Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar
Technion, Haifa, Israel

Oded Naor
Technion, Haifa, Israel
StarkWare Industries Ltd., Netanya, Israel

Ouri Poupko
Ben-Gurion University, Beer Sheva, Israel

Ehud Shapiro
Weizmann Institute of Science, Rehovot, Israel

Abstract

Cordial Miners are a family of efficient Byzantine Atomic Broadcast protocols, with instances for asynchrony and eventual synchrony. They improve the latency of state-of-the-art DAG-based protocols by almost 2× and achieve optimal good-case complexity of $O(n)$ by forgoing Reliable Broadcast as a building block. Rather, Cordial Miners use the *blocklace* – a partially-ordered counterpart of the totally-ordered blockchain data structure – to implement the three algorithmic components of consensus: Dissemination, equivocation-exclusion, and ordering.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases Byzantine Fault Tolerance, State Machine Replication, DAG, Consensus, Blockchain, Blocklace, Cordial Dissemination

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.26

Related Version *Full Version*: `https://arxiv.org/abs/2205.09174`

Funding *Oded Naor*: Oded Naor is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship, and to the Technion Hiroshi Fujiwara Cyber-Security Research Center for providing a research grant.

Acknowledgements Ehud Shapiro is the Incumbent of The Harry Weinrebe Professorial Chair of Computer Science and Biology at the Weizmann Institute.

1 Introduction

The problem of ordering transactions in a permissioned Byzantine distributed system, also known as *Byzantine Atomic Broadcast (BAB)*, has been investigated for four decades [30], and in the last decade, has attracted renewed attention due to the emergence of cryptocurrencies.

Recently, a line of works [4, 14, 20, 33, 21, 27] suggests ordering transactions using a distributed Directed Acyclic Graph (DAG) structure, in which each vertex contains a block of transactions as well as references to previously sent vertices. The DAG is distributively constructed from messages of *miners* running the consensus protocol. While building the DAG structure, each miner also totally orders the vertices in its DAG locally. That is, as the DAG is being constructed, a consensus on its ordering emerges without additional communication among the miners.

Table 1 Performance summary. Bullshark is for the ES model, and DAG-Rider is for the asynchronous model. Both protocols employ RB, which requires at least two rounds of communication of simple messages for optimal latency [2] and $O(n^2)$ amortized message complexity, or four rounds with erasure coding when using Das et al. [15] and $O(n)$ amortized message complexity.

Protocol	Reliable Broadcast Used	Latency				Amortized Message Complexity
		Eventual Synch.		Async.		
		Good	Expected	Good	Expected	
Cordial Miners (this work)	None	3	4.5	5	7.5	good-case: $O(n)$ worst-case: $O(n^2)$
Bullshark (for ES)	Optimal latency [2]	4	9	8	12	good- & worst-case: $O(n^2)$
DAG-Rider (for asynch.)	Das et al. [15]	8	18	16	24	good- & worst-case: $O(n)$

The two state-of-the-art protocols in this context are DAG-Rider [21] and Bullshark [33]. DAG-Rider works in the asynchronous setting, in which the adversary controls the finite delay on message delivery between miners, and Bullshark works in the Eventual Synchrony (ES) model, in which eventually all messages between correct miners are delivered within a known time-bound.

Both protocols use *Reliable Broadcast (RB)* [7] as a building block to disseminate vertices in the DAG. RB ensures that Byzantine miners cannot equivocate, i.e., they cannot successfully send two conflicting vertices to the correct miners. By using RB to exclude equivocation, the DAGs of all correct miners eventually contain the same vertices.

But using RB has costs in terms of message complexity and latency. The well-known Bracha RB [7] protocol entails $O(n^2)$ message complexity for each broadcast message, where n is the number of miners, and has a latency of 3 rounds of communication. The lower bound for RB is 2 rounds [2], and the message complexity lower bound is $O(n^2)$ [19]. Recent RB protocols [15, 16] improve the message complexity to $O(n)$ in some cases by using erasure codes [5], but require between 4 to 5 rounds of communication.

DAG-Rider and Bullshark need to invoke a sequence of RB instances several times to reach a single instance of consensus. E.g., DAG-Rider requires 6 sequential instances of RB in the expected case, making its latency between 12 to 24 rounds of communication, depending on the RB protocol it uses. Bullshark requires between 9 to 18 rounds in the expected case in the ES model.

It is within this context that we introduce *Cordial Miners* – a family of simple, efficient, self-contained Byzantine Atomic Broadcast [9] protocols that forgo RB, and present two of its instances for the models ES and asynchrony.

The ES Cordial Miners protocol reduces the expected latency from 9 rounds in today's state-of-the-art to 4.5, and the good case latency from 4 to 3. The asynchronous version of Cordial Miners improves the expected latency from 12 rounds to 7.5, and the good case latency from 8 to 5. This is while maintaining the same amortized quadratic message complexity in the worst case. Cordial Miners also demonstrates better performance with $O(n)$ complexity in the good case when the actual number of Byzantine miners is $O(1)$ and the network is synchronous. Protocols that use RB do not differ in their performance between the good and worst cases. Tab. 1 summarizes Cordial Miners' performance compared to DAG-Rider (for asynchrony) and Bullshark (for ES).

The crux of the Cordial Miners protocols is that instead of using RB to eliminate equivocation (and absorbing its rather high latency), miners cooperatively create a data structure that accommodates equivocations, termed *blocklace*, which is a partially-ordered counterpart of the blockchain data structure [29]. When a miner wishes to disseminate a block, it simply sends it to all other miners, taking a single round of communication, instead of at least two when using reliable broadcast.

Although the blocklace may contain equivocating blocks created by Byzantine miners, they are excluded by the ordering protocol, which is locally computed by each miner without inducing any extra communication or latency. This is realized by the function τ that converts the partially-ordered blocklace to a totally-ordered sequence of blocks while excluding equivocations along the way. Thus, by "complicating" the local ordering task to exclude equivocations, we forgo the extra communication rounds and latency associated with RB.

Roadmap. The rest of the paper is structured as follows: §2 describes the models and defines the problem; §3 provides intuition and overview of the different components; §4 introduces the blocklace data structure; §5 explains the τ function that locally turns the blocklace into a totally-ordered sequence of blocks; §6 describes the entire Cordial Miners protocols for the two network models; §7 presents the performance analysis; §8 is related work; and lastly, §9 concludes the paper. To accommodate the space limitations some details are deferred to the appendices. App. A describes a formal mathematical model for cordial miners, and some explanatory figures are deferred to App. B. Two additional appendices: one that details the full proofs and another that describes further future directions and optimizations appear in the full version of this paper [22].

2 Model and Problem Definition

We assume a set Π of $n \geq 3$ *miners* (aka agents, processes), of which at most $f < n/3$ may be *faulty* (act under the control of the adversary, be "Byzantine"), and the rest are *correct* (also honest or non-faulty). Each miner is equipped with a single and unique cryptographic key-pair, with the public key known to others. Miners can create, sign, and send messages to each other, where any message sent from one correct miner to another is eventually received. In addition, each miner can sequentially *output* (aka "deliver") messages (e.g., to a local output device or storage device). Thus, each miner outputs a *sequence* of messages.

Let Λ denote the empty sequence; for a set X, X^* is the set of all sequences over X; for sequences x and y, $x \preceq y$ denotes that x is a prefix of y; $x \cdot y$ denotes the concatenation of x and y; and x, y are *consistent* if $x \preceq y$ or $y \preceq x$.

The problem we aim to solve in this paper is to devise an ordering consensus protocol that is safe and live:

▶ **Definition 1** (Safety and Liveness of an Ordering Consensus Protocol). *An ordering consensus protocol is:*

Safe *if output sequences of correct miners are consistent.*

Live *if every message sent by a correct miner is eventually output by every correct miner with probability 1.*

Here, we aim to devise safe and live ordering consensus protocols for models of distributed computing with two types of adaptive adversaries that can corrupt up to f miners throughout the run: First, *Asynchrony*, in which the adversary controls the finite delay of every message. Second, *Eventual Synchrony (ES)*, in which there is a point in time, known as the *Global Stabilization Time (GST)*. After GST, the adversary controls the delivery time of messages sent between correct miners, but they must be delivered within a known bound Δ. We further assume the adversary is computationally bounded and, therefore, cannot break cryptographic signatures.

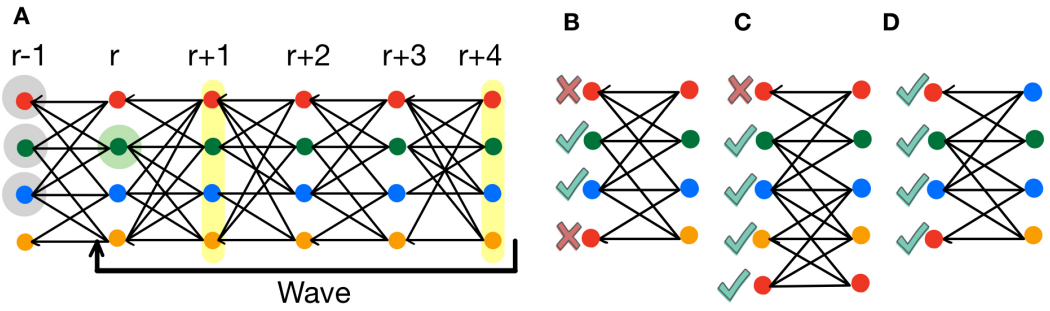

Figure 1 The blocklace data structure, equivocations, approval, and ratification. Four miners (red, green, blue, yellow). Each circle represents a block and each line a hash pointer to the left block. (A) A single wave consisting of five consecutive rounds. The green block in round r with the halo is the leader block. Each of the highlighted blocks in yellow in rounds $r+1$ and $r+4$ have a path to the leader block, making it a final leader block. The blocks with a gray halo are ordered by τ when the leader block becomes final. (B) The red equivocates, with the top red block approved by the green block of the next round, the bottom red block approved by the yellow block of the next round, and the blue block of the next round, observing both equivocating red blocks, approves neither, and hence neither of the red blocks has the three approvals (including the red block itself) needed for ratification. (C) Here the blue block of the next round observes only the bottom red block and hence approves it, which together with the yellow block and the red block itself form a supermajority, and hence the bottom red block is ratified, but not the top one. (D) Here the blue miner equivocates in the next round, with the top blue block of the next round approving the top red block, which together with the green and red form a supermajority that ratifies it. Similarly, the bottom blue block, the yellow block, and the red (which is not illustrated) ratify the bottom red block. Indeed, with two equivocators (red and blue) out of four, an equivocation can be ratified.

We note that safety and liveness, combined with message uniqueness (e.g., a block in a blocklace, see next), imply the standard Byzantine Atomic Broadcast guarantees: Agreement, Integrity, Validity, and Total Order [9, 21]. Hence, protocols that address the problem defined here are in fact protocols for Byzantine Atomic Broadcast.

Next, we provide an overview of the Cordial Miners protocol, including the blocklace, the dissemination of blocks, and the local ordering of the blocks to a final sequence.

3 Cordial Miners Overview

In the Cordial Miners protocols, the miners jointly built the *blocklace* data structure, a partially-ordered counterpart of the totally-ordered blockchain. A blocklace created by four miners, each of a different color, is is illustrated in Fig. 1. The *depth* of a block in a blocklace is the length of the maximal path emanating from it, and a *round* of a blocklace consists of blocks of the same depth. A set of blocks by more than $\frac{1}{2}(n+f)$ miners is termed a *supermajority*; note that if $f = 0$ then a supermajority is a simple majority. Correct miners are *cordial* in that they wait for round r to attain a supermajority before contributing a block to round $r+1$.

Fig. 1.A presents a blocklace constructed by four miners s.t. each column is a single round representing blocks from different miners, and each row in the same color consists of blocks from the same miner. Thus, each correct miner creates a single block in each round. Note that different miners can have different partial views of the blocklace, and the goal is to "converge" the order of the blocks to a consistent order for all the miners.

Each block holds a set of transactions as well as hash pointers (the edges in the DAG) to blocks of previous rounds. When a miner observes that round r has attained a supermajority (is *cordial*) it creates a new block b of round $r+1$ with pointers to the *tips* of its blocklace

up to round r, which are the blocks in the blocklace with no incoming edges from blocks of depth up to r. The tips must include a supermajority of blocks of round r, but possibly also blocks of earlier rounds not already observed by the blocks received in round r. (One block observes another if there is a path of pointers from one to the other.) E.g., if the figure represents the local blocklace of the red miner, then since round $r + 4$ is cordial, the red miner can create a new block b in round $r + 5$ with pointers to all the blocks in round $r + 4$. The miner then sends b to all other miners. The blocklace data structure is defined in §4.

Next, we explain how Cordial Miners use the blocklace for the three algorithmic components of consensus: Dissemination, equivocation-exclusion, and ordering.

Dissemination. In the good case, dissemination is realized simply by each miner sending each new block to all other miners. However, faulty miners may fail to do so, possibly intentionally, and send new blocks only to some of the miners.

The principle of cordial dissemination [29] is: *Send to others blocks you know and think they need.* Its blocklace-based Byzantine-resilient implementation uses each block in the blocklace as an ack/nak message: A new block created by a correct miner p points, directly or indirectly, to the blocks in p's local blocklace. It thus discloses the blocks known by p at the time of its creation and, by omission, also of the blocks not yet known to p. This way, a miner q that receives p's block can send back to p any block known to q and not known to p according to the disclosure made by p's block. E.g., the green block in round $r + 4$ serves as an ack message for all the blocks that it observes, including the red, green, and blue blocks in round $r + 3$. It also serves as a nak message for the yellow block of round $r + 3$. As an example of cordial dissemination, when the red miner sends the block it creates to the green miner in round $r + 5$, it will also send to the green miner the yellow block in round $r + 3$. The dissemination protocol is formally defined in §6.

Equivocation exclusion. Two blocks b_1, b_2 of the same miner are *equivocating* if neither observes the other, i.e., there is no path of pointers from b_1 to b_2 or from b_2 to b_1. Since Cordial Miners do not use RB to disseminate blocks, the blocklace created by Cordial Miners may include equivocations created by Byzantine miners, which are later excluded when each miner locally orders the blocks in its blocklace to a sequence of final blocks. The Cordial Miners protocol uses supermajority approval to exclude equivocations s.t. for each set of equivocating blocks, at most, one is included in the final output. In addition, after detecting an equivocation, correct miners ignore the Byzantine miner by not including direct pointers to their blocks. Thus, a Byzantine miner that equivocates is eventually detected, which results in it eventually being ignored by all correct miners. Equivocation exclusion is part of the τ ordering function which is detailed in §5.

Ordering. Ordering the partially-ordered blocklace can be achieved by topological sort of the DAG. The challenge is to ensure that all correct miners exclude equivocations and order the blocks identically so that they all produce the same total order. To this end, the blocklace is divided into *waves*, each consisting of several rounds, the number of which is different for ES and asynchrony (3 and 5 rounds per wave, respectively). E.g., Fig. 1.A depicts the asynchronous version which has 5 rounds in each wave.

For each wave, one of the miners is elected as the *leader*, and if the first round of the wave has a block produced by the leader, then it is the *leader block*. The figure depicts the green block in round r in the green halo as the leader block of that wave. When a wave ends, i.e., when the last round of the wave is cordial, the leader block becomes *final* if it has

sufficient blocks that *approve* it, namely, it is not equivocating and there is a supermajority where each block observes a supermajority that observes the leader block. The figure shows two supermajorities, highlighted in yellow, where each block in the supermajority of round $r+4$ observes the supermajority of round $r+1$. The supermajority in round $r+1$ observes the leader block, and the leader block is not equivocating, making it final.

A final leader block b serves as the "anchor" of the ordering function τ, which topologically sorts (while excluding equivocations) all the blocks observed by b that have not been ordered yet. Thus, each time a wave ends with a final leader block, a portion of its preceding blocklace is ordered. In the figure, the blocks in round $r-1$ with a grey halo are ordered when the leader block in round r is final since it observes them. In case a wave ends with no final leader block, unordered blocks will be ordered when some subsequent wave ends with a final leader block. The full details of τ are in §5.

4 The Blocklace

A blocklace [28] is a partially-ordered counterpart of the totally-ordered blockchain data structure: In a blocklace, each block may contain a finite set of cryptographic hash pointers to previous blocks, in contrast to one pointer (or zero for the initial/genesis block) in a blockchain. Thus, a blocklace induces a DAG in which vertices represent its blocks and edges represent the pointers among its blocks. Next, we present the basic definitions of a blocklace, which appear as pseudocode in Alg. 1. A formal mathematical description of these definitions appears in App. A.

4.1 Blocklace Basics

In addition to the set of miners Π, we assume a given set of block **payloads** $\mathcal{A}$, typically sets of transactions, and a cryptographic hash function *hash*. A **block** consists of a payload $a \in \mathcal{A}$ and a set of hash pointers to previously created blocks, signed by its creator p, in which case it is also referred to as a ***p*-block** (Def. 14). A block **acknowledges** another block if it contains a hash pointer to it, and is **initial** if the set of hash pointers is empty. A *blocklace* is a set of blocks (Def. 15). Note that *hash* being cryptographic implies that a blocklace that includes a cycle cannot be effectively computed, and thus a blocklace B induces a DAG, with blocks as vertices in B and an edge among two vertices if the first includes a hash pointer to the second.

We say that a block b **observes** another block b', denoted $b \succeq b'$, if there is a path from block b to b'. If b is a p-block in a blocklace B, we say that miner p **observes** b' **in** B (Def. 16). We note that "observe" is the transitive closure of "acknowledge". Each miner maintains a local blocklace of blocks it created and received. With a correct miner p, any newly created p-block observes all the blocks in p's local blocklace.

The main violation a Byzantine miner q can perform is an **equivocation**, by creating a pair of q-blocks that do not observe each other (See Fig. 1.B). Such a miner q is an **equivocator** (Def. 17). If the payloads of the two blocks are financial transactions, the equivocation may represent an attempt at double-spending. As any p-block is cryptographically signed by p, an equivocation by p is a volitional fault of p, to which p can be held accountable.

When a block b observes another block b', and does not observe any equivocating block (a block b'' that together with b' forms an equivocation), we say that b **approves** b' (Def. 18). Note that a block b by a correct miner can observe two equivocating blocks b', b'', which means that b approves neither b' nor b'' (See Fig. 1.B). Block approval is not transitive. If b^+ approves b and b approves b', yet b^+ also observes b'' (which together with b' forms an equivocation), then b^+ does not approve b'.

Algorithm 1 Cordial Miners: Blocklace Utilities. Code for miner p.

Local variables:
  struct *block* b: ▷ The structure of a block b in a blocklace, Def. 14
    $b.creator$ – the miner that created b
    $b.payload$ – a set of transactions
    $b.pointers$ – a possibly-empty set of hash pointers to other blocks
  $blocklace \leftarrow \{\}$ ▷ The local blocklace of miner p
1: **procedure** *create_block*(d): ▷ Add to $blocklace$ a new block b pointing to its tips of depth $\leq d$
2: **new** b ▷ Allocate a new block structure
3: $b.payload \leftarrow payload()$ ▷ e.g., dequeue a payload from a queue of proposals (aka mempool)
4: $b.creator \leftarrow p$
5: $b.pointers \leftarrow hash(tips)$, where $tips$ are the tips of $blocklace\_prefix(d)$, at most two tips per miner ▷ Def. 19; two-tips limitation to prevent a Byzantine miner from flooding the blocklace before being excommunicated
6: **return** b
7: **procedure** $hash(b)$: **return** hash value of b ▷ Def. 14
8: **procedure** $b \succeq b'$: ▷ Def. 16, also refereed as b observes b'
9: **return** $\exists b_1, b_2, \ldots, b_k \in blocklace$, $k \geq 1$, s.t. $b_1 = b$, $b_k = b'$ and $\forall i \in [k-1]$: $hash(b_{i+1}) \in b_i.pointers$
10: **procedure** $closure(B)$: **return** $\{b' \in blocklace : b \in B \wedge b \succeq b'\}$ ▷ Def. 19. Also referred to as $[B]$. If $B = \{b\}$ is a singleton we use $[b]$ instead of $[\{b\}]$.
11: **procedure** $equivocation(b_1, b_2)$: ▷ Def. 17, Fig. 1.B
12: **return** $b_1.creator = b_2.creator \wedge b_1 \not\succeq b_2 \wedge b_2 \not\succeq b_1$
13: **procedure** $equivocator(q, B)$: ▷ Def. 17, Fig. 1; more faults can be added
14: **return** $(\exists b_1, b_2 \in B : b_1.creator = b_2.creator = q \wedge equivocation(b_1, b_2))$
15: **procedure** $correct\_block(b)$: ▷ See Def. 25; other conditions can be added
16: **return** $\{b'.creator : hash(b') \in b.pointers\}$ is a supermajority $\wedge \neg equivocator(b.creator, [b])$
17: **procedure** $approves(b, b_1)$: **return** $b_1 \in [b] \wedge \forall b_2 \in [b] : \neg equivocation(b_1, b_2)$ ▷ Def. 18, Fig. 1.C
18: **procedure** $ratifies(B_1, b_2)$: ▷ Def. 22, Fig. 1.C
19: **return** $\{b.creator : b \in [B_1] \wedge approves(b, b_2)\}$ is a supermajority
20: **procedure** $super\_ratifies(B_1, b_2)$: ▷ Def. 22, Fig. 1.A
21: **return** $\{b.creator : b \in [B_1] \wedge ratifies([b], b_2)\}$ is a supermajority
22: **procedure** $depth(b)$:
23: **return** max $\{k : \exists b' \in blocklace$ with a path from b to b' of length $k\}$. ▷ Def. 20
24: **procedure** $blocklace\_prefix(d)$: **return** $\{b \in blocklace : depth(b) \leq d\}$ ▷ Def. 20
25: **procedure** $cordial\_round(r)$:
26: **return** $\{b.creator : b \in blocklace \wedge depth(b) = r\}$ is a supermajority ▷ Def. 25
27: **procedure** $completed\_round()$:
28: **return** max $\{r : cordial\_round(r)\}$
29: **procedure** $last\_block(p)$: ▷ The p-block with the highest round
30: **return** $b \in blocklace$ s.t. $b.creator = p \wedge (\forall b' \in blocklace : b'.creator = p \implies b' \not\succ b)$

A miner p **approves** b' **in** a blocklace B, if p has a p-block b in B that approves b' (Def. 18). This holds even if p has a later p-block b^+ in B that observes an equivocation b' and b''. Namely, if miner p approves b' in B it also approves b' in any $B' \supset B$.

A miner p can approve both equivocating blocks b' and b'' in a blocklace B, but only if p is an equivocator. An example will be if B includes a p-block b that observes b' but not b'', and another block b^+ that observes b'' but not b', which can happen only if b and b^+ do not observe each other, namely form an equivocation.

The **closure** of a block b, denoted $[b]$, is the set of all blocks observed by b. The closure of a set of blocks B, denoted $[B]$, is the union of the closures of the blocks in B. A blocklace is **closed** if it does not contain "dangling pointers' (a pointer to a block that is not in the blocklace). In other words, B is closed if $B = [B]$. A block b is a **tip** of a blocklace B if there are no other blocks $b' \in B$ that observe b (Def. 19). The **depth** (or **round**) of a block b is the length of the longest path emanating from b. The **depth-d prefix of B**, denoted $B(d)$, is the set of all blocks with depth less than or equal to d. The **depth-d suffix of B**, is the set of all blocks with depth greater than d (Def. 20).

4.2 Blocklace Safety

Note that as equivocation is a fault, at most f miners may equivocate. Ensuring that the majority of correct miners approve a given block, requires approval from a **supermajority** of all miners, that is more than $\frac{n+f}{2}$ of the miners. A set of blocks is a **supermajority** if it includes blocks from a supermajority of miners (Def. 21). We show that there cannot be a supermajority approval of an equivocation.

A block b **ratifies** a block b' if the closure of b includes a supermajority of blocks that approve b'. A set of blocks B **super-ratifies** a block b', if it includes a supermajority of blocks that ratify b' (Def. 22 and Fig. 1).

The rounds in the blocklace are divided into **waves**, such that each wave has a fixed length of $w \geq 1$, defined as the **wavelength** (Def. 23), and the wave consists of all the blocks in those rounds. E.g., if the wavelength is 2, then the blocks in rounds 0 and 1 are in the first wave, and the blocks in rounds 3 and 4 are included in the second wave. We assume the existence of a leader selection function that chooses randomly for each wave w a single miner who will be the **leader** of that wave. A p-block b is a **leader block** of wave w if p is chosen as the leader of w and the blocklace contains b in the first round of w. E.g., if miner p is chosen as the leader of the first wave, and p has a block b in round 0, then b is the leader block of the first wave. We use leader blocks as part of the ordering function τ which is detailed in §5 and is used to totally order the blocklace.

Note that an equivocating leader can have several leader blocks in the same round. A leader block is **final** (Def. 24) if it is super-ratified within its wave, i.e., we say that the leader block b of round r is final if the blocklace prefix $B(r + w - 1)$ super-ratifies b.

The following notion of blocklace safety is the basis for the monotonicity of the blocklace ordering function τ, and hence for the safety of a protocol that uses τ for blocklace ordering.

▶ **Definition 2** (Blocklace Leader Safety)**.** *A blocklace B is **leader-safe** if every final leader block in B is ratified by every subsequent leader block in B.*

A sufficient condition for blocklace leader safety is for every block in the blocklace to acknowledge blocks by at least a supermajority of miners (see Fig. 2). Such a block is a **cordial block** and a blocklace with only cordial blocks is a **cordial blocklace** (Def. 25). A **correct block** b is a p-block s.t. b is a cordial block and p does not equivocate in $[b]$ (Def 26).

▶ **Proposition 3.** *A cordial blocklace is leader-safe.*

4.3 Blocklace Liveness

Next, we discuss conditions that ensure blocklace leader liveness.

▶ **Definition 4** (Blocklace Leader Liveness)**.** *A blocklace B is **leader-live** if for every block $b \in B$ by a miner not equivocating in B there is a final leader block in B that observes b.*

Given a blocklace, a set of miners P is (mutually) **disseminating** if every block by a miner in P is eventually observed by every miner in P (Def. 27). We show that dissemination is unbounded, meaning that if a set of miners P is disseminating in B then B is infinite, and in particular any suffix of B has blocks from any member of P. It follows that a cordial blocklace with a non-equivocating and disseminating supermajority of miners is leader-live (Fig. 3).

Algorithm 2 Cordial Miners: Ordering of a Blocklace with τ. Pseudocode for miner $p \in \Pi$, including Algorithms 1 & 4.

Local Variable:
$outputBlocks \leftarrow \{\}$
31: **procedure** $\tau()$: ▷ Called from Algorithm 3
32: $\tau'(last\_final\_leader())$
33: **procedure** $\tau'(b_1)$:
34: **if** $b_1 \in outputBlocks \vee b_1 = \emptyset$ **then return**
35: $b_2 \leftarrow previous\_ratified\_leader(b_1)$
36: $\tau'(b_2)$ ▷ Recursive call to τ'
37: **output** $xsort(b_1, [b_1] \setminus [b_2])$ ▷ Output a new equivocation-free suffix
38: $outputBlocks \leftarrow outputBlocks \cup xsort(b_1, [b_1] \setminus [b_2])$
39: **procedure** $xsort(b, B)$: ▷ Exclude equivocations and sort
40: **return** topological sort wrt $\succ$ of the set $\{b' \in B : approves(b, b')\}$
41: **procedure** $previous\_ratified\_leader(b_1)$:
42: **return** $arg_{b \in R} \max\ depth(b)$
43: where $R = \{b \in [b_1] \setminus \{b_1\} : b.creator = leader(depth(b)) \wedge ratifies([b_1], b)\}$
44: **procedure** $last\_final\_leader()$: ▷ Fig. 2
45: **return** $arg_{u \in U} \max\ depth(u)$ where
46: $U = \{b \in blocklace : b.creator = leader(depth(b)) \wedge final\_leader(b)\}$
47: **procedure** $final\_leader(b)$: ▷ Def. 24
48: **return** $super\text{-}ratifies((blocklace\_prefix(depth(b) + w - 1), b)$
procedure $leader()$ (Def. 23) and wavelength w are defined in Alg. 4.

▶ **Proposition 5** (Blocklace Leader Liveness Condition). *If $B \subset \mathcal{B}$ is a cordial blocklace with a non-equivocating and disseminating supermajority of miners, such that for every $r > 0$ there is a final leader block of round $r' > r$, then B is leader-live.*

5 Blocklace Ordering with τ

Here we present a deterministic function τ that, given a blocklace B, employs final leaders to topologically sort B into a sequence of its blocks, respecting $\succ$. The intention is that in a blocklace-based ordering consensus protocol, each miner would use τ to locally convert their partially-ordered blocklace into the totally-ordered output sequence of blocks.

The section concludes with Theorem 8, which provides sufficient conditions for the safety and liveness of any blocklace-based ordering consensus protocol that employs τ. The proof method is novel, in that it does not argue operationally, about events and their order in time, but rather about the properties of an infinite data structure – the blocklace. In the following section, we prove that the Cordial Miners protocols, which employ Alg. 2 that realizes τ, satisfy these conditions, and thus establish their safety and liveness. The operation of τ is depicted in Fig. 4.

We show that τ is monotonic, in that if it is repeatedly called with an ever-increasing blocklace then its output is an ever-increasing sequence of blocks. This monotonicity ensures finality, as it implies that any output will not be undone by a subsequent output. With τ, final leaders are the anchors of finality in the growing chain, each "writes history" backward till the preceding final leader.

The following recursive ordering function τ maps a blocklace into a sequence of blocks, excluding equivocations along the way. Formally, the entire sequence is computed backward from the last super-ratified leader, afresh by each application of τ. Practically, a sequence up to a super-ratified leader is final (Prop. 9) and hence can be cached, allowing the next call to τ with a new super-ratified leader to be computed backward only till the previously-cached

super-ratified leader, while producing as output all the blocks approved by the new super-ratified leader (the approval ensures that the new fragment does not introduce equivocations) that are not observed by the previously-cached final leader.

▶ **Definition 6** (τ)**.** *We assume a fixed topological sort function $xsort(b, B)$ (exclude and sort) that takes a block b and a blocklace B, and returns a sequence consistent with $\succ$ of all the blocks in B that are approved by b. The function $\tau : 2^{\mathcal{B}} \to \mathcal{B}^*$ is defined for a blocklace $B \subset \mathcal{B}$ backward, from the last output element to the first, as follows: If B has no final leaders then $\tau(B) := \Lambda$ (empty sequence). Else, let b be the last final leader in B. Then $\tau(B) := \tau''(b)$, where τ' is defined recursively:*

$$\tau'(b) := \begin{cases} xsort(b, [b]) & \text{if } [b] \text{ has no leader ratified by } b, \text{ else} \\ \tau'(b') \cdot xsort(b, [b] \setminus [b']) & \text{if } b' \text{ is the last leader} \\ & \text{ratified by } b \text{ in } [b] \end{cases}$$

Note that when τ' is called with a leader b, it makes a recursive call with a leader ratified by b, which is not necessarily super-ratified.

A pseudo-code implementation of τ is presented as Alg. 2. The algorithm is a literal implementation of the mathematics described above: It maintains *outputBlocks* that includes the prefix of the output τ that has already been computed. Upon adding a new block to its blocklace (Line 31), it computes the most recent final leader b_1 according to Definition 24, and applies τ to it, realizing the mathematical definition of τ (Def. 6), with the optimization, discussed above, that a recursive call with a block that was already output is returned. Hence the following proposition:

▶ **Proposition 7** (Correct implementation of τ)**.** *The procedure τ in Alg. 2 correctly implements the function τ in Definition 6.*

The following theorem provides a sufficient condition for the safety and liveness (Def. 1) of any blocklace ordering consensus protocol that employs τ, and thus offers conditions for solving the problem defined in §2:

▶ **Theorem 8** (Sufficient Condition for the Safety and Liveness of a Blocklace-Based Ordering Consensus Protocol)**.** *Assume a given blocklace-based consensus protocol that employs τ for ordering. If in every run of the protocol all correct miners have in the limit the same blocklace B that is leader-safe and leader-live, then the protocol is safe and live.*

Next, we provide a proof outline of Theorem 8.

τ Safety. A safe blocklace ensures a final leader is ratified by any subsequent leader, final or not. Hence the following:

▶ **Proposition 9** (Monotonicity of τ)**.** *Let B be a cordial blocklace with a supermajority of correct miners. Then τ is monotonic wrt the superset relation among closed subsets of B, namely for any two closed blocklaces $B_2 \subseteq B_1 \subseteq B$, $\tau(B_2) \preceq \tau(B_1)$.*

The following proposition ensures that if there is a supermajority of correct miners, which jointly create a cordial blocklace, then the output sequences computed by any two miners based on their local blocklaces would be consistent. This establishes the safety of τ under these conditions.

▶ **Proposition 10** (τ Safety)**.** *Let B be a blocklace with a supermajority of correct miners. Then for every $B_1, B_2 \subseteq B$, $\tau(B_1)$ and $\tau(B_2)$ are consistent.*

Algorithm 3 **Cordial Miners: Blocklace-Based Dissemination**
Code for miner p, including Algorithms 1, 2 & 4.

Local variables:
$r \leftarrow 0$ ▷ The current round of p, see Def. 20
49: **upon receipt** of $b : b.pointers \subseteq hash(blocklace) \wedge correct\_block(b)$ **do** ▷ Received "out of order" blocks are buffered; incorrect blocks are ignored
50: $blocklace \leftarrow blocklace \cup \{b\}$
51: $\tau()$ ▷ Defined in Algorithm 2
52: **if** $completed\_round() \geq r$ **then** ▷ Defined in Algorithm 1, line 27
53: $es\_advance\_round()$ ▷ Advance round conditions for ES, no-op for asynchrony. Defined in Algorithm 4
54: $b \leftarrow create\_block(completed\_round())$
55: $r \leftarrow depth(b)$ ▷ Advance round
56: **for** $q \in \Pi$ **do** ▷ Cordial Dissemination
57: **send** $\{b\} \cup blocklace\_prefix(r-2) \setminus [last\_block(q)]$ to q

Table 2 Cordial Miners' differences between Eventual Synchrony and Asynchrony.

Property	Asynchrony	Eventual Synchrony
Wavelength w:	5 (Line 58)	3 (Line 66)
Leader Selection:	Retrospective via coin toss (Line 61)	Prospective by a known order (Line 76)
Condition for advancing round:	None (Line 59)	*timeout* or finality conditions (Line 67)

τ Liveness. While τ does not output all the blocks in its input, as blocks not observed by the last final leader in its input are not in its output, the following observation and proposition set the conditions for τ liveness:

▶ **Observation 11** (τ output). *If a p-block $b \in B$ by a miner p not equivocating in B is observed by a final leader in B, then $b \in \tau(B)$.*

▶ **Proposition 12** (τ Liveness). *Let $B_1 \subset B_2 \subset \ldots$ be a sequence of finite blocklaces for which $B = \bigcup_{i \geq 1} B_i$ is a cordial leader-live blocklace. Then for every block $b \in B$ by a correct miner in B there is an $i \geq 1$ such that $b \in \tau(B_i)$.*

Thus, we conclude that the safety and liveness properties of τ carry over to Alg. 2.

Next, we prove that the two Cordial Miners consensus protocols – for eventual synchrony and asynchrony – satisfy the conditions of Theorem 8, and hence are safe and live.

6 The Cordial Miners Protocols

So far, we presented the blocklace and how a blocklace can be totally ordered using τ. Next, we show how miners disseminate their blocks to form a blocklace.

The shared components of the Cordial Miners protocols are specified via pseudocode in Algs. 1 (blocklace utilities), 2 (the ordering function τ), and 3 (dissemination). Alg. 4 details the differences between the Cordial Miners protocols for ES and asynchrony. We begin by explaining the dissemination protocol.

Algorithm 4 Cordial Miners: Specific Utilities. Code for miner p.

4.1 Procedures for Asynchrony

58: $w \leftarrow 5$
59: **procedure** *es_advance_round*(): ▷ No-op
60: **return**
61: **procedure** *leader*(d):
62: **if** $d \bmod w = 0$ **then**
63: **return** $q \in \Pi$ via a shared coin tossed at round $d + w - 1$
64: **else**
65: **return** $\bot$

4.2 Procedures for Eventual Synchrony

66: $w \leftarrow 3$
67: **procedure** *es_advance_round*():
68: **return** max r : *cordial_round*(r) $\wedge$ ▷ Last cordial round, Algorithm 1
69: (($r \bmod w = 0 \implies$ ▷ First round of the wave, leader is included in the round.
70: $\exists b \in blocklace : (leader(r) = b.creator) \wedge$
71: (($r \bmod w = 1 \implies$ ▷ Second round of the wave, round $r-1$ leader is ratified by round r blocks
72: $\exists b \in blocklace : (leader(r-1) = b.creator \wedge ratifies(blocklace\_prefix(r), b))) \wedge$
73: (($r \bmod w = 2 \implies$ ▷ Third round, round $r-2$ leader is super-ratified by r blocks
74: $\exists b \in blocklace : (leader(r-2) = b.creator \wedge super\text{-}ratifies(blocklace\_prefix(r), b)))$
75: $\vee$ *timeout*) ▷ Or timeout occurred. *timeout* is measured from when round r is cordial. This is p's estimation of Δ.
76: **procedure** *leader*(d):
77: **if** $d \bmod w = 0$ **then**
78: **return** $q \in \Pi$ selected deterministically
79: **else**
80: **return** $\bot$

6.1 Dissemination (Alg. 3)

A correct block is buffered until it has no dangling pointers, and then it is received (Line 49). We prove that an equivocating miner eventually can only produce incorrect blocks (Def. 26) and therefore is eventually excommunicated by all correct miners. After including a received block in its local blocklace, a miner calls τ (Line 51), which outputs new blocks if the received block results in the blocklace having a new final leader block.

If there is a new completed round in the blocklace (Line 52), the miner creates a new block b (Line 54), computes the new round (Line 55), and sends b to its fellow miners. The package sent to miner q contains any blocks up to the previous round that p knows that q might not know, based on the last block received from q (Line 57). Note that as the network is reliable, **send** is defined to be idempotent, namely to send each block to each miner at most once.

We note that there is a tradeoff between latency and message complexity, and there is a range of possible optimizations and heuristics. These are discussed in [22]. Here, we present a version of Cordial Miners protocols in which every block is communicated among every pair of correct miners in the worst case.

6.2 Specific utilities (Alg. 4)

Overview. There are several differences between the Cordial Miners protocols for ES and asynchrony, which are specified in Alg. 4 and summarized in Tab. 2.

First, in asynchrony, each wave consists of 5 rounds, and the leader block in the first round is chosen randomly using a shared coin tossed in the last round of the wave, i.e., the leader election is retrospective. We expand on the coin below. In ES, each wave has 3 rounds, and the leader block is elected in advance using any deterministic prospective method, e.g., round robin.

The reason a wave in asynchrony is longer is to counter the adversary: If the adversary knows in advance the leader block in the first round of the wave, it can manipulate block arrival times s.t. a wave with a final leader block will never happen. We prove that by using such coin at the last round of the wave, the adversary cannot affect the probability the the leader block is final. In ES, a wave consists of three rounds. We prove that this is sufficient to allow super-ratification of the leader block, making it final in case the leader is an honest miner.

Another difference is if an honest miner waits before proceeding to the next round when the current round becomes cordial. In asynchrony, the miner proceeds immediately to the next round when it is cordial (Line 59). In ES, a miner advances to the next round after a round either if *timeout* passes, or conditions for leader block finality occur (Line 67). The conditions are: if this is the first round of a wave, then the round contains the leader block (Line 69). If this is the second round, then the miner advances immediately if the round has a supermajority of blocks that ratifies the leader block (Line 71), and lastly, if the third round of a wave has a supermajority of blocks that super-ratifies the leader block (Line 73). These conditions are to prevent the adversary from ordering the messages after GST, in particular, the leader block and the blocks that super-ratify it, as the leader is known in advance.

Algorithm walkthrough. The *leader* (Lines 61, 76) procedure, which is called as part of τ, is an implementation of Def. 23.

The Cordial Miners asynchrony protocol, for which $w = 5$ (Line 58), elects leaders retrospectively using a shared random coin. To elect the leader of round r, when $r \bmod 5 = 0$, all correct miners toss the coin in round $r + 3$ and know in round $r + 4$ the elected leader of round r, as follows. We assume two **shared random coin** functions: *toss_coin* and *combine_tosses*. The function *toss_coin*(p_s, d) takes the secret key p_s of miner $p \in \Pi$ and a round number $d \geq 0$ as input, and produces p's share of the coin of round d, $s_{p,d}$, as output. If the protocol needs to compute the shared random coin for round d, then $s_{p,d}$ is incorporated in the payload of the d-depth p-block of every correct miner p. The function *combine_tosses*(S, d) takes a set S of shares $s_{p,d}$, $d \geq 0$, for which $|\{p : s_{p,d} \in S\}| > f + 1$, and returns a miner $q \in \Pi$. The properties of a similar function were presented in [21], which details how to implement such a coin as part of a distributed blocklace-like structure.

We formally define the shared coin in definition 28. Examples of such a coin implementation using threshold signatures [6, 23, 31] are in [10, 21]. The ES protocol elects leaders in a prospective manner via a fixed deterministic function, e.g., round-robin between the miners.

6.3 Correctness Proof Outline

The main theorem we prove is the following:

▶ **Theorem 13** (Cordial Miners Protocols Safety and Liveness). *The protocols for eventual synchrony and asynchrony specified in Algs. 1, 2, 3, & 4 are safe and live (Def. 1).*

We argue that in the limit the blocklaces of correct miners that participate in a run of a Cordial Miners protocol are identical, are leader-safe, and leader-live.

A formal description of blocklace-based protocols in terms of asynchronous multiagent transition systems with faults has been carried out in reference [28]. Here, we employ pseudocode, presented in Algorithms 1, 2, 3 & 4 to describe the correct behaviors of a miner in a protocol, and discuss only informally the implied multiagent transition system and its computations. A run of the protocol by the miners Π results in a sequence of configurations $\rho = c_0, c_1, \ldots$, each encoding the local state of each miner. A miner is *correct* in a run ρ if it behaves according to the pseudocode during ρ, *faulty* otherwise. As stated above, we assume that there are at most $f < n/3$ faulty miners in any run. We use $B_p(c)$ to denote the local blocklace of miner $p \in \Pi$ in configuration c, $B_p(\rho)$ to denote the blocklace of miner p in the limit, $B_p(\rho) := \bigcup_{c \in \rho} B_p(c)$, and $B(\rho)$ to denote the unions of the blocklaces of all correct miners in the limit, $B(\rho) := \bigcup_{p \in P} B_p(\rho)$, where $P \subseteq \Pi$ is the set of correct miners in run ρ.

We start by showing miner asynchrony (not to be confused with the model of asynchrony), that is, if a miner can create a block, then it can still create it regardless of additional blocks it receives from other miners. Miner asynchrony combined with the standard notion of *fairness*, that a transition that is enabled infinitely often in a run is eventually taken in the run, implies that once a Cordial Miners block creation transition is enabled then it will eventually be taken. We conclude that every miner p correct in a run produces the blocklace of the run, namely $B_p(\rho) = B(\rho)$. We can now argue the safety of the Cordial Miners protocols.

We now proceed to argue the liveness of the Cordial Miners protocols. We show that the Cordial Miners eventual synchrony protocol is leader-live with probability 1. We note that, following GST, the probability of a leader block being final is at least $\frac{|P|}{n}$, where $P \subseteq \Pi$ is the set of correct miners, and given that $w = 3$, if $\frac{|P|}{n} > \frac{2}{3}$, then the expected latency is at most $3/(2/3) = 4.5$ rounds.

The next proposition ensures that all correct miners eventually repel all equivocators and stop observing their blocks. We define an **equivocator-repelling** block recursively (Def. 29), through the set of blocks B that it acknowledges, terminating in an initial block, where $B = \emptyset$. Note that a block (or blocklace) that is equivocator-repelling may include equivocations, for example, two equivocating blocks each observed by a different block in B. However, once an equivocation by miner q is observed by a block b, q would be repelled: Any block that observes b would not acknowledge any q-block, preventing any further q-blocks from joining the blocklace. Also note that equivocators are eventually excommunicated since they eventually cannot produce correct blocks.

We argue in a lemma the existence of a blocklace **common core**, which is the blocklace-variant of the notion of a common core that appears in [3, 17]. Its proof is an adaptation to the cordial blocklace setting of the common core proof in [17], which in turn is derived from the proof of get-core in [3]. Fig. 5 illustrates its proof as well as the proof of the following Corollary about the existence of a super-ratified common core.

The lemma and corollary require an equivocators-free section of the blocklace, which may be the entire equivocation-free suffix of the blocklace as in the proof. But the proof also holds if there is a long enough stretch of rounds without equivocation, in which case a common core also exists. We conclude that if a Cordial Miners protocol relies on the common core for liveness dissemination, and cordiality are sufficient to ensure it. Finally, we complete the proof of liveness of the Cordial Miners protocols.

This concludes the proof outline that Cordial Miners is live and safe and thus completes the proof of Theorem 13.

7 Performance Analysis

We analyze the performance of Cordial Miners assuming the maximum number of Byzantine miners, i.e., $n = 3f + 1$. For the good case bit complexity, we assume $f \in O(1)$ and the network is synchronous.

Latency (See Table 1). Latency is defined as the number of blocklace rounds between every two consecutive final leaders, i.e., the number of blocklace rounds between two instances where τ outputs new blocks. This is also equivalent to the number of communication rounds since we do not use RB to disseminate blocks. The good case latency for both models is simply the wavelength.

For the expected case, in the asynchronous instance of the protocol, each wave w consists of 5 rounds. The probability that the leader block is final in the first round r of w, namely that a supermajority of the blocks in $r + 4$ each super-ratify the leader block at r is $\frac{2}{3}$. Therefore, in the expected case a leader block is final every 1.5 waves, and therefore the expected latency is $1.5w = 7.5$ rounds of communication.

The adversary can equivocate or not be cordial up to f times, but after each Byzantine process p equivocates, all correct processes eventually detect the equivocation and do not consider p's blocks as part of their cordial rounds when building the blocklace. Thus, in an infinite run, equivocations do not affect the overall expected latency.

In the ES version, each wave w consists of 3 rounds. The probability that the leader block is final is if the leader block is created by a correct miner, i.e., the probability is $\frac{2}{3}$, i.e., same as asynchrony. Thus, in the expected case, the latency is $1.5w = 4.5$ rounds.

Bit complexity. An equivocator is eventually excommunicated, and therefore eventually the number of equivocating blocks that are disseminated is limited. Each block in the blocklace is linear in size since it has a linear number of hash pointers to previous blocks. A Byzantine miner can cause the block it creates to be sent to all miners by all the other correct miners, causing the block's bit complexity to be $O(n^3)$ per such block. Thus, in the worst case, where $f \in \Theta(n)$, the asymptotic bit complexity is $O(n^3)$ per block. But, since the block size is $O(n)$, we can batch $O(n)$ transactions in it without increasing its asymptotic size. Therefore, we can amortize the bit complexity by a linear factor for each transaction, causing the amortized bit complexity per transaction to be $O(n^2)$ in the worst case.

For the good case in the ES version, where $f \in O(1)$ and the network is synchronous after GST, every block created by a correct miner is sent once from its creator to the other miners. Miners wait for *timeout* time after a round r is cordial before they move to the next round, which ensures that all blocks sent by correct miners in round r arrive to all other correct miners before they move to round $r + 1$. Therefore, blocks by correct miners in round $r + 1$ observe all blocks by correct miners in round r. Thus, the Byzantine miners can cause only a constant number of blocks per round to be sent by every correct miner to every other correct miner. Therefore, the bit complexity of sending each block in the good case is $O(n^2)$, and by batching $O(n)$ transaction per block, we get an amortized bit complexity of $O(n)$ per transaction.

8 Related Work

The use of a DAG-like structure to solve consensus has been introduced in previous works, especially in asynchronous networks. Hashgraph [4] builds an unstructured DAG, with each block containing two references to previous blocks, and on top of the DAG, the miners run

an inefficient binary agreement protocol. This leads to expected exponential time complexity. Aleph [20] builds a structured round-based DAG, where miners proceed to the next round once they receive $2f + 1$ DAG vertices from other miners in the same round. On top of the DAG construction protocol, a binary agreement protocol decides on the order of vertices to commit. Blockmania [13] uses a variant of PBFT [11] in the ES model and also uses reliable broadcast to disseminate blocks. Both protocols have higher latency than Cordial Miners since they use RB. GHOST [32], IOTA [25], and Avalanche [26] are DAG protocols for the permissionless model.

As mentioned in the introduction, the two state-of-the-art DAG-based protocols are DAG-Rider [21] and Bullshark [33]. DAG-Rider is a BAB protocol for the asynchronous model in which the miners jointly build a DAG of blocks, with blocks as vertices and pointers to previously created blocks as edges, divided into strong and weak edges. Strong edges are used for the commit rule, and weak edges are used to ensure fairness. Narwhal [14] is an implementation based on DAG-Rider for a relaxed networking model and works well assuming messages arrival is not bounded, but also not controlled by the adversary. Tusk [14] is a similar consensus protocol to DAG-Rider built on top of Narwhal. Bullshark [33] is a variation of DAG-Rider designed for the ES model with about half the latency of DAG-Rider. Cordial Miners outperform these protocols in terms of latency (for the same message complexity). Other DAG-based protocols include [12, 18], which are for a non-Byzantine failure model.

Another category of Byzantine consensus protocols is Leader-based. Examples include PBFT [11], Tendermint [8], HotStuff [34, 24], and VABA [1]. In these protocols, a designated leader proposes a block, sends them to the miners, and collects votes on its proposal, and a Byzantine leader can result in wasted time in which no blocks are output. Another difference is that these protocols are unbalanced in terms of the network as the leader is in charge of disseminating its block, collecting votes, and disseminating them, while the other miners only need to vote. On the other hand, DAG-based protocols like Cordial Miners are symmetric in that all miners perform exactly the same tasks.

9 Conclusion

We presented Cordial Miners, a family of low-latency, high-efficiency consensus protocols with instances for eventual synchrony and asynchrony. Cordial Miners achieve that by forgoing Reliable Broadcast and using the blocklace for the three major tasks of consensus – dissemination, equivocation exclusion, and ordering.

References

1 Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of Byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

3 Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

4 Leemon Baird. The swirlds Hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance. Report, Swirlds, 2016.

5 Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 52–61, 1993.

6 Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

7 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

8 Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

9 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

10 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

11 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, New Orleans, Louisiana, USA, 1999. USENIX Association.

12 Gregory V Chockler, Nabil Huleihel, and Danny Dolev. An adaptive totally ordered multicast protocol that tolerates partitions. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 237–246, 1998.

13 George Danezis and David Hrycyszyn. Blockmania: from block DAGs to consensus. *arXiv preprint arXiv:1809.01620*, 2018.

14 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

15 Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2721, 2021.

16 Sourav Das, Zhuolun Xiang, and Ling Ren. Near-optimal balanced reliable broadcast and asynchronous verifiable information dispersal. *Cryptology ePrint Archive*, 2022.

17 Danny Dolev and Eli Gafni. Some garbage in-some garbage out: Asynchronous t-byzantine as asynchronous benign t-resilient system with fixed t-trojan-horse inputs. *arXiv preprint arXiv:1607.01210*, 2016.

18 Danny Dolev, Shlomo Kramer, and Dalia Malki. Early delivery totally ordered multicast in asynchronous environments. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 544–553. IEEE, 1993.

19 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

20 Adam Gągol and Michał Świętek. Aleph: A leaderless, asynchronous, byzantine fault tolerant consensus protocol. *arXiv preprint arXiv:1810.05256*, 2018.

21 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

22 Idit Keidar, Oded Naor, and Ehud Shapiro. Cordial miners: A family of simple, efficient and self-contained consensus protocols for every eventuality. *arXiv preprint arXiv:2205.09174*, 2022.

23 Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.

24 Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive*, 2023.

25 Serguei Popov. The tangle. `https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf`, 2018.

26 Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

27 Maria A Schett and George Danezis. Embedding a deterministic BFT protocol in a block DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 177–186, 2021.

28 Ehud Shapiro. Multiagent transition systems: Protocol-stack mathematics for distributed computing. *arXiv preprint arXiv:2112.13650*, 2021.

29 Ehud Shapiro. Grassroots distributed systems: Concept, examples, implementation and applications. *arXiv preprint arXiv:2301.04391*, 2023.

30 Robert Shostak, Marshall Pease, and Leslie Lamport. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

31 Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

32 Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

33 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

34 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

A Formal Model

The following is a mathematical formal definition of the cordial miners consensus protocols.

▶ **Definition 14** (Block, Acknowledge). *A* ***block*** b *is a triple* $b = (p, a, H)$ *signed by* p*, referred to as a* p***-block****, s.t.* $p \in \Pi$ *is the miner that creates* b*,* $a \in \mathcal{A}$ *is the* ***payload*** *of* b*, and* H *is a finite set of hash pointers to blocks. Namely, for each* $h \in H$*,* $h = hash(b')$ *for some block* b'*. In which case we also say that* b ***acknowledges*** b'*. If* $H = \emptyset$ *then* b *is* ***initial****.*

▶ **Definition 15** (Blocklace). *Let* $\mathcal{B}$ *be the maximal set of blocks over* Π*,* $\mathcal{A}$*, and hash for which the induced directed graph* $(\mathcal{B}, \mathcal{E})$ *is acyclic. A* ***blocklace*** *over* $\mathcal{A}$ *is a set of blocks* $B \subseteq \mathcal{B}$*.*

▶ **Definition 16** ($\succ$, Observe). *Given two blocks* b, b'*, the strict partial order* $\succ$ *is defined by* $b' \succ b$ *if there is a nonempty path from* b' *to* b*. A block* b' ***observes*** b *if* $b' \succeq b$*. Given a blocklace* B*, Miner* p ***observes*** b ***in*** B *if there is a* p*-block* $b' \in B$ *that observes* b*. A group of miners* $Q \subseteq \Pi$ ***observes*** b ***in*** B *if every miner* $p \in Q$ *observes* b*.*

▶ **Definition 17** (Equivocation, Equivocator). *A pair of* p*-blocks* $b \neq b' \in \mathcal{B}$*,* $p \in \Pi$*, form an* ***equivocation*** *by* p *if they are not consistent wrt* $\succ$*, namely* $b' \not\succ b$ *and* $b \not\succ b'$*. A miner* p *is an* ***equivocator in*** B*, equivocator*(p, B)*, if* B *has an equivocation by* p*.*

▶ **Definition 18** (Approval). *Given blocks* $b, b' \in \mathcal{B}$*, the block* b ***approves*** b' *if* b *observes* b' *and does not observe any block* b'' *that together with* b' *forms an equivocation. A miner* $p \in \Pi$ ***approves*** b' ***in*** B *if there is a* p*-block* $b \in B$ *that approves* b'*. A set of miners* $Q \subseteq \Pi$ ***approve*** b' ***in*** B *if every miner* $p \in Q$ *approves* b' *in* B*.*

▶ **Definition 19** (Closure, Closed, Tip). *The* ***closure of*** $b \in \mathcal{B}$ ***wrt*** $\succ$ *is the set* $[b] := \{b' \in \mathcal{B} : b \succeq b'\}$*. The* ***closure of*** $B \subset \mathcal{B}$ ***wrt*** $\succ$ *is the set* $[B] := \bigcup_{b \in B} [b]$*. A blocklace* $B \subseteq \mathcal{B}$ *is* ***closed*** *if* $B = [B]$*. A block* $b \in \mathcal{B}$ *is a* ***tip*** *of* B *if* $b \notin [B \setminus \{b\}]$*.*

▶ **Definition 20** (Block Depth/Round, Blocklace Prefix & Suffix). *The* ***depth*** *(or* ***round****) of a block $b \in \mathcal{B}$, $depth(b)$, is the maximal length of any path of pointers emanating from b. For a blocklace $B \subseteq \mathcal{B}$ and $d \geq 0$, the* ***depth-d prefix of*** *B is $B(d) := \{b \in B : depth(b) \leq d\}$, and the* ***depth-$d$ suffix of*** *B is $\bar{B}(d) := B \setminus B(d)$.*

▶ **Definition 21** (Supermajority). *A set of miners $P \subset \Pi$ is a* ***supermajority*** *if $|P| > \frac{n+f}{2}$. A set of blocks B is a* ***supermajority*** *if the set of miners $P = \{p \in \Pi : \exists b \in B$ is a p-block$\}$ is a supermajority.*

▶ **Definition 22** (Ratified and Super-Ratified Block). *A block $b \in \mathcal{B}$ is (i)* ***ratified by a set of blocks*** *$B \subseteq \mathcal{B}$, if $[B]$ includes a supermajority of blocks that approve b; (ii)* ***ratified by a block*** *b if it is ratified by the set of blocks $[b]$; and (iii)* ***super-ratified*** *by blocklace $B \subset \mathcal{B}$ if $[B]$ includes a supermajority of blocks, each of which ratifies b*

▶ **Definition 23** (Wavelength, Leader Selection Function, Leader Block). *Given a* ***wavelength*** *$w \geq 1$, a* ***leader selection function*** *is a partial function $l : \mathbb{N} \mapsto \Pi$ satisfying (i)* ***coverage:*** *$\forall r \in \mathbb{N}$: $l(r) \in \Pi$ if r* ***mod*** *$w = 0$ else $l(r) = \perp$ and (ii)* ***fairness:*** *with probability 1 $\forall r \in \mathbb{N}, p \in \Pi\ \exists r' > r : l(r') = p$. A p-block b is a* ***leader block*** *if $l(depth(b)) = p$.*

▶ **Definition 24** (Final Leader Block). *Let $B \subseteq \mathcal{B}$ be a blocklace. A leader block $b \in B$ of round r is* ***final*** *in B if it is super-ratified in $B(r + w - 1)$.*

▶ **Definition 25** (Cordial Block, Blocklace). *A block $b \in \mathcal{B}$ of round r is* ***cordial*** *if $r = 1$ or it acknowledges blocks by a supermajority of miners of round $r - 1$. A blocklace $B \subset \mathcal{B}$ is* ***cordial*** *if all its blocks are cordial.*

▶ **Definition 26.** *A p-block $b \in \mathcal{B}$ in* ***correct****, if it is cordial and p does doe equivocate in $[b]$.*

▶ **Definition 27** (Disseminating). *Given a blocklace $B \subseteq \mathcal{B}$, a set of miners $P \subseteq \Pi$ is* ***mutually disseminating*** *in B, or* ***disseminating*** *for short, if for any $p, q \in P$ and any p-block $b \in B$ there is a q-block $b' \in B$ such that $b' \succ b$. The blocklace B is* ***disseminating*** *if it has a disseminating supermajority.*

▶ **Definition 28** (Shared Random Coin). *Assume some $d > 0$ and let $S = \{toss\_coin(p_s, d) : p \in P\}$ for a set of miners $P \subseteq \Pi$, $|P| > f + 1$. For the shared random coin, the function combine_tosses has the following properties:*

Agreement *If both $S', S'' \subseteq S$ and both $|S'|, |S''| > f + 1$, then $combine\_tosses(S', d) = combine\_tosses(S'', d)$*

Termination *$combine\_tosses(S, d) \in \Pi$.*

Fairness *The coin is fair, i.e., for every set S computed as above and any $p \in \Pi$, the probability that $p = combine\_tosses(S, d)$ is $\frac{1}{n}$.*

Unpredictability *If $S' \subset S$, $|S'| < f + 1$, then the probability that the adversary can use S' to guess the value of $combine\_tosses(S, d)$ is less than $\frac{1}{n} + \epsilon$.*

▶ **Definition 29** (Equivocator-Repelling). *Let $b \in \mathcal{B}$ be a p-block, $p \in \Pi$, that acknowledges a set of blocks $B \subset \mathcal{B}$. Then b is* ***equivocator-repelling*** *if p does not equivocate in $[b]$ and all blocks in B are equivocator-repelling. A blocklace B is* ***equivocator-repelling*** *if every block $b \in B$ is equivocator-repelling.*

B Figures

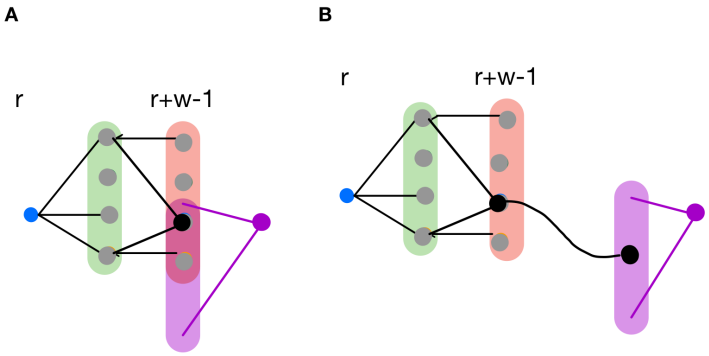

Figure 2 Finality of a Super-Ratified Leader (Definition 24): Assume that a leader block (blue dot) is super-ratified. A ratifying supermajority is represented by a thick red line, each member of which observes a possibly different approving supermajority represented by a green thick line. We show that the blue leader is ratified by any subsequent cordial leader. (A) The successive cordial leader (purple dot) is one round following the ratifying supermajority. Being cordial, it observes a supermajority (thick purple line) that must have an intersection (black dot) with the ratifying supermajority, hence it observes an approving supermajority and thus ratifies the blue leader. (B) A successive leader is more than one round following the ratifying supermajority. Being cordial, it observes a supermajority (thick purple line). There must be a correct miner common to the purple and red supermajority, with blocks in both (black dots); being a correct miner, its later block observes the earlier block (black line). Hence the purple leader observes the approving supermajority (via black lines) and hence ratifies the blue leader.

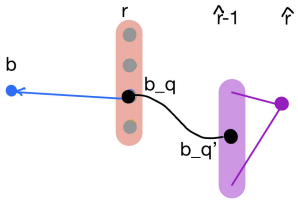

Figure 3 Liveness Condition, Proposition 5.

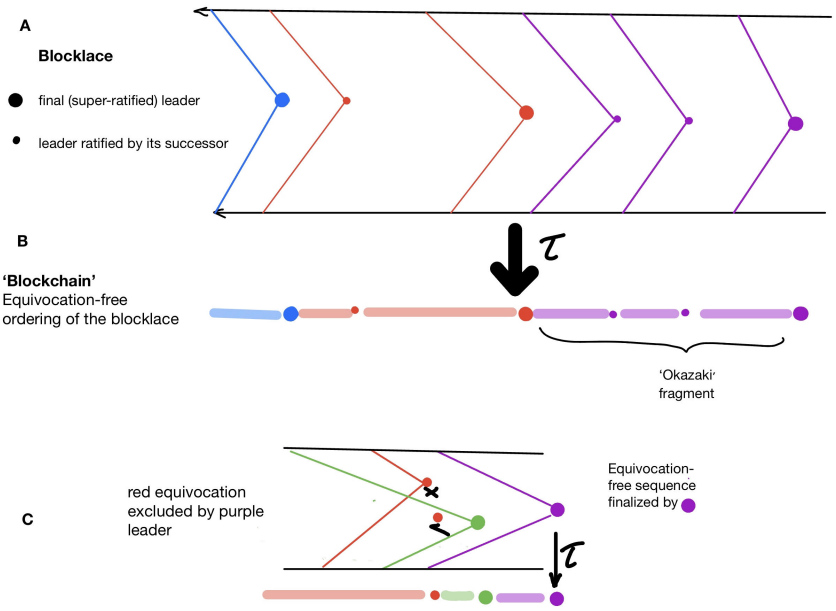

Figure 4 The Operation of τ, Safety and Liveness: (A) *The Input of* τ: A blocklace with final leaders (large dots) and leaders ratified by their successors (small dots). Each leader observes the portion of the blocklace below it (including the lines emanating from it). (B) *The Output of* τ: A sequence of blocks consisting of fragments. The sequence of fragments is computed recursively backward, starting from the last final leader, and back from each leader to the previous leader it ratifies. The input to computing the fragment consists of the portion of the blocklace observed by the current leader but not observed by the previous ratified leader. The output from each fragment is a sequence of blocks computed forward by topological sort of the input blocklace fragment, respecting $\succ$ and using the leader of the fragment to resolve and exclude equivocations. Final leaders are final, hence the backward computation starting from the last purple final leader need not proceed beyond the recursive call to the previous red final leader, as the output sequence up to the previous final leader has already been computed by the previous invocation of τ. **Safety Requirement:** A final leader (large dot) is ratified by any subsequent leader (large or small dot). **Liveness Requirement:** Any leader will eventually have a subsequent final leader (large dot) with probability 1. (C) *Leader-Based Equivocation Exclusion*: The green fragment created by the green leader includes the V-marked red block, since the green leader does not observe the red equivocation. However, the red X-marked red block is excluded from the purple fragment created by the purple leader, since the purple leader observes the equivocation among the two red blocks.

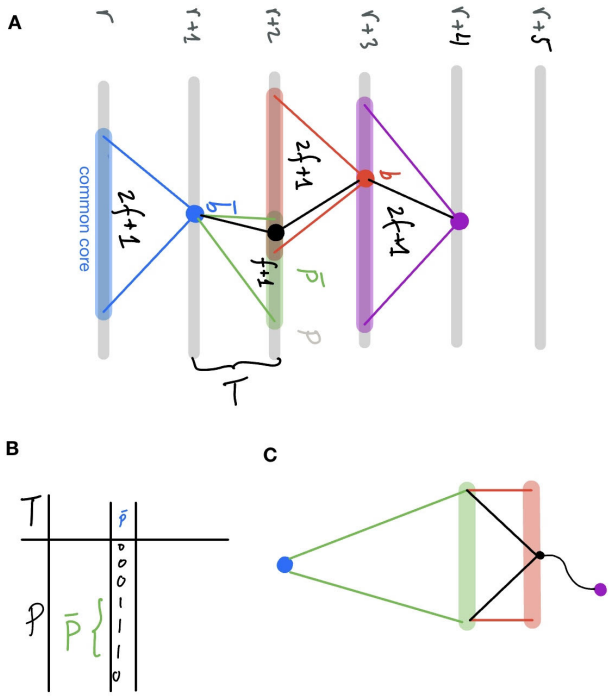

Figure 5 Common Core, Ratified Common Core, Safety and Liveness of Decision Rule for Asynchrony: (A) Rounds r to $r+3$ relate to the proof of the existence of a common core at round r is established. Round $r+4$ relates to establishing that all cordial blocks at round $r+4$ ratify all members of the common core of round r via a supermajority at round $r+3$. (B) *The common-core table* T used in the proof to relate rounds $r+1$ and $r+2$. (C) *The decision rule for asynchrony*: Protocol wavelength is 5. **Liveness:** Common-core ensures that the blue leader at round r is super-ratified by a red supermajority at round $r+4$ with probability $\frac{2f+1}{3f+1}$, thus ensuring liveness and expected latency of 6 rounds. **Safety:** A blue leader is approved by every cordial block at round $r+3$ (green) and hence is ratified by every cordial block at round $r+4$ (red) and beyond.