Dynamically available consensus on a DAG with fast confirmation for UTXO transactions

***Abstract*—This paper introduces a Byzantine Fault Tolerance (BFT) consensus mechanism for distributed networks with synchronized clocks. In this protocol, nodes propose blocks of transactions to be added to a directed acyclic graph (DAG) and generate commitments for fixed-duration time slots based on their DAG view. During network partitions, nodes update their *available* DAG. Post-recovery, slot commitment synchronization is achieved via a random coin, and the *finalized* DAG catches up with the available one. The protocol exhibits optimal fault tolerance resilience in an eventually synchronous model and enables fast UTXO-transaction confirmation during synchronous periods.**

***Index Terms*—Leaderless distributed ledgers, DAG-based consensus, dynamic availability, fast confirmation**

I. Introduction

The problem of ordering blocks or more generally events in a Byzantine distributed system is a fundamental challenge that has been extensively studied for many years. In recent years, a new promising approach has been proposed that distributes the responsibility for consensus among several nodes in the network and allows for balanced network utilization and high throughput. The idea is to collectively build a directed acyclic graph (DAG), where each vertex in the DAG represents a block of transactions and references to some previously received vertices. Through the logical interpretation of the locally maintained DAG, each node finds the deterministic order on the set of final blocks [1]–[9].

While the majority of consensus protocols sequentially order blocks and consequently transactions on the ledger, it has been shown [10] that a system, that enables participants to make simple payments, needs to solve a simpler task. In cases where payments are independent of one another (e.g. single-owned token assets or UTXO transactions), there is no necessity to order them, and a partial ordering is sufficient. This was observed in multiple papers [11], [12] and different solutions that utilize DAGs were suggested [7], [13].

As mentioned in [14], an important attribute of a modern consensus protocol is *dynamic availability*: the ability of the protocol to support an unknown number of nodes each of which can go to sleep and awake dynamically. While there are several dynamically available DAG-based protocols [15]–[18], all of them achieve probabilistic finality only.

A. Our contribution

Inspired by the IOTA 2.0 consensus protocol [19], we build a new DAG-based consensus protocol that achieves the following design goals: 1) supporting dynamic availability, 2) providing deterministic finality in an eventually synchronous network, 3) enabling fast confirmation for UTXO transactions during synchronous periods.

II. System model

The network comprises n nodes, with f *Byzantine* nodes. The remaining $n-f$ *correct* nodes strictly adhere to the protocol. Nodes utilize unique public keys for secure authentication. Time is divided into *slots* of $f+2$ *instances*, so each moment is encoded with a *timestamp* $\langle s,i\rangle$ of slot and instant indices with $s\in\mathbb{N}$ and $1\le i\le f+2$. Nodes use *synchronized* clocks.

Communication between correct nodes occurs over secure channels, albeit with adversary-controlled delays described below. Before Global Stabilization Time GST (unknown to nodes), communication is asynchronous; after GST, every block sent by a correct node is received by any other correct node before the next instant. Before Global Awake Time GAT (unknown to nodes), nodes follow a sleep-awake cycle (start sleeping and wake up after the last instant of a slot), whereas after GAT all correct nodes are awake. For $GST>0$ and $GAT=0$ this mimics an *eventually synchronous* model. For $GST=0$, this corresponds to a *dynamically available lock-step synchronous* model. At the last instant of each slot $\langle s,f+2\rangle$, an oracle's random value determines the slot leader for synchronization.

The above model operates under a permissioned setting, contrasting with IOTA 2.0's permissionless network. These n nodes issue blocks with transactions and can be regarded as a validator committee in IOTA's peer-to-peer network where every user can create their block with a transaction and gossip it. The system's integrity hinges on synchronized clocks, approximated in the real network using Network Time Protocol [20]. The common coin mechanism, vital for demonstrating the liveness of our finalization mechanism, is not a part of the actual IOTA 2.0 protocol.

A. Problem statement

We consider a notion of *finality* for blocks, which allows nodes to linearly sequence final blocks.

Problem 1: Design a DAG-based consensus protocol that satisfies two requirements. **Safety:** For the sequences of final blocks by any two correct nodes, one has to be the prefix of another. **Liveness:** If a correct node creates a block, then the block becomes eventually final.

Definition 1 (UTXO transaction and double-spend): A UTXO is a pair consisting of a positive value and an account that can consume this UTXO. A single-owned UTXO transaction is a tuple containing (i) UTXO inputs, which can be consumed by one account, (ii) the account signature, and (iii) UTXO outputs. The sum of values in the inputs and the outputs are the same. Two UTXO transactions are called a double-spend if they attempt to consume the same UTXO.

We consider a notion of *confirmation* for UTXO transactions, which must satisfy the condition that if any correct node regards a UTXO transaction as confirmed, then all correct nodes eventually regard it as confirmed.

Problem 2: Integrate a payment system in a DAG-based consensus protocol that satisfies two requirements. **Safety:** Two UTXO transactions that constitute a double-spend are never both confirmed. **Liveness:** Every UTXO transaction created by an honest account is eventually confirmed.

III. Protocol

In the protocol, each block B contains multiple fields, including $B.refs$ (a list of hash references to previous blocks), $B.commit$ (a slot commitment - see Definition 7), $B.txs$ (transactions), $B.node$ (a node's unique identifier), $B.time$ (a timestamp). The set of all blocks constitutes a DAG which starts at block $genesis$. A block B (or a transaction in B) is called *reachable* from a block C if one can reach B starting at C and traversing the DAG along the directed edges, e.g. B is reachable from B.

Definition 2 (Past cone): A past cone $cone(B)$ of block B is the set of all blocks reachable from B.

In the protocol, described later, a correct node issues a block every instant and always references its previous block. Thus, all blocks of a correct node are linearly ordered.

Definition 3 (Equivocation): A node is an equivocator if two of the node's blocks are not linearly ordered in the DAG.

We assume that if a correct node detects an equivocation, then this node includes proof for the act of equivocation in the next created block. After receiving this block, other correct nodes could get familiar with the equivocator.

Definition 4 (Supermajority): A set of blocks S is called a supermajority if $|\{B.node : B \in S\}| \geq (n+f)/2$.

The following two definitions are inspired by Hashgraph [6] and Cordial Miners [2], two other DAG-based consensus protocols that use best-effort broadcast.

Definition 5 (Approval): A block B approves a block C if C is reachable from B in the DAG and no block with the proof of equivocation of $C.issuer$ is reachable from B. Similarly, a UTXO transaction is approved by a block B, if this transaction is reachable from B and no block with a double-spend for this transaction is reachable from B.

Definition 6 ((Super)ratification): A block B (a UTXO transaction tx) is said to be ratified by a block C if C approves a supermajority of blocks, each of which approves block B (transaction tx). A UTXO transaction tx is super-ratified if $|C.node : C \text{ ratifies } tx| \geq (n+f)/2$.

Each node maintains an *available* DAG and *finalized* DAG. An available DAG after instant i of slot s is defined to be the past cone of the block created at instant i by the node maintaining this DAG. A finalized DAG is a prefix of the available DAG consisting of final blocks.

Definition 7 (Slot commitment): A commitment σ_s for a slot s is defined based on the available DAG at the moment $\langle s+1, f+2\rangle$ in an iterative way: $\sigma_{-1} = 0; \sigma_0 = \text{hash}(0, genesis)$; for $s \geq 1$, $\sigma_s = \text{hash}(\sigma_{s-1}, Blocks_s)$, where $Block_s$ is the concatenation of all blocks in the available DAG that are created at or before slot s and don't include any of the commitments in the chain $(\sigma_{-1}, \sigma_0, \ldots, \sigma_{s-1})$.
If two correct nodes agree on σ_s, they have the same prefix of their available DAGs and, consequently, know the same set of equivocators (which can be extracted from that prefix).

Definition 8 (Quorum certificate): A quorum certificate (QC) is defined as a set of blocks in the available DAG that are issued by $(n+f)/2$ distinct nodes at the same slot. The timestamp of a QC is the timestamp of the latest block that is reachable by all blocks in the QC. Hereafter, when referring to a QC, we mean the QC with the *latest* timestamp, and this QC can be deterministically found from the available DAG.

A. Protocol description

We use $\langle s, i\rangle$ to denote the current timestamp. During slot s, a node adapts one slot commitment chain $(\sigma_{-1}, \sigma_0, \ldots, \sigma_{s-2})$. Switching the slot commitment chain could happen only at the slot boundary and it is important for synchronization after the asynchronous period ends. Broadcasting blocks during $f+2$ instances of slot s is used to generate identical slot commitment σ_{s-1} by all correct nodes at that chain after GST. The following delves into different parts of the protocol.

Creating a block: When creating a new block, the node includes the timestamp $\langle s, i\rangle$, a subset of transactions, the currently adopted slot commitment σ_{s-2}. It also references all the blocks in the available DAG from the previous instant (which is either $\langle s, i-1\rangle$ for $i > 1$ or $\langle s-1, f+2\rangle$ otherwise).

Best-effort broadcast: The key principle in broadcasting a newly created block is to ensure that the recipient knows the past cone of the block. Each node needs to track the history of the known blocks by every other node, e.g. for node η, this set is denoted as $history[\eta]$. So, when sending a newly created block B, a node sends to node η the set $cone(B) \setminus history[\eta]$ and updates $history[\eta] \leftarrow history[\eta] \cup cone(B)$. A similar history update happens after receiving blocks.

Updating the available DAG: When receiving a new block, a node puts this block into a buffer of blocks. Once the buffer contains a block B, which is created at $\langle s, i\rangle$, together with its past cone, the node updates the available DAG with $cone(B)$ if all the following conditions are met. **CU1:** No block at slot s in $cone(B)$ is created by equivocator (known based on σ_{s-2}). **CU2:** Block B contains commitment σ_{s-2} (the same as the node adapts) and the correctness of computation of this commitment is verified by inspecting $cone(B)$. **CU3:** For every block C in $cone(B)$ which is not committed to the chain $(\sigma_{-1}, \sigma_0, \ldots, \sigma_{s-2})$ and created at slot $s-1$ or before, the node computes its *reachable number* $reachNum(C, \langle s, i\rangle)$ - the number of distinct nodes who created blocks at slot s from which one can reach C. For every such C, it holds that $reachNum(C, \langle s, i\rangle) \geq i$.

Chain switching rule: Before moment $\langle s, f+2\rangle$, the node generates the commitment σ_{s-1} based on its available DAG and includes it in the next block. At moment $\langle s, f+2\rangle$, the node receives the random value from the oracle which determines who is the leader for synchronization. If the node decides to switch its slot commitment chain to the one of the leader, then it updates its available DAG with the past cone of the leader block. At the slot boundary between slot s and slot $s+1$, the node does not switch the chain if one of the following conditions holds. **CS1:** the number of non-equivocator nodes that generated the same commitment σ_{s-1} is at least than $(N_s+1)/2$, where N_s is the total number of nodes from whom the node receives the block with timestamp $\langle s, f+2\rangle$. **CS2:** The timestamp of the latest QC of the leader is smaller than the one maintained by the node.

Finality: A slot commitment σ_s becomes final when there exists a supermajority of blocks including σ_s at the same slot, that ratify a block including σ_s. In that case, all blocks committed to σ_s become final. Then one can apply any deterministic ordering on the set of final blocks, which satisfies two rules. **R1:** The ordering respects the causal order relationships of blocks in the finalized DAG. **R2:** any block committed to σ_{s-1} is ordered before any block committed to σ_s.

Confirmation: A UTXO transaction becomes confirmed if it is super-ratified.

References

[1] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, “Narwhal and tusk: a dag-based mempool and efficient bft consensus,” in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[2] I. Keidar, O. Naor, and E. Shapiro, “Cordial miners: A family of simple, efficient and self-contained consensus protocols for every eventuality,” *arXiv preprint arXiv:2205.09174*, 2022.

[3] A. Gagol and M. Świetek, “Aleph: A leaderless, asynchronous, byzantine fault tolerant consensus protocol,” *arXiv preprint arXiv:1810.05256*, 2018.

[4] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, “Bullshark: Dag bft protocols made practical,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2705–2718.

[5] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, “All you need is dag,” in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, pp. 165–175.

[6] L. Baird, “The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance,” *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, vol. 34, pp. 9–11, 2016.

[7] D. Malkhi, C. Stathakopoulou, and M. Yin, “Bbca-chain: One-message, low latency bft consensus on a dag,” *arXiv preprint arXiv:2310.06335*, 2023.

[8] A. Spiegelman, B. Aurn, R. Gelashvili, and Z. Li, “Shoal: Improving dag-bft latency and robustness,” *arXiv preprint arXiv:2306.03058*, 2023.

[9] K. Babel, A. Chursin, G. Danezis, L. Kokoris-Kogias, and A. Sonnino, “Mysticeti: Low-latency dag consensus with fast commit path,” *arXiv preprint arXiv:2310.14821*, 2023.

[10] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi, “The consensus number of a cryptocurrency,” in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 307–316.

[11] M. Baudet, G. Danezis, and A. Sonnino, “Fastpay: High-performance byzantine fault tolerant settlement,” in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 163–177.

[12] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, “Scalable and probabilistic leaderless bft consensus through metastability,” *arXiv preprint arXiv:1906.08936*, 2019.

[13] A. Lewis-Pye, O. Naor, and E. Shapiro, “Flash: An asynchronous payment system with good-case linear communication complexity,” *arXiv preprint arXiv:2305.03567*, 2023.

[14] J. Neu, E. N. Tas, and D. Tse, “Ebb-and-flow protocols: A resolution of the availability-finality dilemma,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 446–465.

[15] C. Li, F. Long, and G. Yang, “Ghast: Breaking confirmation delay barrier in nakamoto consensus via adaptive weighted blocks,” 2020.

[16] H. Yu, I. Nikolić, R. Hou, and P. Saxena, “Ohie: Blockchain scaling made simple,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 90–105.

[17] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, “Prism: Deconstructing the blockchain to approach physical limits,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 585–602.

[18] M. Fitzi, P. Ga, A. Kiayias, and A. Russell, “Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition,” *Cryptology ePrint Archive*, 2018.

[19] IOTA Foundation, “Consensus on a dag,” 2023, accessed: 19.11.2023. [Online]. Available: https://wiki.iota.org/learn/protocols/iota2.0/core-concepts/consensus/introduction/

[20] D. L. Mills, *Computer network time synchronization: the network time protocol on earth and in space*. CRC press, 2017.