Flexible consensus was first introduced in [19]. [22] presents a modular construction that may be added on to an existing consensus protocol for enabling clients to opt for higher safety and resilience at the expense of reduced liveness. [24] improves upon prior works by presenting a flexible BFT protocol with linear message complexity and only marginal bookkeeping overhead.

B FULL PROTOCOL ANALYSIS

In the following, we refer to the protocol in Figure 2 as the *main protocol* and the protocol in Figure 4 as the *joining protocol.*

Lemma B.1. *If $j \geq ideal$, and all honest parties have $est \leq j$, then for all $j' > j$, $\mathsf{lock}_{j'}$ held by any honest party was decided by an honest party in π_j.*

Proof. There are 4 points at which lock updates occur: in the main protocol, there are those on lines 15 and 25. We refer to these as cases (i) and (ii) respectively. In the joining protocol, there are the lock updates on line 19 and 36. We refer to these as cases (iii) and (iv) respectively.

We prove this lemma by induction on j, demonstrating that the statement holds for $j' = j + 1$.

We start with cases (i), (ii), and (iv), and we use a proof by induction. It is evident that the statement holds at the beginning of the protocol. Now, assuming it holds at the start of epoch $e_{j'} \geq 0$ of $\pi_{j'}$, we aim to prove that it continues to hold at the start of epoch $e_{j'} + 1$ (for cases (i), (ii), and (iv)).

First, we address case (i). At the start of epoch $e_{j'}$, honest parties input to $\mathsf{GA}_{0,j'}$ the highest block they decided in π_j (if it extends $\mathsf{lock}_{j'}$) or $\mathsf{lock}_{j'}$. All honest parties input a block that was decided by an honest party in π_j. This follows directly from the inductive assumption. The honest parties then update $\mathsf{lock}_{j'}$ to the highest grade 1 output from $\mathsf{GA}_{1,j'}$ (line 15). The lemma holds for these lock updates because of the integrity property of GA and the fact that honest parties chose their inputs to $\mathsf{GA}_{1,j'}$ from their outputs from $\mathsf{GA}_{0,j'}$.

Next, we address case (ii). A necessary condition for honest parties (all having $est < j'$ by the lemma statement) to evaluate predicate to true on a block in GPE of $\pi_{j'}$ is that they output it (or a block extending it) with any grade from $\mathsf{GA}_{0,j'}$. This implies that if the leader's proposal succeeds, i.e., some honest parties output a non-$\perp$ block B from GPE, then some honest party decided the block B in π_j (by the integrity property of GA). The lemma therefore follows for lock updates after GPE outputs (line 25), case (ii).

Finally, observe that the logic for case (iv) is the same as the logic for cases (i) and (ii); the only difference is that the locking party has status *recovering*, but this does not change the argument. It follows that the lemma statement holds for $j' = j + 1$ and for lock updates due to cases (i), (ii), and (iv.)

Next, we address case (iii). We do not use a proof by induction for this case; all we need to show is that if an honest party locks on a block due to case (iii), some honest party already locked on it due to cases (i), (ii) or (iv) prior to that time; since we showed that the statement holds for lock updates due to cases (i), (ii) and (iv), this proves the statement for case (iii). If an honest party sets lock_j to a block B due to case (iii), it must have output that block from $\mathsf{GA}_{lock,j}$. It follows from the integrity property of GA that some honest party must have input a block extending B to that instance of $\mathsf{GA}_{lock,j}$, and therefore have had lock_j extending B. By the integrity property of GA, the first party to have set lock_j equal to a block B' extending B couldn't have done so due to case (iii), as it had to have output B' from $\mathsf{GA}_{lock,j}$.

We have thus proven the statement for $j' = j + 1$. The the lemma therefore holds for every two consecutive pairs of sub-protocols where both are at least j.

Next, we address general $j \geq ideal$ and $j' > j$. To do so, we will use the fact that we proved the statement for $j' = j + 1$ already. If an honest party sets $\mathsf{lock}_{j'} = B$ in $\pi_{j'}$ at time t, then some honest party decided it in $\pi_{j'-1}$ prior to time t, which follows from the fact that the lemma holds for $j' = j + 1$. Since an honest party decided B in $\pi_{j'-1}$, they must have locked on it as well. This follows directly from the fact that if an honest party decides a block due to line 26 of the main protocol, then they lock on it on line 25 [9]. Otherwise, they must have decided it on line 20 of the joining protocol, in which case it follows from the integrity property of GA, which ensures that some awake honest party must have already decided that block and input it to $\mathsf{GA}_{high,j}$ (implying that the *first* honest party to decide the block must have done so due to line 26 of the main protocol and therefore locked on it on line 25). It follows from the fact that the lemma holds for $j' = j + 1$ that some honest party must have decided B in $\pi_{j'-2}$ prior to this. Using this line of reasoning, we see that the lemma holds for general $j \geq ideal$ and $j' > j$. □

We say an honest party decides a block *indirectly* if it decides it on line 20 of the joining protocol; otherwise, we say that it decides it *directly*.

Lemma B.2. *If an honest party decides block B directly in epoch e_j of π_j, Δ_j holds, and all honest parties have $est \leq j$, then all honest parties awake at any time t' in any epoch $e'_j > e_j$ have* lock_j *extending B from time t' onward.*

[9] Note that this could also be when they are simulating the protocol during the joining protocol.

Proof. Note that all honest parties awake or in the recovering state at the time that the GPE of epoch e_j of π_j outputs update lock_j to B due to graded delivery of GPE (line 25). We need to show that all honest parties that are awake at any time t' during an epoch $e'_j > e_j$ also have lock_j equal to a block extending B. We therefore only need to prove that no awake or recovering honest party locks on a block that doesn't extend B in any epoch after e_j.

We use a proof by induction to show that the lemma holds for all epochs after epoch e_j. We already showed that the lemma holds at the end of epoch e_j, as a base case. Assume that it continues to hold up to the end of some epoch $e'_j \geq e_j$. We show that it holds up to the end of epoch $e'_j + 1$.

In epoch $e'_j + 1$ of π_j, all honest awake parties input to $\mathsf{GA}_{0,j}$ their lock_j or their highest decided block from π_{j-1} if it extends lock_j. In either case, they input a block extending B. It follows from integrity and validity that all honest awake parties input a block extending B. By integrity, no block output by an honest party (whether they are awake or simulating the protocol) from $\mathsf{GA}_{0,j}$ or $\mathsf{GA}_{1,j}$ may conflict with B, and by validity, the lock of all honest parties set on line 15 extends B. By validity of GPE, if an honest party updates their lock due to outputting a block from GPE, then the block must extend B. Note that by the condition on line 35 of the joining protocol, all honest parties that join the protocol do so only after listening to (and therefore simulating) the protocol for one full epoch of π_k (which must include one full epoch of all $\pi_{j'<k}$), so what we have shown implies that all honest parties in recovery during epoch $e'_j + 1$ set lock_j equal to a block extending B during simulation of the protocol. It follows that all honest parties that are awake at the end of epoch $e'_j + 1$ have lock_j extending B. □

Lemma B.3 (Conditional Safety). *If no honest party has est $> j$ and Δ_j holds, then if an honest party decides block B in protocol π_j, no honest party decides a block B' conflicting with B in π_j.*

Proof. Suppose an honest party h decides block B in protocol π_j in epoch e_j. We need to show that an honest party h' cannot decide on a conflicting block B' in π_j at any time, whether directly or indirectly. Assume that h' decides block B' conflicting with B directly in epoch e'_j; we will show that this implies a contradiction.

First, note that it cannot be the case that $e'_j = e_j$. This is because of the consistency property of the execution of GPE in epoch e_j. Without loss of generality, let us assume $e_j < e'_j$. Also, since we have $est \leq j$, by Lemma B.2 all honest parties awake at any time $t' \geq t$ have lock_j extending B from time t' onwards. A necessary condition for parties to decide on a leader's proposal in GPE (i.e., evaluate predicate = *true* for the proposal) is if it extends their current lock. Thus, since all awake honest parties have lock_j extending B, no party would directly decide on a block conflicting with it (by the validity property of GPE). Thus, h' will not decide B' directly in any epoch e'.

Next, we show that no honest party h' decides a block B' conflicting with B *indirectly* after epoch e. For this to happen h' must output B' or a block extending it from $\mathsf{GA}_{high,j}$. By integrity of GA, this implies that some awake honest party input to $\mathsf{GA}_{high,j}$ B' or a block extending it. We already showed that no honest party directly decides a block B' conflicting with B. So it must be the case that the honest party that input B' (or a block extending it) to $\mathsf{GA}_{high,j}$ decided it indirectly. W.l.o.g. let h'' be the first honest party to decide a block B' conflicting with B (note that it must be indirectly decided). It must have output it from $\mathsf{GA}_{high,j}$, which implies a contradiction because it means some other honest party must have decided B' before h'' by integrity of GA. □

Lemma B.4. *If Δ_j holds and no honest party has est $> j$, then B_{cand} of every honest party during epoch e_j awake at time $(10e_j + 6)\Delta_j$ extends lock_j of all honest parties awake at that time, and a block extending B_{cand} was output by all honest parties awake at any point during epoch e_j from time $(10e_j + 3)\Delta_j$ onward from $\mathsf{GA}_{0,j}$ of that epoch.*

Proof. Because honest parties input to $\mathsf{GA}_{1,j}$ their highest grade 1 output from $\mathsf{GA}_{0,j}$ such that there were no conflicting outputs, it must be the case that none of the outputs of $\mathsf{GA}_{1,j}$ of honest parties conflict during epoch e_j. This follows from the graded delivery and integrity properties of GA. Parties set their lock_j to be the highest grade 1 output from $\mathsf{GA}_{1,j}$. The graded delivery property of GA ensures that all honest parties output the lock_j of all other honest parties, at least with grade 0. Moreover, B_{cand} is chosen to be the block with the greatest height from the output of $\mathsf{GA}_{0,j}$. Since, the outputs of $\mathsf{GA}_{0,j}$ do not conflict, B_{cand} set by all honest parties awake at time $(10e_j + 6)\Delta_j$ must either be equal to lock_j or a block that extends lock_j for all honest parties that are awake at that time. This proves the first part of the lemma.

To prove the second part of the lemma, recall that parties choose as B_{cand} their highest output of any grade from $\mathsf{GA}_{1,j}$. The integrity property implies that some honest party must have input a block extending B_{cand} to $\mathsf{GA}_{1,j}$ at time $(10e_j + 3)\Delta_j$. Honest parties choose as B_{cand} their highest grade 1 output from $\mathsf{GA}_{0,j}$ (with no conflicting outputs). The graded delivery property ensures that if an honest party outputs a block with grade 1, all honest parties output it with at least grade 0. □

Lemma B.5. *If Δ_j holds, no honest party has est $> j$, $j < k$, and all honest parties awake at time t have decided B (or a block extending it) in π_j by time t, then all honest parties awake at any point at or after t have* lock_{j+1} *equal to a block extending B for all epochs of π_{j+1} beginning at or after time t.*

Proof. By conditional safety (Lemma B.3), no honest party decides a block conflicting with B in π_j. By conditional safety and Lemma B.1, it follows that no honest party ever has lock_{j+1} conflicting with B. From the way that parties choose their inputs to $\mathsf{GA}_{0,j+1}$ on lines 9- 10, it follows that all honest inputs to $\mathsf{GA}_{0,j+1}$ of π_{j+1} are B or a block extending it for all epochs beginning at or after time t.

In every epoch of π_{j+1} starting after time t, by the validity and integrity properties of GA, no honest party outputs a block conflicting with B from $\mathsf{GA}_{0,j+1}$ and all honest parties that output from $\mathsf{GA}_{0,j+1}$ (i.e. all honest parties that are awake or in the recovering state at that time) do so with B (or a block extending it) with grade 1. As a result, all awake honest parties input a block extending B to $\mathsf{GA}_{1,j+1}$.The lemma therefore follows for lock updates prior to GPE due to the integrity and validity properties of GA. By the condition on line 35 of the joining protocol, honest parties don't become awake in the middle of any epoch. It follows that all honest parties awake at any point during the GPE of that epoch will have lock_{j+1} equal to a block extending B. By the way honest parties evaluate predicate(), the lemma follows for lock updates after GPE due to integrity. □

Lemma B.6. *If there are 2κ consecutive epochs in π_j where all honest parties awake at any point during that time period have est $= j$ and Δ_j holds, then during at least κ of those epochs all honest parties awake or recovering by the end of that epoch decide B, where B is the input of an honest party, with probability $1 - e^{-2\kappa\epsilon^2}$[10]. In addition, during the time in which all awake honest parties have est $= j$, $\forall j' > j$ there is at least one epoch out of every κ in which all of the honest parties awake by the end of that epoch output B, where B is the input of an honest party, with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$.*

Proof. We start by proving the first statement. By Lemma B.4, and line 19 of the full protocol, in each such epoch of π_j, the inputs of all honest parties to GPE satisfy *predicate* for all honest parties that are awake at any point during the GPE (this holds because honest parties don't become awake in the middle of any epoch, by the condition on line 35 of the joining protocol). As such, in expectation, a block that is the input of an honest party to GPE is decided every $\frac{1}{\frac{1}{2}+\epsilon}$ epochs in expectation by the validity property of GPE by all honest parties that are awake or recovering by

[10]Note that a recovering party may decide B while it is simulating the protocol.

the end of that epoch. With the application of a Chernoff Bound, it follows that with probability $1 - e^{-2\kappa\epsilon^2}$, this occurs at least κ epochs out of every 2κ.

We now address the second statement in the lemma. Again, by Lemma B.4, the inputs of all awake honest parties to GPE satisfy *predicate* for all honest parties awake during GPE in every epoch where all honest parties have $est = j$. The remainder of the lemma therefore follows from a simple Chernoff Bound. □

Lemma B.7. *If an honest party p_i decides block B directly in epoch e_j of π_j, Δ_j holds, and all honest parties have $est < j$ then all honest parties that are awake at the end of epoch $e_j + \kappa$ will have decided B in π_j with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$.*

Proof. Because p_i decided B in π_j at time t, it follows from Lemma B.2 that all honest parties awake at time t' after the end of epoch e_j have lock_j equal to a block extending B from time t' onward. By conditional safety (Lemma B.3), no honest party decides a block conflicting with B in π_j. Furthermore, by Lemma B.4 and the fact that all honest parties have $est < j$, in all subsequent epochs the blocks input to GPE by honest parties satisfy *predicate* for all honest parties awake during that GPE (and therefore extend B).

From the validity property of GPE, it follows that with probability $\frac{1}{2} + \epsilon$ all awake honest parties output $(B', 1)$ from GPE (and therefore decide it) where B' is the input of an honest party and therefore extending B in each such epoch. It follows with the application of a Chernoff Bound that this occurs within κ epochs with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. □

Lemma B.8. *If an honest party sets $est := j$ at time t and no honest party ever has $est + move > j$, then all honest parties awake at time $t' \geq t + 3\Delta_k$ onward have $est = j$ by time t'.*

Proof. On line 7 of the joining protocol, a joining party sets its values of *est* and *move* for the first time during the joining protocol based on one iteration of π_{beacon}. Since no honest party ever has $est + move > j$, no honest party can ever output $j' > j$ from the GA in π_{beacon} by integrity, so no honest party can ever have $est > j$. It follows that if all honest parties awake at any point $t'' > t$ input j to π_{beacon}, then all honest parties awake at any time $t' \geq t + 3\Delta_k$ onward will have $est = j$ by time t'. We show that this holds.

By graded delivery, all honest parties awake or in the recovering state output j with grade 0 or 1 and no honest parties output $j' > j$ by integrity from GA in π_{beacon} that begins at time $t - 3\Delta_k$. From lines 5- 9 of π_{beacon}, it follows that all awake honest parties input to the GA of the subsequent iteration of π_{beacon} j. It follows from validity that all honest parties that output from the iteration of π_{beacon} that ends at time $t + 3\Delta_k$ output $(j, 1)$, and all honest parties that are awake have $est = j$ and $move = 0$ at this point by lines 5- 7 of π_{beacon}. Since joining parties set their starting values of *est* and *move* based on the output of an iteration of π_{beacon} prior to becoming awake, it follows that any party that is awake and therefore inputs to the next iteration of π_{beacon} has $est = j$ at $move = 0$ at the time of doing so. This continues to hold for all subsequent iterations of π_{beacon} due to the fact that joining parties set their starting values of *est* and *move* based on the output of an iteration of π_{beacon} prior to becoming awake and the fact that honest parties never set *move* to a value less than 0. □

Lemma B.9. *No honest party ever sets $est > ideal$ except for probability $1 - \text{negl}(\kappa)$.*

Proof. From the protocol in Figure 3, an honest party must output a value from GA_B that is greater than *ideal* in order to set *est* past *ideal*. By the integrity property of GA, this implies that some honest party input a value $j > ideal$ to GA. To do so, that party must already have $est > ideal$ or $est \leq ideal$ and $move \geq 1$.

Now, we show a proof by contradiction. Let p be the first party to set $est > ideal$ (if multiple did at the same time, choose one arbitrarily without loss of generality), and let t be the time at which they did so. If p set $est > ideal$ while simulating the protocol based on saved messages during the joining protocol, let t be the time at which the GA in π_{beacon} that caused them to do so ended. This implies three possibilities prior to time t due to the integrity property of GA: (a) some awake honest party p' had $est = ideal$ and set $move := 1$ due to meeting one of the criteria for moving up or (b) some honest party had $est \leq ideal$ and $move > ideal - est$ due to outputting a value $v > ideal$ from GA_B in π_{beacon} or (c) some honest party joined the protocol with $est = ideal$ and $move = 1$.

We start by addressing the first case, and check if it is possible for some party p' to have had $est = ideal$ and set $move := 1$ due to meeting one of the criteria for moving up. Without loss of generality, let p' be the first party to have $est = ideal$ and $move := 1$. We go through each possible criteria for moving up to check if it's possible for p' to meet one of them at some time t^* prior to time t, assuming that none of the other criteria are met prior to time t^*. If none of the criteria are met in the absence of any of the other criteria already having been met, it follows that possibility (a) could not have happened.

We start with criterion 1 for moving up, and let $t' \leq t^* < t$ be the time at which p' set $est = ideal$. All honest parties awake at time $t' + 3\Delta_k$ onwards must have had $est = ideal$ by time $t' + 3\Delta_k$ by Lemma B.8. By Lemma B.6 there are κ epochs out of every 2κ epochs from time $t' + 3\Delta_k$ onward in which all honest parties that are awake or recovering by the end of that epoch decide a new block that is the input of an honest party with probability $1 - e^{-2\kappa\epsilon^2}$, and this holds for every 2κ epoch period after that (as long as no honest party increments est past $ideal$ for any other reason).

Next, we address criterion 2. Again, let $t' \leq t^* < t$ be the time at which p' set $est = ideal$. All honest parties awake at time $t'' \geq t' + 3\Delta_k$ onward must have had $est = ideal$ by time t'' by Lemma B.8 (prior to which this criterion could not have been met for p'). By Lemma B.7, when an honest party directly decides a block in π_j for $j \geq ideal$ and all awake honest parties have $est = ideal$, all honest parties awake by the end of the subsequent κ epochs will have decided that block with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. By Lemma B.5, once all honest parties awake at a certain time τ decide a block B_τ in $\pi_{j \geq ideal}$, all honest parties awake at any time $\tau' \geq \tau$ are locked on a block extending B_τ in π_{j+1} for all epochs of π_{j+1} beginning at or after time τ. By Lemma B.4 and Lemma B.6, all honest parties awake at the end of the κ subsequent epochs of π_{j+1} will have decided that block with probability $1 - 2^{-\kappa}e^{-2\kappa\epsilon}$. If a party decides a block indirectly during the joining protocol in protocol π_j, this implies that some honest party must have previously decided this block directly by the integrity property of GA. It follows from the fact that $\Delta_j = 2\Delta_{j-1}\ \forall 1 < j \leq k$ that any block decided in protocol π_j for $ideal \leq j < k$ all honest parties awake at the end of the subsequent 2κ epochs of $\pi_{j'+1}$ will have decided that block with high probability, in which case the criterion cannot be met in the absence of any of the other criterion having been met.

Finally, we address criterion 3. We start by checking whether the first part of the statement is possible: $\text{lock}_{j'}$ conflicts with the chain constructed in π_j for some $j \geq ideal$ and $j' > j$. Conditional safety (Lemma B.3) applies for π_j since no honest party has $est > ideal$, and Lemma B.1 ensures that $\text{lock}_{j'}$ was decided by an honest party in π_j. It follows that this part of the criterion cannot be met. The second part of the statement, that there is a fork in the chain constructed in π_j for $j \geq ideal$, also cannot be met due to conditional safety.

It therefore follows that option (a) could not have happened.

Next, we address (b): the possibility that p' had $est \leq ideal$ and $move > ideal - est$ due to outputting a value $v > ideal$ from GA in π_{beacon} prior to time t. For this to happen, p' must have output $v \geq ideal + 1$ from an instance of GA in π_{beacon} prior to time t. By the integrity property of GA, some honest party p'' must have input $v' \geq v$ to GA prior to time t. Without loss of generality,

assume that p'' is the first party to have done so prior to time t. As already addressed when exploring possibility (a), it is not possible for p'' to have done so due to having $est = ideal$ and $move = 1$ from meeting the criteria for moving up except for negligible probability. Assume for now that (c), p'' joining the protocol with $est = ideal$ and $move = 1$, also cannot happen (we will later show that this is the case). By the definition of p'', possibility (b) also cannot be because it would imply that a party input $v'' \geq v$ to GA prior to p'' doing so.

Finally, we address (c): the possibility that p' joined the protocol with $est = ideal$ and $move = 1$.

Assume first that p' recovered with $est = ideal$. By Lemma 5.5, it must be the case that $t^*_{1,ideal} \leq t^*_{ideal} \leq t^*_{1,ideal} + 3\Delta_k$. For p_{join} to set $move := 1$, we address three possibilities: (i) it must set $move := 1$ on line 30 due to having decided too few blocks to some chain Λ_j for $j \geq ideal$ on line 20 (ii) it meets criteria 1, 2, or 3 for moving up or (iii) there are conflicting outputs (of any grade) in $\text{out}_{high,j}$ for $j \geq est$ (line 16).

We start by addressing (i). By time ts, $\lfloor \frac{ts-(t^*_{1,ideal}+3\Delta_k)}{20\kappa\Delta_{ideal}} \rfloor$ 2κ epoch periods must have elapsed in which there were κ epochs such that a new block was decided by all honest parties awake at the end of that epoch in π_{ideal}. By the integrity property of GA, the way that honest parties set backlog_{ideal} on line 4 of π_{beacon}, and Lemmas B.3 and B.6, it follows that at least $\text{backlog}_{ideal} + \max(\lfloor \frac{ts-(t^*_{1,ideal}+3\Delta_k)}{20\kappa\Delta_{ideal}} \rfloor \cdot \kappa, 0)$ blocks must have been decided in π_{ideal} by all honest parties awake at time ts. By Lemma 5.1, it follows that p_{join} decides all blocks that were decided by all honest parties awake at time ts in π_{ideal}. For all $j > ideal$, the only way for p_{join} to set $move := 1$ on line 30 is if more than $3\Delta_k + \sum_{i=est}^{j} 20\kappa\Delta_i$ time has passed since t^*_{est} and fewer than $\text{backlog}_{est} + min\_blocks_j$ blocks were decided. By Lemma 4.8, $\text{backlog}_{est} + min\_blocks_j$ blocks must have been decided by time ts by all awake honest parties that were awake at time ts in π_j and their logs must not conflict by Lemma B.3. By Lemma 5.1, p_{join} must have decided those blocks in π_j as well on line 20. It follows that p_{join} does not set $move := 1$ for this reason.

Next, we address (ii), and we start with criterion 1. For criterion 1 not to be triggered, p_{join} must see κ blocks decided every 2κ epochs in π_{ideal} from time $t_c = t^*_{est} + 3\Delta_k + 20\kappa\Delta_{est}\lfloor \frac{ts-(t^*_{est}+3\Delta_k)}{20\kappa\Delta_{ideal}} \rfloor$ onward. If p_{join} decided an excess of $\text{backlog}_{ideal} + min\_blocks_{ideal}$ blocks on line 20, then these count toward the blocks it sees in the first 2κ epoch period after t_c (as per line 37 and the way that it sets $ts_{B,ideal}$ for each of these blocks on line 32). If it didn't see an excess of min_blocks_{ideal} blocks decided, by Lemma 5.1 this implies that none of the κ epochs in the 2κ period starting at time t_c in which a new block would have been decided by all honest parties has happened prior to time ts. By Lemma B.6, except for negligible probability those κ epochs will occur in each 2κ epoch period after t_c, and p_{join} *will* decide those blocks since it starts receiving messages from the time it enters the *recovering* state, which is prior to time ts (and it will store the timestamps of those blocks based on the simulation). It follows from Lemma B.6 that p_{join} will not meet criterion 1 for moving up for any of the 2κ epoch periods after t_c prior to setting its status to awake except for negligible probability.

For criterion 2, p_{join} starts checking this criteria for the blocks in $\text{diff}(\Lambda_j, \Lambda_{j+1})$ for $est \leq j < k$. By Lemma 5.1, it follows that because p_{join} has not decided such a block B in π_{j+1} yet, this is because the point at which it has been decided by *all* awake honest parties hasn't happened yet. For such a block, $ts_{B,j}$ (as set on line 34) must be lower bounded by the time at which the first honest party to decide B in π_j does so. This is because an honest party must have decided B and input a block extending B to $\text{GA}_{high,j}$ for it to have been output by p_{join} by the integrity property of GA. It therefore follows from Lemma B.6 that for any block B in $\text{diff}(\Lambda_j, \Lambda_{j+1})$ such that $ts_{B,j} > t^*_j$ for $j \geq est$, p_{join} *will* decide B in π_{j+1} within the subsequent 2κ epochs after $ts_{B,j}$, because the point at

which it is decided by all awake parties is after ts, at which point p_{join} starts receiving all messages. Lemmas B.2 and B.3, and the integrity property of GA, ensure that criterion 3 will not be triggered.

Similarly, Lemmas 5.1 and B.3, and the integrity property of GA ensure that (iii) will not happen. p_{join} cannot $move := 1$ with $est = ideal$ without recovering with $est = ideal$ prior to joining due to the same argument as case (a).

We have thus arrived at a contradiction. □

Definition B.10 ($t^*_{1,j}$). $t^*_{1,j}$ is defined as the earliest time at which an honest party outputs $(j, 1)$ from π_{beacon}.[11]

Definition B.11 ($checkpoint_{i,t}$). $checkpoint_{i,t}$ for party p_i at time t is defined as decide_j for p_i s.t. $j \geq ideal$ and $\forall j' \geq ideal$, $height(\text{decide}_j) \geq height(\text{decide}_{j'})$.

Definition B.12 ($min\_backlog\_height_j$). Let H_t be the set of honest parties awake at time $t = t^*_{1,j} - 3\Delta_k$. $min\_backlog\_height_j$ is defined as, for the set A of honest parties awake at any time from $t^*_{1,j} - 3\Delta_k$ onward that do not recover with $est = j$, $checkpoint_{i,t}$ s.t. $checkpoint_{i,t} \leq checkpoint_{j,t'} \forall p_j \in A$ and $t' \geq t$.

Recall from Section 5 that we say that a party *recovers with* $est = j$ if it sets $est = j$ if it sets $est = j$ during its first full iteration of π_{beacon} in the recovery protocol (i.e. the one beginning at time ts).

Lemma B.13. *For any honest party that recovers with $est = j$, $backlog_j \geq min\_backlog\_height_j$.*

Proof. This follows from the integrity property and validity properties of GA, and the fact that if an honest party recovers with $est = j$, they must have set backlog_j based on the output of $\text{GA}_{backlog,j}$ beginning at time ts s.t. $ts \geq t^*_{1,j} - 3\Delta_k$ (otherwise they couldn't have recovered with $est = j$), and the way that honest parties choose their inputs to $\text{GA}_{backlog,j}$. □

Lemma B.14. *Blocks are eventually decided at a rate of 1 block per $O(\Delta_{ideal})$ time in expectation in all $\pi_{j \geq ideal}$.*

Proof. To prove the lemma statement, we show that in all π_j for $ideal \leq j \leq k$, $2^{j-ideal}\kappa$ blocks are decided every 2κ epochs. Because $\Delta_j = \frac{\Delta_{j+1}}{2}$ $\forall j < k$, this implies the lemma statement.

First, we prove the claim that new blocks are decided at a rate of κ blocks every 2κ epochs in π_{ideal}. By Lemma B.9, we know that all honest parties have $est \leq ideal$ for the duration of the protocol. If there eventually comes a time that all awake honest parties have $est = ideal$, the claim follows from Lemma B.6. If this does not eventually happen, validity of GA implies that for every iteration of π_{beacon}, some honest party inputs $est + move < ideal$. Eventually, by validity of GPE, there must be some $j < ideal$ such that some honest party (note that it need not be the same one each time) inputs j to each iteration of π_{beacon} for the remainder of the protocol. Otherwise, at some point, all the awake honest parties *would* reach $est = ideal$.

As per Definition B.10, let $t^*_{1,j}$ be the earliest time at which an iteration of π_{beacon} output $(j, 1)$ for an honest party. By graded delivery and validity of GA and the fact that honest parties listen to (more than) one iteration of π_{beacon} prior to becoming awake, all honest parties awake at time $t^*_{2,j} = t^*_{1,j} + 3\Delta_k$ must have output $(j, 1)$ by that time. Note that no honest party could output $(j', 1)$ for $j' > j$ from π_{beacon}, as this would contradict the definition of j.

[11] Note that the honest party may do this while simulating the protocol based on received messages in the joining protocol, in which case $t^*_{1,j}$ is they time they would have output $(j, 1)$ had they been awake at the time of receiving those messages, rather than the timestamp during simulation.

Let t be some time during the course of the protocol s.t. $t > t^*_{2,j}$. Let p be the party that input j to the latest instance of π_{beacon} that completed at or prior to time t with $move = 0$ (note that some honest party must have done this, otherwise validity of GA would imply that all of the awake honest parties output $(j+1, 1)$ from the subsequent GA). If p did not recover with $est = j$ (i.e. it was awake at the time of doing so or did so while simulating the protocol), this implies that it didn't meet any of the criteria for moving up. That is, it decided κ blocks for every 2κ epochs from $t^*_j \leq t^*_{2,j}$ onward and every block it decided in $\pi_{j'}$, it decided in $\pi_{j'+1}$ for $j < j' < k$ within the 2κ subsequent epochs. In this case, the lemma follows from Lemma B.7 (which shows that all honest parties decide the blocks decided by p in $\pi_{j'' \geq ideal}$ with a small delay but at the same latency) and the fact that $\Delta_{j+1} = 2\Delta_j \forall j < k$.

Otherwise, if p recovered with $est = j$, this implies that it did not set $move := 1$ on line 30 of the joining protocol. That is, the length of its log Λ_j is at least $min\_blocks_j + min\_backlog\_height_j$, which is equivalent to having decided $min\_backlog\_height_j$ blocks prior to t^*_j and having decided blocks at a rate of κ blocks for every 2κ epoch of π_j from time t^*_j onward. Note that by Lemma 5.5, $t^*_j \leq t^*_{1,j} + 3\Delta_k$. Furthermore, for all $\pi_{j'>j}$ the length of $\Lambda_{j'}$ is at least $\text{backlog}_j + min\_blocks_{j'-1} - 2^{j'-j}\kappa$, implying that $\Lambda_{j'}$ is growing at the rate of κ blocks per every 2κ epochs of π_j from time t^*_j with a small, constant, bounded delay. Furthermore, p has never met any of the criteria for moving up by the time that that it inputs to π_{beacon}, implying that it continued to decide the equivalent of κ blocks per 2κ epochs if π_j from time t^*_j onward up to time t, and that those blocks were confirmed in higher sub-protocols. Because every honest party that joins the protocol has $t^*_j \leq t^*_{1,j} + 3\Delta_k$, if an honest party recovers with $est = j$ and $move := 0$, the above argument, Lemmas B.7 and B.13, and the fact that $\Delta_{j+1} = 2\Delta_j \forall j < k$ imply that the lemma holds.

□

Lemma B.15 (Liveness). *Blocks containing transactions continue to be appended to $\Lambda_j \forall j \geq ideal$.*

Proof. The lemma follows from Lemma B.14. □

Lemma B.16 (Safety). *If two honest parties deliver logs $\Lambda_{j'}$ and $\Lambda'_{j'}$, then $\Lambda_{j'}$ and $\Lambda'_{j'}$ do not conflict $\forall \Delta_{j'} \geq \Delta_{ideal}$.*

Proof. The lemma follows from conditional safety (Lemma B.3) and the fact that no honest party ever has $est > ideal$ (Lemma B.9). □