Low-Latency Dynamically Available Total Order Broadcast

Sravya Yandamuri[1], Nibesh Shrestha[3], Luca Zanolini[4], and Kartik Nayak[5]

[1] Common Prefix, EE and University of Luxembourg, LU
syandamuri31@gmail.com
[2] Supra Research, USA
nibeshrestha2@gmail.com
[3] Ethereum Foundation, UK
luca.zanolini@ethereum.org
[4] Duke University, USA
kartik@cs.duke.edu

Abstract. 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 [17] introduced the first protocol operating in the *dynamically available* model [13], 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 [18]. 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 [11,2,9,19]. [5] Initial works using the longest-chain approach incurred latency that depended on the security parameter and the actual level of participation [1,3,18]. Several works, including Goyal et al. [11], Prism [2], parallel chains [9], and OHIE [19], removed the dependency on one of these parameters but not the other. Finally, MR [16] and MMR [15,14] designed protocols with $O(\Delta)$ latency where Δ is a pessimistic network delay. However, all of these protocols only work under a synchronous [7] 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 [8] 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 [13]. Thus, we have the following natural question:

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 on the network delay enjoys consistency with all other clients that 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 $\Delta_{ideal} < 2\delta$.

To model dynamic availability, we work within the extended sleepy model of Malkhi et al. [15], referred to as the (T_f, T_b, α)-sleepy model. In this model, parties can be in one of three states: awake, asleep, or recovering. A party that is awake participates in the protocol execution. A party that is asleep does not send or receive any messages, nor does it participate in the protocol. When a party transitions from asleep to awake, it first enters the recovering state and receives messages from the other parties to learn the state that it should adopt prior to setting its status to awake. The T_f and T_b signify that a malicious party is counted as corrupt an extra T_f time forward and an extra T_b time backward from the time that it is active: in the (T_f, T_b, α)-sleepy model, 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 [15] 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,

[5] 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.

Theorem 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 [14]), 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 $O(\Delta_{ideal})$. [6]

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

[6] Note that if network conditions improve and Δ_{ideal} decreases, our protocol does not ensure that progress is made at the speed of the new value of Δ_{ideal}. We leave this for future work.

overcome these challenges and present a joining protocol that ensures that new parties joining the protocol do not prevent liveness.

A note on finality. In case a party proceeds with a lower-latency instance based on its local view but later observes conflicting blocks in a higher-latency instance (indicating the lower delay bound does not hold and is less than Δ_{ideal}), any blocks it committed in the lower latency instance that were not also committed in the higher-latency instance may be reverted because its possible that they are not committed to the log of the higher latency instance. Since a user does not know if it will later learn that it's given estimate of Δ_{ideal} is wrong, they may want to see a block committed in π_k to be sure that it has been finalized. This this could result in delayed finality if $\Delta_k > \Delta_{ideal}$. Note that the amortized block production rate would still be $O(1)$ blocks per $O(\Delta_{ideal})$ time.

This work is structured as follows. In Section 2, we present the necessary preliminaries. In Section 3, we present the dynamically available TOB protocol with an assumption on the state adopted by parties joining the network [7]. We remove this assumption in the full version of this work, where we show the joining protocol.

2 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.[8]

[7] See the full version of the paper for a detailed technical overview of our solution.

[8] 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

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*[9]. 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 [15], which differs from the original sleepy model of [18]. 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 [15] 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 parties awake at time t. Then:

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

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.

Definition 1 (Total-Order Broadcast [5]). *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.

3 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

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.

[9] 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.

pseudo-code for the protocol and provide a detailed explanation of its execution, highlighting how the protocol operates step by step under various conditions.

3.1 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 [18]. This has led to the creation of a series of dynamically available Total-Order Broadcast protocols [15,6,4,16,10,5], which are capable of tolerating fluctuations in the participation levels of parties. These protocols must be synchronous [12][13, 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 [13].

The protocol. In Figures 1 and 2 we present our protocol for dynamically available TOB. [10] Figure 1 contains the protocol that is run by each of the parties for each of the k delays, while Figure 2 contains the protocol that is used to determine when a party should increase its estimate of Δ_{ideal}. Please see the full version of this work for a detailed explanation of the pseudocode.

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. 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

[10] The protocol uses Graded Agreement and Graded Proposal Election; the definitions of these primitives may be found in the full version of this work.

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 $\mathsf{GA}_{1,j}$
    $\mathsf{out}_{1,j} := \mathsf{GA}_{1,j}(B_{GA0})$
    $\mathsf{lock}_j :=$ block with the greatest height output with grade 1 from $\mathsf{out}_{1,j}$
    $B_{\mathsf{cand}} :=$ 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
      $(B_{\mathsf{proposed}}, g) := \mathsf{GPE}_j(B_{\mathsf{proposal}}, \mathsf{predicate}(B) = \mathsf{true}$ if B extends $\mathsf{lock}_j)$
    else
      $B_{\mathsf{proposal}} := B_{\mathsf{cand}}$
      $(B_{\mathsf{proposed}}, g) := \mathsf{GPE}_j(B_{\mathsf{proposal}}, \mathsf{predicate}(B) = \mathsf{true}$ if B extends 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. 1. Protocol π_j for party P_i.

1: **at** time $t = 3\Delta_k, 6\Delta_k, 9\Delta_k, \ldots$ **do**
2: $\mathsf{out} := \mathsf{GA}_{beacon}(est + move)$
3: let $v = val$ s.t. $(val, *) \in \mathsf{out}$ and $\forall (val', grade') \in \mathsf{out}$, $val \geq val'$
4: let $v_1 = val$ s.t. $(val, 1) \in \mathsf{out}$ and $\forall (val', 1) \in \mathsf{out}$, $val \geq val'$
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. 2. Protocol π_{beacon} for party P_i run with delay Δ_k.

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 2 (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*. Honest parties propose new blocks in the GPE of π_{est} and evaluate $\mathsf{predicate}(B) = \mathsf{true}$ 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 1, 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 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. 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 $\mathsf{predicate}$ 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 2. *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.

Lemma 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. 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 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 $\mathsf{predicate} = true$ 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. *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 5. *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 $\mathsf{GA}_{0,j}$ 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 that B_{cand} of all honest parties at time $(10e_j + 6)\Delta_j$ was output from $\mathsf{GA}_{0,j}$ 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 $\mathsf{GA}_{1,j}$. 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 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 6. *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 3), no honest party decides a block conflicting with B in π_j. By conditional safety and Lemma 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 awake honest parties input to $\mathsf{GA}_{0,j+1}$ 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 $\mathsf{GA}_{0,j+1}$ 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 $\mathsf{GA}_{1,j+1}$. 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 $\mathsf{predicate}()$, the lemma follows for lock updates after GPE due to integrity.

The following lemma will later be used 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 7. *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 5, 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 5, 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 8. *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 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 3), no honest party decides a block conflicting with B in π_j. Furthermore, by Lemma 5 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 9. *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 10. *No honest party ever sets $est > ideal$ except for probability $1 - \mathsf{negl}(\kappa)$.*

Proof. From the protocol in Figure 2, an honest party must output a value from $\mathsf{GA_B}$ 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 9. By Lemma 7 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 9 (prior to which this criterion could not have been met for p'). By Lemma 8, 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 6, 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 5 and Lemma 7, 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: $\mathsf{lock}_{j'}$ conflicts with the chain constructed in π_j for some $j \geq ideal$ and $j' > j$. Conditional safety (Lemma 3) applies for π_j since no honest party has $est > ideal$, and Lemma 1 ensures that $\mathsf{lock}_{j'}$ 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 11 (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 7 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 12 (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 3) and the fact that no honest party ever has $est > ideal$ (Lemma 10).

4 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. 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.
4. 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.
5. 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.
6. 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.
7. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
8. Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.
9. 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.
10. 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.
11. 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.
12. 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.
13. Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *CoRR*, abs/2304.14701, 2023.

14. Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine consensus under fully fluctuating participation. *IACR Cryptol. ePrint Arch.*, page 1448, 2022.
15. 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.
16. 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.
17. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.
18. 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.
19. 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.