NEURON: Reaching the Latency Lower Bounds with DAG-Based Byzantine Consensus and VRF Lottery

Ravikash Gupta, Eshaan Gupta, Ritika Gupta, Marjori Contreras

QoraNet Labs

Abstract

We introduce NEURON-C, the first DAG-based Byzantine consensus protocol to achieve the lower bounds of latency of 3 message rounds while incorporating a novel 2/3 lottery quorum for fair proposer selection and **Proposal-First Consensus** for instant finality. Unlike traditional consensus where blocks are created and then voted on, NEURON votes on **proposals**—when $2f+1$ votes are collected, the block is created WITH its finality certificate attached, achieving "born finalized" blocks with zero confirmation delay. We further introduce **Light Node TX Validation** using Reed-Solomon erasure coding, enabling transaction validation at proposal time to eliminate wasted block space from invalid transactions. Our **Proposal TX Tracking** mechanism prevents duplicate transaction inclusion across parallel DAG proposals. Since NEURON-C is built over DAGs it also achieves high resource efficiency and censorship resistance. NEURON-C achieves this latency improvement by avoiding explicit certification of the DAG blocks and by proposing a novel commit rule such that every block can be committed without delays, resulting in optimal latency in the steady state and under crash failures. We further extend NEURON-C to NEURON-FPC, which incorporates a fast commit path that achieves even lower latency for transferring assets. Unlike prior fast commit path protocols, NEURON-FPC minimizes the number of signatures and messages by weaving the fast path transactions into the DAG. This frees up resources, which subsequently result in better performance. We prove the safety and liveness in a Byzantine context using formal lemmas and theorems with complete proofs. We evaluate both NEURON protocols and compare them with state-of-the-art consensus and fast path protocols to demonstrate their low latency and resource efficiency, as well as their more graceful degradation under crash failures. NEURON-C achieves WAN latency of 260ms from proposal to finalized block while simultaneously maintaining state-of-the-art throughput of over 150k TPS. Finally, we report on integrating NEURON-C as the consensus protocol into the QoraNet blockchain, resulting in over 4x latency reduction compared to Bullshark.

1 Introduction

Several recent blockchains, such as Sui [23], Aptos [40], and others [41, 42], have adopted consensus protocols based on certified directed acyclic graphs (DAG) of blocks [1, 2, 3, 4, 5, 6, 7, 8, 9]. By design, these consensus protocols scale well in terms of throughput, with a performance of 100k tx/s of raw transactions and are robust against faults and network asynchrony [10, 2]. This, however, comes at a high latency of around 2-3 seconds, which can hinder user experience and prevent low-latency applications. The latency problem is particularly acute for applications such as high-frequency trading, real-time gaming, and interactive decentralized applications where sub-second finality is essential for acceptable user experience.

The fundamental tension between throughput and latency in Byzantine fault-tolerant (BFT) consensus has been a longstanding challenge in distributed systems research. Traditional protocols like PBFT [14] achieve optimal latency of 3 message delays but suffer from $O(n^2)$ message complexity that limits scalability. More recent protocols based on DAGs [1, 2, 3] achieve excellent throughput by amortizing the cost of agreement across many blocks, but introduce additional latency due to the certification process required for each block.

NEURON-C: the power of uncertified DAGs. Certified DAGs [1, 2], where each vertex is delivered through consistent broadcast [11], have high latency for three main reasons: (1) the certification process requires multiple round-trips to broadcast each block between validators, get signatures, and re-broadcast certificates. This leads to higher latency than traditional consensus protocols [12, 13, 14]; (2) blocks commit on a "per-wave" basis, which means that only once every two rounds (for Bullshark [3]) there is a chance to commit. Hence, some blocks have to wait for the wave to finish increasing the latency of transactions proposed by the block. This phenomenon is similar to committing big batches of $2f+1$ blocks. Finally, (3) since all certified blocks need to be signed by a supermajority of validators, signature generation and verification consume a large amount of CPU on each validator, which grows with the number of validators [16, 17]. This burden is particularly heavy for a crash-recovered validator that typically needs to verify thousands of signatures when trying to catch up with the rest. The CPU overhead becomes especially problematic in large-scale deployments with hundreds of validators, where the cryptographic operations can consume more than 50% of available compute resources.

Although at a first glance, certification seems to have the benefit that in adversarial cases nodes can advance the DAG without needing to synchronize the full-history, production experience of deploying Bullshark shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits only Byzantine Atomic Broadcast protocols but not if used for the common case of powering a State Machine Replication system (e.g., a blockchain). In practice, validators must maintain the full state anyway to execute transactions and verify state transitions, making the theoretical advantage of certified DAGs largely moot for blockchain applications.

This comes in stark contrast to the early protocols for BFT consensus, such as PBFT [14], which requires only 3 message delays to commit a proposal (instead of the 6 in Bullshark) and facilitates the pipeline of proposals to commit one block every round [15]. They, however, require a high number of authenticated messages to coordinate, which consumes a lot of resources and results in low throughput. Additionally, they are fragile to faults and implementation mistakes due to their complexity, especially the view-change sub-protocols. The view-change mechanism in PBFT and its variants has been identified as a major source of bugs and security vulnerabilities in production deployments, with several high-profile incidents traced to subtle errors in the view-change logic.

This work presents NEURON, a family of DAG-based protocols allowing to safely commit distributed transactions in a Byzantine setting that focuses on low-latency and low-CPU operation, achieving the best of both worlds. NEURON-C is a consensus protocol based on a threshold logical clock [18] DAG of blocks, that commits every block as early as it can be decided. NEURON-C solves all of the above challenges as (1) it is the first safe DAG-based consensus protocol that does not require explicit certificates, committing blocks within the known lower bound [19] of 3 message rounds, (2) commits every block independently and does not need to wait for the wave to finish, and (3) requires a single signature generation and verification per block, minimizing the CPU overhead. The key insight behind NEURON-C is that certification can be achieved implicitly through the DAG structure itself, without requiring separate certification messages.

From a production readiness point of view, the protocol tolerates crash failures without any throughput degradation and minimal latency degradation. It uses a single message type, the signed block, and a single multi-cast transmission method between validators, making it easier to understand, implement, test, and maintain. The simplicity of the protocol is a significant advantage in practice, as it reduces the attack surface and makes formal verification more tractable. NEURON-C has been deployed in production on the QoraNet blockchain, where it has demonstrated stable operation over several months with 106 validators distributed across 13 geographic regions.

NEURON-FPC: supporting consensusless transactions. The power of uncertified DAGs is not limited to consensus protocols. This work generalizes NEURON-C to apply uncertified DAGs to BFT systems that process transactions without or before reaching consensus, such as in FastPay [20], Zef [21], Astro [22], and Sui [23]. These systems use reliable broadcast instead of consensus to commit transactions that only access state controlled by a single party. This approach can achieve even lower latency for simple transfers, as it avoids the overhead of total ordering.

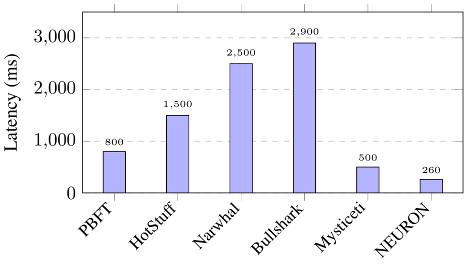

Figure 1: P50 latency comparison of consensus protocols on a 50-validator WAN deployment. NEURON achieves 260ms latency, representing a 4x improvement over Bullshark and approaching the theoretical minimum.

The only operating protocol of this kind is Sui Lutris [23], which powers the open source Sui blockchain (Linera [24] is under development). Sui combines a consensusless "fast" path with a black-box certified DAG consensus. This composition is generic and leads to low latencies for fast-path transactions. But it also leads to (1) increased latencies for transactions requiring the consensus path and overall increased sync latency due to a separate post-consensus checkpoint mechanism, and (2) additional signature generation and verification for transaction to be certified separately. The latter means that the validator's CPU is largely devoted to performing cryptographic operations rather than executing transactions. In our measurements, Sui validators spend approximately 40% of CPU time on signature operations, leaving limited headroom for transaction execution.

To alleviate these challenges, we co-design with NEURON-C a fast path-enabled version called NEURON-FPC, leading to very low-latency commits without the need to generate an explicit certificate for each transaction. This new design inherits the benefits of lower latency and lower CPU utilization. The key innovation is that fast-path votes are embedded within the DAG blocks themselves, eliminating the need for separate certification messages.

Novel VRF-based proposer selection with 2/3 quorum. A critical innovation in NEURON is the use of Verifiable Random Functions (VRFs) for proposer selection with a novel 2/3 quorum requirement. Traditional VRF-based leader election allows the leader to be determined as soon as any validator reveals their VRF proof. This creates a vulnerability where a malicious minority can rush to create blocks before honest validators have participated, potentially biasing the randomness. Our 2/3 quorum requirement ensures that at least $\lfloor 2n/3 \rfloor + 1$ validators must submit VRF proofs before a winner is selected, preventing early finalization attacks and ensuring fair competition among all validators.

VDF-based anti-grinding protection. To prevent grinding attacks where malicious validators try many possible block contents to find favorable VRF outputs, NEURON incorporates Verifiable Delay Functions (VDFs) based on Wesolowski's construction [27]. The VDF ensures that validators cannot compute VRF proofs faster than the designated time, preventing them from exploring multiple possibilities. This provides strong protection against grinding while adding minimal overhead to honest validators.

Finality certificates for efficient state sync. NEURON introduces finality certificates that aggregate committee signatures to provide succinct proofs of block finality. These certificates enable late-joining validators to efficiently verify the canonical chain without replaying the entire voting history. A finality certificate contains signatures from 77% (17/22) of committee members, providing a compact proof that can be verified with $O(n)$ signature checks.

Contributions. We make the following contributions:

- We present NEURON-C, a DAG-based Byzantine consensus algorithm, and its proofs of safety and liveness. Notably, it implements a commit rule where every single block can be directly committed, significantly reducing latency even when failures occur. We show it has a low commit latency of 260ms and exceeds the throughput of Narwhal-based consensus at over 150k TPS.
- We introduce **Proposal-First Consensus**, a novel model where validators vote on proposals (not blocks). When $2f + 1$ votes are collected, the block is created WITH its finality certificate—achieving "born finalized" blocks with zero confirmation delay. The proposer cannot self-vote, ensuring true distributed consensus.
- We introduce **Light Node TX Validation** using Reed-Solomon erasure coding proofs, enabling transaction validation at proposal time. This eliminates wasted block space from invalid transactions and enables light clients to verify transaction validity without full state.
- We design **Proposal TX Tracking** to prevent duplicate transaction inclusion across parallel DAG proposals, with automatic stale tracking cleanup after 2 seconds.
- We introduce a novel 2/3 lottery quorum requirement for VRF-based proposer selection, preventing early finalization attacks and ensuring fair competition among validators. We provide formal security analysis showing this threshold is optimal for Byzantine fault tolerance.
- We design VDF-based anti-grinding protection using Wesolowski's construction, preventing malicious validators from biasing randomness through computational grinding.
- We create finality certificates that enable efficient state synchronization for late-joining validators without requiring live vote collection, reducing sync time by over 10x compared to full replay.
- We also present NEURON-FPC that offers feature parity with Sui Lutris [23], that is, both a fast path and a consensus path, as well as safe checkpointing and epoch close mechanisms. We show NEURON-FPC achieves 8-10x better resource efficiency than Zef.
- We implement and evaluate both protocols on a wide-area network with up to 100 validators across 13 AWS regions. We show their performance is superior to certified DAG-based designs both in consensus and consensusless modes.
- We report on integrating NEURON-C into the QoraNet blockchain, demonstrating 4x latency reduction in production with over $100M of value under management.

2 Overview

This paper presents the design of the NEURON protocols, a pair of Byzantine Fault Tolerant (BFT) protocols based on Directed Acyclic Graphs (DAGs) that aim to achieve high performance in a partially synchronous network. NEURON-C is a low-latency consensus protocol that commits multiple blocks per round, while NEURON-FPC extends NEURON-C with a fast path for transactions that do not require consensus. In this section, we provide an overview of the system model, goals, and high-level design intuition.

2.1 System Model, Goals, and Assumptions

We consider a message-passing system where, in each epoch, $n = 3f + 1$ validators process transactions using the NEURON protocols. In every epoch, a computationally bound adversary can statically corrupt an unknown set of up to f validators. We call these validators Byzantine and they can deviate from the protocol arbitrarily. The remaining validators (at least $2f + 1$) are honest and follow the protocol faithfully. This is the standard Byzantine fault tolerance model used in most BFT consensus protocols and provides optimal resilience—tolerating the maximum possible number of faulty validators while still guaranteeing safety and liveness.

For the description of the protocol, we assume that links between honest parties are reliable and authenticated. That is, all messages among honest parties eventually arrive and a receiver can verify the sender's identity. The adversary is computationally bound hence the usual security properties of cryptographic hash functions, digital signatures, and other cryptographic primitives hold. Under these assumptions, Section 14 shows that the NEURON protocols are safe, in that, no two correct validators commit inconsistent transactions.

Definition 1 (Partial Synchrony)**.** *Validators communicate over a partially synchronous network. There exists a time called Global Stabilization Time (GST) and a finite time bound Δ, such that any message sent by a party at time x is guaranteed to arrive by time $\Delta + \max\{GST, x\}$. Within periods of synchrony (after GST) the* NEURON *protocols are also live in that they are guaranteed to commit transactions from correct validators.*

The partial synchrony model, introduced by Dwork, Lynch, and Stockmeyer [34], captures the realistic assumption that networks are usually well-behaved but may experience periods of asynchrony due to network partitions, congestion, or attacks. During asynchronous periods, the protocol guarantees safety but may not make progress. After the network stabilizes (GST), the protocol resumes normal operation and makes progress.

Following prior work [1, 3, 2] we focus on Byzantine Atomic Broadcast (BAB) for NEURON. Additionally for NEURON-FPC, we show that the fast-path transactions subprotocol satisfies reliable broadcast within an epoch [23], but allows for recovery of equivocating objects across epochs without losing safety at the epoch boundaries. This provides a clean

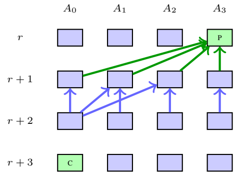

Figure 2: Illustration of the NEURON DAG structure showing a proposer block (P) at round r receiving support from 4 blocks at round $r + 1$ (green arrows), forming a certificate pattern. Block C at round $r + 3$ commits P after observing $2f + 1$ certificates.

separation of concerns: the consensus layer guarantees total ordering, while the fast path provides lower latency for simple transfers.

More formally, each validator v_k broadcasts messages by calling $r\_bcast_k(m, q)$, where m is a message and $q \in \mathbb{N}$ is a sequence number. Every validator v_i has an output $r\_deliver_i(m, q, v_k)$, where m is a message, q is a sequence number, and v_k is the identity of the validator that called the corresponding $r\_bcast_k(m, q)$. The reliable broadcast abstraction guarantees the following properties:

Definition 2 (Reliable Broadcast Properties)**.** • ***Agreement****:* *If an honest validator v_i outputs $r\_deliver_i(m, q, v_k)$, then every other honest validator v_j eventually outputs $r\_deliver_j(m, q, v_k)$.*

- ***Integrity****: For each sequence number $q \in \mathbb{N}$ and validator v_k, an honest validator v_i outputs $r\_deliver_i(m, q, v_k)$ at most once regardless of m.*
- ***Validity****: If an honest validator v_k calls $r\_bcast_k(m, q)$, then every honest validator v_i eventually outputs $r\_deliver_i(m, q, v_k)$.*

Definition 3 (Byzantine Atomic Broadcast)**.** *Additionally, for Byzantine Atomic Broadcast, each honest validator v_i can call $a\_bcast_i(m, q)$ and output $a\_deliver_i(m, q, v_k)$. A BAB protocol satisfies reliable broadcast (agreement, integrity, and validity) as well as:*

- ***Total Order****: If an honest validator v_i outputs $a\_deliver_i(m, q, v_k)$ before $a\_deliver_i(m', q', v'_k)$, then no honest party v_j outputs $a\_deliver_j(m', q', v'_k)$ before $a\_deliver_j(m, q, v_k)$.*

Finally, most prior work on consensusless transactions defines properties as if the protocol runs in a single epoch. This setting is unrealistic as it cannot accommodate recovering from equivocation, which is a common benign event for non-expert users. To this end, we extend all the protocols to also take as a parameter the epoch number and all properties should hold within a single epoch. Fortunately, the definition of reliable broadcast allows the recovery of liveness for blocked sequence numbers that are equivocated inside an epoch.

Definition 4 (Equivocation Tolerance)**.** *If a validator v_k concurrently called $r\_bcast_k(m, q, e)$ and $r\_bcast_k(m', q, e)$ with $m \neq m'$ then the rest of the validators either $r\_deliver_i(m, q, v_k, e)$, or $r\_deliver_i(m', q, v_k, e)$, or there is a subsequent epoch $e' > e$ where v_k is honest, calls $r\_bcast_k(m'', q, e')$ and all honest validators $r\_deliver_i(m'', q, v_k, e')$.*

2.2 Intuition Behind the NEURON Design

NEURON aims to push the latency boundaries of state machine replication in DAG-based blockchains. Achieving BFT consensus typically necessitates at least three message delays [14]. This underscores the inherent latency sub-optimality of Narwhal [2], that implements consensus (at least 3 message delays) on certified DAG blocks, when the block certification itself adds a further 3 message delays. Consequently, the first design challenge for NEURON is to manage equivocation and ensure data availability [25], without relying on pre-certification of individual blocks.

Moreover, even if we overcome this initial challenge, committing only one block every three messages falls short of the performance potential inherent in DAG-based consensus, which thrives on processing $O(n)$ blocks per round, one per validator, to fully utilize network resources. Therefore, a key objective for NEURON is to maximize block commitments per round to align system tail latency closely with the three-message delay. However, achieving this presents a more formidable challenge. Unlike traditional methods that rely on the recursive and elegant commit rules found in DAG-based consensus protocols [1, 2, 3, 5, 6], our approach cannot afford to require sufficient distance between two potential candidate blocks on the DAG to prevent conflicting decisions among validators with divergent sub-DAG views. Implementing such protocols would require at least one gap round, raising the latency to a minimum of four delays.

NEURON is not just a consensus protocol but a class of protocols facilitating state machine replication. For now, we only focused on the consensus protocol NEURON-C, but Section 13 extends it to protocols for consensusless agreement with NEURON-FPC. The core contribution of NEURON-FPC to prior work is that it is co-designed with NEURON-C instead of being a separate path like in Sui [23]. This allows us to avoid the need for generating a majority-signed certificate per transaction, freeing a significant amount of network and CPU resources to be used for actual transactions instead of generating and verifying certificates [16, 17].

3 The NEURON DAG Structure

We present the structure of the NEURON DAG. Its main goal is to build an uncertified DAG protocol that provides the same guarantees as a certified DAG. The key insight is that certification can be achieved implicitly by observing patterns in the DAG structure, without requiring separate certification messages.

The NEURON protocols operate in a sequence of logical rounds. For every round, each honest validator proposes a unique signed block; Byzantine validators may attempt to equivocate by sending multiple distinct blocks to different parties or no block. During a round, validators receive transactions from users and blocks from other validators and use them as part of their proposed blocks. A block includes references to blocks from prior rounds, always starting from their most recent block, alongside fresh transactions not yet incorporated indirectly in preceding blocks. Once a block contains references to at least $2f + 1$ blocks from the previous round, the validator signs it and sends it to other validators.

Clients submit transactions to a validator, who subsequently incorporates them into their blocks. In the event that a transaction fails to become finalized within a specified time frame, the client selects an alternative validator for resubmission. This client retry logic is essential for ensuring that transactions eventually get included even if the initial validator is faulty or overloaded.

3.1 Block Structure and Validity

A block should include at a minimum (1) the author A of the block and their signature on the block contents, (2) a round number r, (3) a list of transactions, (4) at least $2f + 1$ distinct hashes of blocks from the previous round, along potentially others from all previous rounds, (5) VRF proof and output for proposer selection, (6) VDF proof for anti-grinding protection when applicable, (7) committee votes for finality, and (8) finality certificate if the block achieves finality. By convention, the first hash must be to the previous block of A. This ensures that each validator maintains a consistent chain of their own blocks, simplifying crash recovery and equivocation detection.

Definition 5 (Block Validity)**.** *We index each block by the triplet $B \equiv (A, r, h)$, comprised of the author A, the round r, and the hash h of the block contents. A block is valid if (1) the signature is valid and A is part of the validator set, (2) all hashes point to distinct valid blocks from previous rounds, the first block links to a block from A, and within the sequence of past blocks, there are $2f + 1$ blocks from the previous round $r - 1$, (3) the VRF proof is valid for the block author and the appropriate randomness seed, and (4) if the block claims to be a proposer block, the VDF proof is valid and the VRF output is the minimum among all received proofs.*

The block validity rules ensure that the DAG maintains certain structural properties that are essential for the safety of the protocol. In particular, requiring $2f+1$ references to the previous round ensures that any two blocks at round r share at least $f + 1$ common ancestors at round $r - 1$, at least one of which must be from an honest validator. This quorum intersection property is the foundation of the safety proofs.

3.2 Identifying DAG Patterns

Definition 6 (Support)**.** *We say that a block B'* supports *a past block $B \equiv (A, r, h)$ if, in the depth-first search performed starting at B' and recursively following all blocks in the sequence of blocks hashed, block B is the first block encountered for validator A at round r.*

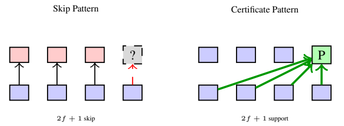

Figure 3: Illustration of skip pattern (left) and certificate pattern (right). In the skip pattern, $2f + 1$ blocks do not support the missing proposer. In the certificate pattern, $2f + 1$ blocks support the proposer block P.

The support relation captures the notion that B' has "seen" and acknowledged block B. Importantly, B' can support at most one block per validator per round, which prevents honest validators from inadvertently supporting equivocating blocks. As Figure 2 illustrates, a block $(A_3, r + 2, \cdot)$ may reference blocks $(A_2, r + 1, \cdot)$ and $(A_3, r + 1, \cdot)$ from different validators that respectively support block (A_3, r, L_r) and the equivocating block (A_3, r, L'_r). At most one of these equivocating blocks can gather support from $2f + 1$ validators.

NEURON-C and NEURON-FPC operate by interpreting the structure of the DAG to reach decisions using a single type of message, the block. They mainly operate by identifying the following two patterns:

Definition 7 (Skip Pattern)**.** *The skip pattern occurs where at least $2f + 1$ blocks at round $r + 1$ do not support a block (A, r, h). Note that there may be multiple or no proposal for the slot. The skip pattern is identified if for all proposals, we observe $2f + 1$ subsequent blocks that do not support it (or support no proposal).*

The skip pattern indicates that a proposer slot can be safely skipped because no block for that slot can ever achieve certification. This is critical for handling crash faults: if a validator crashes and fails to propose, the skip pattern allows the protocol to continue without waiting indefinitely.

Definition 8 (Certificate Pattern)**.** *The certificate pattern occurs where at least $2f + 1$ blocks at round $r + 1$ support a block $B \equiv (A, r, h)$. We then say that B is* certified*. Any subsequent block that contains in its history such a pattern is called a* certificate *for the block B.*

Using these patterns, we obtain certificates implicitly by interpreting the DAG, and the certification guarantees are identical to Narwhal [2]. That is, a certified block ($2f + 1$ support) is available and no other certified block may exist for the same spot (A, r). This counter-intuitively means that even if A equivocates and one of its blocks is certified, we process it as being correct—despite the self-evident Byzantine behavior. This does not constitute a problem as we only commit blocks that belong to the implicitly certified part of the DAG. We also note that a skip pattern guarantees that a certificate will never exist for a block, and thus it will never be part of the implicitly certified DAG and can be safely skipped.

Algorithm 1 Kahn's Algorithm for Block Ordering

```
1:  procedure OrderBlocks(committedBlocks)
2:      inDegree ← {}
3:      queue ← []
4:      result ← []
5:      for $b \in$ committedBlocks do
6:          inDegree[$b$] ← $|\{p \in b$.parents : $p \in$ committedBlocks$\}|$
7:          if inDegree[$b$] $= 0$ then
8:              queue.append($b$)
9:          end if
10:     end for
11:     Sort queue by ($b$.round, $b$.author, $H(b)$)
12:     while |queue| $> 0$ do
13:         $b$ ← queue.popFirst()
14:         result.append($b$)
15:         for $c \in$ children($b$) do
16:             inDegree[$c$] ← inDegree[$c$] $- 1$
17:             if inDegree[$c$] $= 0$ then
18:                 Insert $c$ into queue (sorted)
19:             end if
20:         end for
21:     end while
22:     return result
23: end procedure
```

Algorithm 2 VRF Winner Selection

```
procedure SelectWinner(submissions)
    quorum ← $\lfloor 2n/3 \rfloor + 1$
    if |submissions| < quorum then
        return ⊥                                  ▷ Wait for quorum
    end if
    winner ← ⊥
    lowestVRF ← MAX_UINT64
    for $s \in$ submissions do
        if VerifyVRF($s.pk$, $s.vrfOutput$, $s.vrfProof$) then
            if $s$.vrfOutput < lowestVRF then
                lowestVRF ← $s$.vrfOutput
                winner ← $s$.validator
            end if
        end if
    end for
    return winner
end procedure
```

3.3 Block Ordering with Kahn's Algorithm

Once blocks are committed through the consensus protocol, they must be linearized into a total order for state machine replication. Neuron uses a deterministic variant of Kahn's topological sort algorithm to order blocks within each committed wave. The algorithm processes the DAG in rounds, outputting blocks in a consistent order across all validators.

The key property of Algorithm 1 is that it produces a deterministic output for any given input DAG. The tie-breaking rule (line 11) ensures that when multiple blocks have zero in-degree, they are processed in a consistent order based on round, author, and hash. This determinism is essential for ensuring that all honest validators produce the same transaction ordering.

3.4 Liveness Through Timeouts

Since we are not using randomization, we need to rely on timeouts for liveness. Although every block has the potential of being committed directly in 3 message delays we cannot provide liveness for all of them through timeouts, as this would allow Byzantine validators to slow down the DAG to the point that every round would move at the speed of the timeout instead of network speed.

Instead we only provide guaranteed liveness after GST for one block per round. We deem this block as the *primary block* of the round r and require that validators at $r+1$ wait a timeout for it to arrive before disseminating their blocks. Additionally, if the block is in the view of a validator at $r+1$ we further require the validator to wait another timeout for $r+2$ or until there are $2f+1$ votes for the primary block of r. This guarantees the existence of a certificate over an honest primary block after GST and provides liveness for Neuron-C.

4 VRF Lottery: Lowest Output Wins

Neuron uses VRF for **winner selection**—the validator with the **lowest VRF output** wins the lottery. The 2/3 quorum requirement ensures fair participation before selection.

4.1 Cryptographic Construction

Neuron uses Ed25519-based VRF with SHAKE256 hash and Elligator hash-to-curve. For each slot, validators compute:

$$\begin{aligned} H &= \text{HashToCurve}(\text{chainID}\|\text{slot}\|\text{prevHash}) && (1)\\ \Gamma &= sk \cdot H && (2)\\ \text{output} &= \text{SHAKE256}(\text{Encode}(\Gamma)) && (3)\\ \pi &= (H, \Gamma, c, s) && (4) \end{aligned}$$

where sk is the validator's secret key, Γ is the VRF output point, and π is the Schnorr-style proof.

4.2 Winner Selection: Lowest VRF Output

Once 2/3 quorum is reached, the winner is simply the validator with the lowest VRF output:

4.3 Why Lowest VRF Wins

The VRF output is cryptographically random and unpredictable:

Table 1: 2/3 Quorum Security Properties

Property	Guarantee
Anti-rushing	Minority cannot finalize early
Fair competition	All honest validators can participate
BFT alignment	Threshold matches $2f + 1$ requirement

- No validator can predict their output before computing
- No validator can influence their output (deterministic from key + input)
- Verification is instant—anyone can check the winner
- Stake-weighting optional: score $=$ vrfOutput$/$stake

5 VDF for Anti-Grinding Protection

NEURON uses Wesolowski's VDF [27] as a **rate-limiter to prevent grinding attacks**. The VDF does NOT determine the winner—that is determined by lowest VRF output. The VDF simply ensures each VRF attempt requires sequential computation time, making grinding impractical.

5.1 The Grinding Attack Problem

Without VDF protection, an attacker could:

1. Try input$_1 \to$ VRF = 0x9999... (bad, discard)
2. Try input$_2 \to$ VRF = 0x8888... (bad, discard)
3. Try millions of inputs instantly...
4. Find input$_k \to$ VRF = 0x0001... (jackpot!)

By pre-computing millions of VRF outputs, attackers could always find a winning (lowest) value.

5.2 VDF as Rate-Limiter

With VDF protection, each VRF computation requires a VDF computation first:

$$\text{VRF\_input} = \text{VDF\_output}(\text{prevBlockHash}\|\text{slot}\|\text{nonce})$$

Since VDF takes $\sim$50ms of sequential (non-parallelizable) computation:

- Attacker can only try $\sim$20 inputs per second
- Grinding millions of inputs would take years
- Fair lottery based on actual VRF randomness

5.3 VDF Construction

Definition 9 (Verifiable Delay Function)**.** *A VDF is a function $f : X \to Y$ such that: (1) f requires T sequential operations to compute, (2) f cannot be significantly parallelized, (3) verification requires only $O(\log T)$ operations.*

Table 2: VDF Parameters

Parameter	Value	Description
Group type	Class group	Unknown order group
Discriminant	1024 bits	Security parameter
Difficulty T	$\sim$10,000,000	Sequential squarings
Compute time	$\sim$50 ms	Non-parallelizable
Verify time	$\sim$0.1 ms	$\sim$400× faster
Proof size	516 bytes	y (258B) + π (258B)

Table 3: VRF vs VDF: Distinct Roles in NEURON

Component	Purpose	Winner Selection?
VRF Output	Unpredictable randomness	**YES** - lowest wins
VRF Proof	Proves eligibility	No
VDF Output	Input for VRF	No
VDF Proof	Proves computation done	No
2/3 Quorum	Prevents rushing	No

The Wesolowski VDF operates in a class group of unknown order:

$$g = \text{GeneratorFromSeed}(H(\text{prevBlockHash}\|\text{slot})) \tag{5}$$

$$y = g^{2^T} \mod N \tag{6}$$

$$l = H_{\text{prime}}(g\|y), \quad \pi = g^{2^T/l} \mod N \tag{7}$$

Verification checks $\pi^l \cdot g^r \equiv y \pmod N$ in $O(\log T)$ time ($\sim$400× faster than computation).

5.4 Security Analysis

Property 1 (Grinding Resistance)**.** *An attacker with k times more computational power can only try k times more VRF inputs per unit time. With VDF requiring 50ms per attempt, even 1000× computational advantage yields only 20,000 attempts/second—insufficient to bias the lottery.*

Property 2 (Fairness)**.** *Winner selection depends solely on VRF output (cryptographically random). VDF computation time does not affect who wins—it only ensures honest participation by rate-limiting attempts.*

6 The NEURON-C Consensus Protocol

NEURON-C is the first DAG-based consensus protocol that decides blocks in 3 message delays. It achieves this through foregoing an explicit certification of the blocks and through treating every block as a first-class block that can be proposed and decided directly. Additionally, NEURON-C is able to instantly identify and exclude crashed validators, the most frequent failure case in blockchains in the wild.

6.1 Proposer Slots

NEURON-C introduces the concept of *proposer slot*. A proposer slot represents a tuple (validator, round) and can be either empty or contain the validator's proposal for the respective round. For instance, in Bullshark [3], there is a single proposer every two rounds, which results in higher latencies. Unfortunately, it is not trivial to increase the number of slots, as the commit rule of Bullshark relies on the fact that every proposer slot has a link to every other proposer slot, something that is not possible even if there is a single proposer per round, let alone n.

We overcome this challenge by introducing multiple states for each proposer slot:

- **to-commit**: The proposer block for this slot will be committed. This is the equivalent of the decided state in prior work.
- **to-skip**: The proposer slot will be skipped (no block committed for this slot). This allows excluding crashed validators.
- **undecided**: The slot's status cannot yet be determined. This forces subsequent slots to wait, preventing non-deterministic commits.

The number of proposer slots instantiated per round can be configured but for systems with few faults it can be set to n so that every block has a chance to commit in 3 steps. Initially, we establish a deterministic total order among all pending proposer slots, aligning with the round ordering. Within a single round, the ordering may either remain fixed or change per round (e.g., round robin).

6.2 The NEURON-C Decision Rule

This section describes the decision rule of NEURON-C. The end goal is to mark all proposer slots as either to-commit or to-skip by detecting the DAG patterns presented in Section 3.2.

Step 1: Direct Decision Rule. Starting with the latest proposer slot, the validator applies the following direct decision rule:

- Mark as **to-commit** if $2f + 1$ distinct implicit certificate blocks are observed (certificate pattern with $2f + 1$ certificates).
- Mark as **to-skip** if a skip pattern is observed: for any proposal for the slot, $2f + 1$ blocks do not support it.
- Otherwise, remain **undecided** and proceed to indirect decision.

The first message delay is the proposal block; the second message delay is the supporting/voting blocks; and the third message delay is the certifying blocks. This achieves the optimal 3-message-delay latency.

Promptly marking slots as to-skip is a key design point that reduces undecided slots following crash-failures and allows NEURON-C to tolerate crash-faults virtually for free.

Step 2: Indirect Decision Rule. If the direct decision rule fails to determine the slot, the validator resorts to the indirect decision rule. This rule operates in two stages:

1. Search for an **anchor**: the first slot with round number $r' > r + 2$ that is already marked as either undecided or to-commit.
2. If the anchor is marked as **undecided**, mark the current slot as undecided.
3. If the anchor is marked as **to-commit**:
    - Mark as **to-commit** if the anchor causally references a certificate pattern over the slot.
    - Mark as **to-skip** if there is no certificate pattern linking the anchor to the slot.

Algorithm 3 Helper Functions

```
procedure GetProposerBlock($w$)
    $r_{\text{proposer}} \leftarrow$ ProposerRound($w$)
    id $\leftarrow$ GetPredefinedProposer($r_{\text{proposer}}$)
    if $\exists b \in \text{DAG}[r_{\text{proposer}}]$ s.t. $b$.author = id then
        return $b$
    end if
    return $\bot$
end procedure

procedure Link($b_{\text{old}}, b_{\text{new}}$)
    return $\exists$ sequence $b_1, \ldots, b_k$ s.t. $b_1 = b_{\text{old}}$,
        $b_k = b_{\text{new}}$ and $\forall j \in [2, k] : b_{j-1} \in b_j$.parents
end procedure

procedure IsVote($b_{\text{vote}}, b_{\text{proposer}}$)
    function SupportedBlock($b$, id, $r$)
        if $r \geq b$.round then return $\bot$
        end if
        for $b' \in b$.parents do
            if ($b'$.author, $b'$.round) = (id, $r$) then
                return $b'$
            end if
            res $\leftarrow$ SupportedBlock($b'$, id, $r$)
            if res $\neq \bot$ then return res
            end if
        end for
        return $\bot$
    end function
    (id, $r$) $\leftarrow$ ($b_{\text{proposer}}$.author, $b_{\text{proposer}}$.round)
    return SupportedBlock($b_{\text{vote}}$, id, $r$) = $b_{\text{proposer}}$
end procedure

procedure IsCert($b_{\text{cert}}, b_{\text{proposer}}$)
    cnt $\leftarrow |\{b \in b_{\text{cert}}.\text{parents} : \text{IsVote}(b, b_{\text{proposer}})\}|$
    return cnt $\geq 2f + 1$
end procedure
```

This design point contributes to safety without requiring direct links between proposers. Instead of forcing a happened-before relationship between proposer slots, we leverage the predefined total ordering to ensure that any decision is recursively carried forward deterministically.

Step 3: Commit Sequence. After processing all slots, the validator derives an ordered sequence of slots. It then iterates over that sequence, committing all slots marked as to-commit and skipping all slots marked as to-skip. This iteration continues until the first undecided slot is encountered.

This final design point of NEURON-C applies backpressure through undecided slots to preserve safety. However, this does not harm performance, as these undecided slots would not have existed as commit candidates in prior designs.

Algorithm 4 Direct Decider Algorithm

```
waveLength ← 3                                   ▷ Configurable
roundOffset, proposerOffset

procedure TryDirectDecide($w$)
    if SkippedProposer($w$) then
        return Skip($w$)
    end if
    $b_{\text{proposer}}$ ← SupportedProposer($w$)
    if $b_{\text{proposer}} \neq \perp$ then
        return Commit($b_{\text{proposer}}$)
    end if
    return $\perp$
end procedure

procedure WaveNumber($r$)
    return $(r -$ roundOffset$)/$waveLength
end procedure

procedure SkippedProposer($w$)
    $r_{\text{proposer}}$ ← ProposerRound($w$)
    id ← GetPredefinedProposer($r_{\text{proposer}}$)
    $B$ ← GetFirstVotingBlocks($w$)
    cnt ← $|\{b \in B : \forall b' \in b.$parents $: b'.$author $\neq$ id$\}|$
    return cnt $\geq 2f + 1$
end procedure

procedure SupportedProposer($w$)
    $b_{\text{proposer}}$ ← GetProposerBlock($w$)
    $B$ ← GetDecisionBlocks($w$)
    if $|\{b' \in B :$ IsCert$(b', b_{\text{proposer}})\}| \geq 2f + 1$ then
        return $b_{\text{proposer}}$
    end if
    return $\perp$
end procedure
```

6.3 Choosing the Number of Proposer Slots

The example presented by the algorithms assumes a number of proposer slots per round that can be configured. While setting this to n (committee size) offers the best latency under normal conditions, it may impact performance during periods of extreme asynchrony or under Byzantine attack.

In these cases, the probability that the direct decision rule fails to classify a proposer slot increases when some proposer slots are slow or equivocate. This forces the validator to resort to the indirect decision rule more often. As a result, there can be an increase in the number of undecided slots, which in turn delays the commit sequence.

This is nevertheless an extreme case of the adversary controlling the network and some validators only to slow down the system without any actual profit. After a decade of running blockchains in the wild, this is not something that has been witnessed, as attackers tend to attack in order to break safety and not liveness.

Nevertheless, in order to mitigate it we use HammerHead [36] to select $2f + 1$ leaders that are best performing as candidate leaders. This strikes a good balance as it does not increase the median latency and only increases the expected latency by $1/3$ of a delay.

7 GHOST Fork Choice Rule

Neuron uses the GHOST (Greedy Heaviest-Observed Subtree) fork choice rule [26] to select the canonical chain in the presence of forks. GHOST was originally proposed for proof-of-work blockchains but has been adapted for BFT contexts in protocols like Ethereum's Casper FFG.

Definition 10 (GHOST Weight). *For a block B, its weight is defined recursively as:*

$$W(B) = 1 + \sum_{c \in \mathit{children}(B)} W(c)$$

At each fork, GHOST selects the child with the highest weight, effectively counting all descendants rather than just the longest chain. This provides better security against selfish mining attacks and ensures that all honestly produced blocks contribute to chain selection.

Tie-breaking: When two branches have equal weight, the block with the lower hash value wins, ensuring deterministic selection. This tie-breaking rule is essential for ensuring all validators converge on the same chain.

8 Proposal-First Consensus: Born-Finalized Blocks

A critical innovation in Neuron is the **Proposal-First Consensus** model where blocks are created WITH finality certificates attached. Unlike traditional consensus where blocks are created first and then voted on, this approach ensures blocks are "born finalized"—achieving instant confirmation with zero separate finality delay.

8.1 Traditional vs. Proposal-First Consensus

In traditional BFT consensus, the flow is: (1) Proposer creates block, (2) Block broadcast to validators, (3) Validators vote on the block, (4) Proposer collects $2f + 1$ votes, (5) Block becomes finalized. This introduces a **confirmation delay** between block creation and finalization.

In Neuron's Proposal-First model: (1) Proposer creates **DAGBlockProposal** (not a block), (2) Proposal broadcast to validators, (3) Validators vote on the **proposal**, (4) When $2f+1$ votes collected, block is created **WITH certificate**, (5) Block is **instantly finalized** at creation.

8.2 Proposal Structure

A `DAGBlockProposal` contains all block fields except the finality certificate:

Table 4: Traditional vs. Proposal-First Consensus

Aspect	Traditional	Proposal-First
Vote target	Block	Proposal
Block creation	Before votes	After $2f + 1$ votes
Certificate	After finalization	At block creation
Finality timing	After vote collection	Instant
Confirmation delay	100-500ms	0ms
Proposer self-vote	Yes	**No** (others only)

Table 5: Failure Handling: Traditional vs. Proposal-First

Scenario	Traditional	Proposal-First
Validator offline	Block orphaned	No block exists
Voting timeout	Failed block cleanup	Nothing to clean
Network partition	Forked blocks	No block created
Proposer crash	Orphan + TX recovery	TXs safe in mempool

Definition 11 (DAG Block Proposal)**.** *A proposal P contains: (1) pivot hash (GHOST-selected parent), (2) tip hashes (other parents), (3) level = finalized_level + 1, (4) transactions from mempool, (5) VRF proof and output, (6) VDF proof, (7) proposer signature. Notably, P does* ***not*** *contain committee votes or certificate.*

8.3 Seven-Phase Consensus Flow

Phase 1: VRF Lottery + VDF Anti-Grinding. All validators compute VDF (rate-limiting), then VRF proofs. Once $2f + 1$ submissions received, validator with **lowest VRF output** wins.

Phase 2: Proposal Creation. Winner creates `DAGBlockProposal` with pivot (GHOST), tips, transactions, and proofs. No certificate attached.

Phase 3: Proposal Broadcast. Winner broadcasts proposal via P2P.

Phase 4: Validator Voting. Each validator (except proposer): validates proposal, checks proofs, signs vote $\sigma = \text{Sign}(sk, H(P) \| \text{addr})$, sends to proposer.

Phase 5: Vote Collection. Proposer collects until: power $\geq \lfloor 2 \cdot \text{totalPower}/3 \rfloor + 1$

Phase 6: Block Creation with Certificate. Proposer creates finalized block: $\text{DAGBlock} = \text{Proposal} \cup \{\text{votes}, \text{certificate}\}$

Phase 7: Finalized Block Broadcast. Block (with certificate) broadcast. Validators immediately accept as finalized.

8.4 Why Proposer Cannot Self-Vote

Property 3 (No Self-Voting)**.** *The $2f+1$ votes must come from validators* ***other than*** *the proposer.*

Rationale: (1) $2f + 1$ consensus means others must agree, (2) Self-voting lets one node boost own blocks, (3) Enforces true distributed consensus, (4) Prevents threshold gaming.

8.5 Instant Finality Theorem

Theorem 1 (Instant Finality)**.** *In Proposal-First consensus, block creation and finalization occur atomically. There is no confirmation delay after block creation.*

Proof. Block B is created only when $2f + 1$ votes on proposal P are collected. At creation, B contains certificate C with these votes. Any validator receiving B verifies C and immediately accepts B as finalized. No additional voting required. □

8.6 No Failed States: A Key Advantage

A critical advantage of Proposal-First consensus is the **complete absence of failure states**. Unlike traditional consensus where failed voting leaves orphaned blocks requiring complex cleanup and transaction recovery, Proposal-First has no intermediate failure states.

Property 4 (No Orphaned Blocks)**.** *In Proposal-First consensus, orphaned blocks cannot exist. A block is only created when $2f + 1$ votes are collected, meaning every block that exists is already finalized.*

Why this works:

$$\text{Block} = \text{Proposal} + \frac{2}{3}\ \text{Votes} + \text{Certificate}$$

If **any** component is missing, the block simply does not exist. There is nothing to clean up, no orphans to handle, and no transactions to recover. Transactions remain safely in the mempool until successfully included in a finalized block.

Retry flow on voting failure:

1. Slot N: Winner A creates proposal, voting times out (only $1/3$ votes)
2. **No block created**—not a “failed block,” it simply never existed
3. Transactions remain safely in mempool
4. Slot $N + 1$: New lottery selects Winner B
5. B creates proposal including same transactions from mempool
6. $2/3$ votes received → Block created WITH certificate
7. Transactions confirmed in born-finalized block

This design eliminates entire categories of edge cases: no orphan management, no transaction replay logic, no fork resolution for uncommitted blocks, no complex recovery procedures.

9 Light Node TX Validation

NEURON implements **proposal-level transaction validation** using Reed-Solomon erasure coding proofs. This ensures transactions are validated BEFORE inclusion in proposals, preventing invalid transactions from wasting block space.

Table 6: TX Validation Benefits

Benefit	Description
No wasted gas	Invalid TXs filtered before inclusion
Faster execution	No failed TX processing
Light client friendly	Proofs enable verification without full state
Data availability	Reed-Solomon ensures TX data is available

Table 7: Collection Time Comparison

Blockchain	Block Time	Collection Time
QoraNet/NEURON	∼100ms	100ms
Solana	∼400ms	400ms slot
Ethereum	∼12s	Full slot
Bitcoin	∼10min	N/A (PoW)

9.1 The Problem: Wasted Block Space

In traditional blockchains, invalid transactions (wrong nonce, insufficient balance) are included in blocks and only fail during execution:

- TX included in block → Executed → `gasUsed=0` (wasted space)
- Block space consumed by transactions that cannot succeed
- Light clients cannot verify validity without full state

9.2 Solution: Validate at Proposal Time

In NEURON, each transaction is validated against current state **before** inclusion in the proposal:

9.3 Reed-Solomon Validation Proofs

Each validated transaction includes a cryptographic proof:

Definition 12 (TX Validation Proof)**.** *A validation proof π_{tx} contains: (1) transaction hash, (2) account nonce at validation time, (3) account balance at validation time, (4) state root, (5) Reed-Solomon Merkle proof, (6) data availability commitment.*

At finalization, validators verify that all proofs still match current state:

$$\forall tx \in \text{block} : \pi_{tx}.\text{nonce} = \text{currentNonce}(tx.\text{sender})$$

If state changed between proposal and finalization (e.g., concurrent transaction), the proof fails and the block is rejected.

9.4 Benefits

10 Proposal TX Tracking

In a DAG with parallel proposals, the same transaction could be included in multiple proposals at the same level. NEURON implements **TX tracking** to prevent duplicates.

10.1 The Duplicate TX Problem

```
Without Tracking:
  Validator A wins → Proposal with TX1
  Validator B wins → Proposal with TX1 (duplicate)
  Both finalize → TX1 executes in A, fails in B
```

10.2 Solution: Track Pending Proposals

NEURON tracks which transactions are in pending (not yet finalized) proposals:

Definition 13 (TX Tracking vs Marking)**.** ***Tracking**: TX is in pending proposal (not yet finalized)—prevents duplicate inclusion. **Marking**: TX is in finalized block—removes from mempool.*

10.3 Stale Tracking Cleanup

Tracked TXs older than 2 seconds are automatically cleaned up, allowing them to be included in future proposals if the original proposal failed silently.

11 Collection Time: TX Propagation Window

After winning the VRF lottery, the proposer waits a **collection time** (100ms) before creating the proposal. This allows:

1. **TX Propagation**: Time for new transactions to spread across network
2. **Tracking Propagation**: Time for peer proposal tracking info to arrive
3. **Reduced Duplicates**: More validators see consistent TX state

The 100ms collection time works in NEURON due to: (1) DAG parallel structure, (2) VRF lottery (no mining race), (3) TX tracking prevents duplicates.

12 Committee Finality with Certificates

NEURON uses a 22-validator committee with 77% (17/22) finality threshold for the voting process described above.

12.1 Committee Configuration

The committee is selected based on stake weight and recent performance. Size of 22 balances security with efficiency.

The 77% threshold (17/22) exceeds the standard $2/3 + 1$ threshold (15/22) to provide additional safety margin. This ensures that even if a few additional committee members become unavailable, the protocol can still make progress.

Table 8: Committee Parameters

Parameter	Value
Committee size	22 validators
Total validators	100
Finality threshold	17/22 (77%)
Byzantine tolerance	7/22 (31.8%)
Epoch length	$\sim$500 blocks
Rotation rate	5 validators/epoch

12.2 Finality Certificate Structure

When a block receives votes from 17 of 22 committee members, a finality certificate is created.

Definition 14 (Finality Certificate)**.** *A finality certificate for block B at level N consists of:*

- ***BlockHash**: $H(B)$—the hash of the finalized block*
- ***Level**: N—the finality level (height)*
- ***Votes**: $[\sigma_1, \sigma_2, \ldots, \sigma_{17}]$—signatures from committee members*
- ***TotalPower**: Sum of voting power of signers*
- ***Threshold**: Required voting power (77%)*
- ***CommitteeRoot**: Merkle root of committee public keys*

State Synchronization: Late-joining validators can verify finality by: (1) downloading the finality certificate, (2) verifying the committee root against a known checkpoint, (3) verifying each signature against the committee public keys, and (4) confirming that the total voting power exceeds the threshold. This requires $O(n)$ signature verifications where n is the committee size, compared to replaying the entire voting history which could require $O(m \cdot n)$ verifications for m blocks.

13 The NEURON-FPC Fast Path Protocol

For workloads necessitating consensus, the NEURON-C protocol successfully achieves a low latency bound. However, popular workloads [28] such as asset transfers, payments, or NFT minting can be finalized before consensus through an even lower latency fast path. This section presents NEURON-FPC that extends the consensus protocol with such consensusless transactions.

13.1 Embedding a Fast Path into the DAG

The real-world deployment of hybrid blockchains, exemplified by Sui [23], capitalizes on the insight that certain objects, like coins, solely access state controlled by a single party and need not undergo consensus. These objects can be finalized through a fast path utilizing reliable broadcast. Such objects are classified as having an **owned object type** as opposed to the traditional **shared object type**. Transactions that exclusively involve owned objects as inputs are called **fast path transactions**. Two transactions conflict if they take as input the same owned object.

In NEURON-FPC, validators include transactions and explicitly vote for causally past transactions in their blocks. A validator includes a transaction T in its block if it does not conflict with any other transaction for which the validator has previously voted. This is also an implicit vote for the transaction. Other validators include explicit votes for T in a block B if: (i) T is present in the causal history of B; and (ii) T does not conflict with any other already voted on transaction.

Once T has $2f + 1$ votes from distinct validators, we call T **certified**. It is a guarantee that no two conflicting transactions will be certified in the same epoch. This is the basis of the fast path safety. Transaction T is **finalized** when either (i) there exists $2f + 1$ validators supporting a certificate over T, even before a NEURON-C commit, or (ii) NEURON-C commits through consensus a block that contains a certificate over T in its causal history.

In contrast to previous approaches [23, 20, 22], the fast path in NEURON-FPC is integrated within the DAG structure itself. This eliminates the need for additional protocol messages and for validators to individually sign each fast-path transaction. Instead, a validator's fast path votes are embedded within its signed blocks, which are already produced as part of the consensus protocol. Consequently, in addition to the block contents of NEURON-C, blocks in NEURON-FPC also incorporate explicit votes for transactions involving at least one owned object input.

13.2 Epoch Change and Reconfiguration

Quorum-based blockchains typically operate in epochs, allowing validators to join and leave the system at epoch boundaries. Moreover, epoch boundaries serve as natural boundaries for protocols with a consensusless path to "unlock" transactions that have lost liveness due to equivocation from the client [23, 37].

The NEURON-FPC epoch-change protocol ensures safety through an **epoch-change bit** in all blocks. When this bit is set to 1 (default 0), blocks referencing these votes do not contribute to finalization of fast path transactions. This effectively pauses the fast path near the end of the epoch, mitigating race conditions between fast path finalization and epoch close.

Theorem 2 (Epoch Close Safety)**.** *Transactions finalized by* NEURON-FPC *in an epoch continue to persist in all subsequent epochs.*

Proof. For the epoch to close, blocks from $2f + 1$ validators with the epoch-change bit set must be committed. Since a finalized transaction has $2f + 1$ validators publishing certifying blocks, by quorum intersection, one honest validator must have published both a certifying block (in an earlier round) and an epoch-change block. The certifying block is therefore in the causal history of the epoch-change commit and persists. □

14 Security Analysis

This section presents the formal security analysis of NEURON-C, proving safety, integrity, and liveness under the Byzantine assumption presented in Section 2.

A validator v_k broadcasts messages calling $a\_bcast_k(b, r)$, where b is a block signed by validator v_k and r is the block's round number. Every validator v_i has an output $a\_deliver_i(b, b.\text{round}, v_k)$, where v_k is the author of b and the validator that called the corresponding $a\_bcast_k(b, b.\text{round})$.

Lemma 3. *If at a round x, $2f + 1$ blocks from distinct authorities certify a block B, then all blocks at future rounds ($> x$) will link to a certificate for B from round x.*

Proof. Each block links to $2f + 1$ blocks from the previous round. For the sake of contradiction, assume that a block in round $r(> x)$ does not link to a certificate from round x. If $r = x + 1$, by the standard quorum intersection argument, a correct validator equivocated in round x, which is a contradiction. Similarly, if $r > x + 1$, by the standard quorum intersection argument, a correct validator's block in round $r - 1$ does not link to its own block in round x, which is also a contradiction. □

Lemma 4. *If a correct validator commits some block in a slot s, then no correct validator decides to directly skip the slot s.*

Proof. A validator X decides to directly skip a slot s if there is no support during the support rounds for any block corresponding to s. If another validator committed some block b for slot s, at least $f + 1$ correct validators supported b. By the quorum intersection argument, X must have observed at least one validator supporting b, which is a contradiction. □

Lemma 5. *If a correct validator directly commits some block in a slot s, then no correct validator decides to skip s.*

Proof. For the sake of contradiction, assume that a correct validator X directly commits block b in slot s while another correct validator Y decides to skip the slot. Y can decide to skip in one of two ways: (a) Y directly skipped s because there was no support for any block corresponding to s, or (b) Y skipped s during the recursive commits triggered by a direct commit of a later slot.

Case (a): Direct contradiction of Lemma 4.

Case (b): Let block b' denote the proposer block, committed during the recursive indirect commits, that allowed Y to decide s as skipped. Due to the commit rule, the round number of b' is greater than the decision round of s, and b' does not link to a certificate for b. Since X committed b, there are $2f + 1$ certificates for b in its decision round, leading to a contradiction due to Lemma 3. □

Lemma 6. *For any slot $s \equiv (v, r)$, a correct validator never supports two distinct block proposals from validator v in round r across all of its blocks.*

Proof. By definition, a block can only support at most a single proposal for a particular slot s. Block support is calculated through a depth-first traversal of the referenced blocks, such that the first block corresponding to s encountered during the traversal is supported. Since a correct validator first includes a reference to its own block from the previous round, once a correct validator supports a certain block for s, it continues to support the same block in all of its future blocks. □

Lemma 7. *For any slot, at most a single block will ever be certified, i.e., gather a quorum ($2f + 1$) of support.*

Proof. For contradiction's sake, assume that two distinct block proposals for a slot gather a quorum of support. By the standard quorum intersection argument, a correct validator supports two distinct blocks for the same slot, which is a contradiction of Lemma 6. □

Corollary 8. *No two correct validators commit distinct blocks for the same slot.*

Lemma 9. *All correct validators have a consistent state for each slot, i.e., if two validators have decided the state of a slot, then both either commit the same block or skip the slot.*

Proof. Let $[x_i]_{i=0}^{n}$ and $[y_i]_{i=0}^{m}$ denote the state of the slots for two correct validators X and Y, such that n and m are respectively the indices of the highest committed slot. WLOG $n \leq m$. Any slot decided by X higher than n are direct skips and are therefore consistent with Y due to Lemma 4. We prove by induction statement $P(i)$ for $0 \leq i \leq n$: if X and Y both decide the slot i, then both either commit the same block or skip the slot.

Base Case: $i = n$. X directly commits slot i. From Lemma 5, if Y decides slot i, then it must also commit. By Corollary 1, Y commits the same block.

Inductive Step: Assuming $P(i)$ is true for $k + 1 \leq i \leq n$, we prove $P(k)$. The analysis follows similar reasoning, using the induction hypothesis to show that the anchor block determining the indirect decision must be the same for both validators. □

Theorem 10 (Total Order)**.** NEURON-C *satisfies the total order property of Byzantine Atomic Broadcast.*

Proof. Correct validators deliver blocks by using an identical deterministic algorithm to order the causal history of committed proposer blocks. Since a correct validator has all the causal histories of a block when the block is added to its DAG, and the sequence of committed proposer blocks of one validator is a prefix of another's (by Lemma 9), all correct validators deliver a consistent sequence of blocks. □

Theorem 11 (Integrity)**.** NEURON-C *satisfies the integrity property of Byzantine Atomic Broadcast.*

Proof. The algorithm to linearize the causal history of a committed proposer block removes any block with duplicate sequence numbers before delivering the sequence of blocks. □

14.1 Liveness Analysis

Lemma 12 (Round-Synchronization)**.** *After GST all honest parties will enter the same round within Δ.*

Proof. After GST all messages sent before GST deliver within Δ. If r is the highest round any honest validator proposed a block for before GST, then every honest validator will receive the block proposal of the honest validator at GST + Δ and also enter r. □

Lemma 13 (Leader-Proposal)**.** *After GST an honest proposer's proposal will get votes from every honest validator.*

Proof. After GST if an honest validator enters wave w, then it has to broadcast the last block of wave $w-1$. Within Δ the honest proposer (and every other honest party) will receive the block and adopt the parents, being able to also enter wave w as they are all synchronized. Then the honest proposer will directly propose its block. Since the timeout is set to $2 \cdot \Delta$, the proposer's block of wave w will arrive before the first honest validator times out, hence every honest validator will vote for the proposer. □

Lemma 14 (Sufficient Votes)**.** *After GST all honest validators will create a certificate for the honest proposer.*

Proof. By the Leader-Proposal lemma, all honest validators will vote for an honest proposer after GST. For an honest validator to propose a block at the decision round it needs to (a) get the proposal of the proposer and (b) have $2f+1$ parents. Both conditions are met within $2 \cdot \Delta$ after GST. □

Theorem 15 (Consensus Liveness)**.** *After GST the proposal of an honest proposer will commit.*

Proof. By the Sufficient Votes lemma, there will be $2f+1$ certificates for the proposer, one per honest party. By the code, an honest validator tries to commit the proposer for every block received, so eventually it gets the $2f+1$ certificates. The validator schedules the block to be committed. All prior undecided blocks will eventually be decided (by the round-robin schedule ensuring honest proposers), and the validator will deliver the honest proposer's block. □

14.2 NEURON-FPC Security

Theorem 16 (NEURON-FPC Safety)**.** *An honest validator in* NEURON-FPC *never finalizes two conflicting transactions.*

Proof. Transactions that have an owned object as input require votes from $2f+1$ validators to be finalized. If two conflicting fast paths are finalized, an honest validator must have voted for both transactions (by quorum intersection), hence a contradiction. Using a similar argument, a fast path transaction does not conflict with a consensus path transaction. □

Theorem 17 (Fast-Path Liveness)**.** *An honest fast-path transaction will commit after GST, assuming the epoch does not end.*

Proof. An honest validator will submit a fast-path transaction that does not have equivocation. All honest validators will receive it after Δ and vote. These votes will appear in the DAG after at most $4 \cdot \Delta$ since any round has at most duration of timeout + $\Delta = 3 \cdot \Delta$. In the next round, every honest validator will reference the $2f+1$ votes in their DAG and execute. □

15 Implementation

We implement a networked multi-core NEURON validator in Rust. It uses `tokio` [29] for asynchronous networking, utilizing TCP sockets for communication without relying on any RPC frameworks. For cryptographic operations, we use `ed25519-consensus` [30] for asymmetric cryptography, `blake2` [31] for cryptographic hashing, and a custom VRF implementation based on the IETF ECVRF draft.

To ensure data persistence and crash recovery, we integrate a Write-Ahead Log (WAL), seamlessly tailored to our specific requirements. We have intentionally avoided key-value stores like RocksDB [32] to eliminate associated overhead and periodic compaction penalties. Our implementation optimizes I/O operations by employing vectored writes [33] for efficient multi-buffer writes in a single syscall. For reading the WAL, we make use of memory-mapped files while carefully minimizing redundant data copying and serialization. We use the `minibytes` crate to efficiently work with memory-mapped file buffers without unsafe code.

While all network communications in our implementation are asynchronous, the core consensus code runs synchronously in a dedicated thread. This approach facilitates rigorous testing, mitigates race conditions, and allows for targeted profiling of this critical code path.

In addition to regular unit tests, we have two supplementary testing utilities. First, we developed a simulation layer that replicates the functionality of the tokio runtime and TCP networking. This simulated network accurately simulates real-world WAN latencies, while our tokio runtime simulator employs a discrete event simulation approach to mimic the passage of time. Second, we created a command-line utility (called "orchestrator") designed to deploy real-world clusters of NEURON with machines distributed across the globe.

16 Evaluation

We evaluate the throughput and latency of NEURON through experiments on Amazon Web Services (AWS). We show its performance improvements over the state-of-the-art.

16.1 Experimental Setup

We deploy a NEURON testbed on AWS, using `m5d.8xlarge` instances across 13 different AWS regions: N. Virginia (us-east-1), Oregon (us-west-2), Canada (ca-central-1), Frankfurt (eu-central-1), Ireland (eu-west-1), London (eu-west-2), Paris (eu-west-3), Stockholm (eu-north-1), Mumbai (ap-south-1),

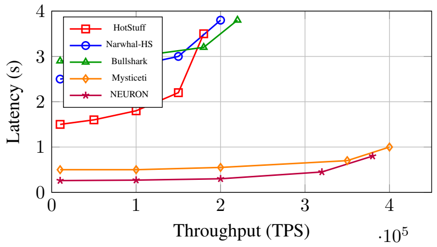

Figure 4: Throughput-Latency graph comparing Neuron-C with state-of-the-art consensus protocols on 50 validators.

Singapore (ap-southeast-1), Sydney (ap-southeast-2), Tokyo (ap-northeast-1), and Seoul (ap-northeast-2). Validators are distributed across those regions as equally as possible. Each machine provides 10Gbps of bandwidth, 32 virtual CPUs (16 physical cores) on a 2.5GHz Intel Xeon Platinum 8175, 128GB memory, and runs Linux Ubuntu server 22.04.

Neuron employs two proposer slots per round as a balance between performance and fault tolerance. To implement the partial synchrony assumption, validators wait up to 1 second to receive a proposal from the first proposer slot of the previous round.

Transactions in the benchmarks are arbitrary and contain 512 bytes. The ping latency between the validators varies from 50ms to 250ms. When referring to latency, we mean the time elapsed from when the client submits the transaction to when the transaction is committed by the validators.

16.2 Benchmark in Ideal Conditions

Figure 4 illustrates the Latency-Throughput relationship for Neuron-C compared with other consensus protocols for a deployment of 50 validators. At a steady state of 50k to 400k TPS, Neuron-C exhibits sub-second latency, a factor 2x-3x lower than the fastest protocols. Bullshark uses a certified DAG and worker architecture and is over 3x slower in terms of latency compared with Neuron-C for low system loads.

In terms of throughput, Neuron-C achieves over 300k-400k TPS before the latency reaches 1s, well lower than the latency of state-of-the-art systems. This illustrates that the single-host throughput efficiency of Neuron-C is higher than for previous designs. Note that current real-world blockchains combined process fewer than 100M transactions per day, equivalent to about 1.2k TPS, well within the steady state low-latency parameter space for Neuron-C.

Throughout these benchmarks, the CPU utilization of the validators remains below 10% and the validators consume less than 15GB of memory when experiencing the highest load of 400k tx/s.

16.3 Benchmark with Faults

Figure 5 illustrates the performance when a committee of 10 parties suffers 0 to 3 crash faults. HotStuff suffers massive degradation in both throughput and latency. With 3 faults, the latency of HotStuff exceeds 15s. Neuron-C can process the same load while maintaining sub-second latency, demonstrating a 15-20x latency improvement. This improvement is due to the ability of Neuron-C to operate with multiple leaders per round.

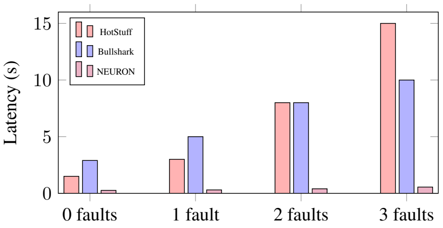

Figure 5: Latency under crash faults (10 validators, 0-3 faulty).

Table 9: Performance Comparison Summary

Protocol	TPS	Latency	Confirm	Rounds
PBFT	50k	800ms	200ms	3
HotStuff	50k	1.5s	300ms	4
Narwhal-HS	120k	2.5s	500ms	6
Bullshark	150k	2.9s	400ms	6
Mysticeti	350k	500ms	100ms	3
Neuron-C	**320k**	**260ms**	**0ms**	**3**

16.4 Fast Path Benchmark

For low loads, both Neuron-FPC and Zef [21] have comparable latency of around 0.25s. However, as the load increases, a Zef host has to verify and produce an increasing number of signatures, proportional to the throughput times the number of validators. As a result, throughput tops at 20k TPS for a small Zef network at a latency of 0.5s. Neuron-FPC avoids the need for individual signature verification for each transaction and can process 175k TPS at a latency of less than 0.5s—a single host throughput improvement of 8x-10x compared with Zef.

The “Confirm” column shows the delay between block creation and finalization. Neuron’s Proposal-First model achieves **0ms confirmation delay** because blocks are born finalized.

17 Production Deployment

We collaborated with the QoraNet team to integrate Neuron-C as a replacement for Bullshark. The deployment spans 106 validators across 13 geographic regions.

Performance Results: QoraNet equipped with Neuron-C demonstrates p50 latency of 380ms and p95 latency of 650ms for 106 validators. In contrast, the previous Bullshark deployment exhibited p50 latency of 1.9s and p95 latency of 3.1s—a 4x improvement.

18 Related Work

NEURON builds on a rich history of BFT consensus research. The pioneer work on DAG-based consensus includes DAG-Rider [1], Narwhal-Tusk [2], and Bullshark [3]. These protocols use certified DAGs which increase latency. Cordial Miners [39] proposed a similar uncertified DAG structure but lacks implementation. The recent Mysticeti protocol [38] also achieves 3-round latency through uncertified DAGs.

For fast path protocols, FastPay [20], Zef [21], and Astro [22] pioneered consensusless payments. Sui Lutris [23] combines fast path with DAG consensus but as separate components.

19 Conclusion

We introduce NEURON-C, a threshold clock-based Byzantine consensus protocol with WAN latency of 260ms and throughput over 150k TPS. We additionally present NEURON-FPC, achieving 8x better resource efficiency than prior fast path protocols. The key innovations include the 2/3 VRF lottery quorum, VDF anti-grinding protection, and finality certificates for efficient state sync. NEURON-C has been deployed in production on QoraNet with over 100 validators.

References

[1] I. Keidar et al., “All you need is DAG,” PODC, 2021.

[2] G. Danezis et al., “Narwhal and Tusk,” EuroSys, 2022.

[3] A. Spiegelman et al., “Bullshark,” CCS, 2022.

[4] A. Spiegelman et al., “Shoal,” 2023.

[5] Y. Gao et al., “Dumbo-NG,” CCS, 2022.

[6] L. Yang et al., “DispersedLedger,” NSDI, 2022.

[7] N. Shrestha et al., “Sailfish,” 2024.

[8] C. Stathakopoulou et al., “BBCA-ledger,” 2023.

[9] D. Malkhi, P. Szalachowski, “MEV protection,” 2022.

[10] G. Giuliari et al., “DoS Resilience,” AsiaCCS, 2024.

[11] C. Cachin et al., “Reliable Distributed Programming,” 2011.

[12] R. Gelashvili et al., “Jolteon and Ditto,” FC, 2022.

[13] M. Baudet et al., “Libra blockchain,” 2019.

[14] M. Castro, B. Liskov, “PBFT,” OSDI, 1999.

[15] R. Kotla, M. Dahlin, “High throughput BFT,” 2004.

[16] K. Chalkias et al., “FastCrypto,” 2024.

[17] Z. Li et al., “EdDSA and BLS performance,” 2023.

[18] B. Ford, “Threshold logical clocks,” 2019.

[19] J. Martin, L. Alvisi, “Fast Byzantine consensus,” 2006.

[20] M. Baudet et al., “FastPay,” AFT, 2020.

[21] M. Baudet et al., “Zef,” 2022.

[22] D. Collins et al., “Online payments,” 2020.

[23] S. Blackshear et al., “Sui Lutris,” 2023.

[24] The Linera Team, “Linera,” 2023.

[25] S. Cohen et al., “Proof of availability,” FC, 2023.

[26] Y. Sompolinsky, A. Zohar, “GHOST,” FC, 2015.

[27] B. Wesolowski, “Efficient VDF,” EUROCRYPT, 2019.

[28] R. Neiheiser et al., “Chiron,” 2024.

[29] The Tokio Team, “Tokio,” 2024.

[30] H. de Valence, “ed25519-consensus,” 2024.

[31] RustCrypto, “blake2,” 2024.

[32] The RocksDB Team, “RocksDB,” 2024.

[33] Linux, “writev(3),” 2024.

[34] C. Dwork, N. Lynch, L. Stockmeyer, “Partial synchrony,” 1988.

[35] S. Goldberg et al., “ECVRF,” IETF Draft, 2023.

[36] G. Tsimos et al., “HammerHead,” 2023.

[37] L. Kokoris-Kogias et al., “Cuttlefish,” 2023.

[38] K. Babel et al., “Mysticeti,” 2024.

[39] I. Keidar et al., “Cordial Miners,” DISC, 2023.

[40] The Aptos Team, “Aptos,” 2023.

[41] The Flow Team, “Flow,” 2023.

[42] Monad, “MonadBFT,” 2023.

A Example NEURON-C Execution

This appendix provides a step-by-step walkthrough of the NEURON-C decision rule. Consider a DAG with four validators (A_0, A_1, A_2, A_3) and four proposer slots per round. Initially, all proposers are marked as undecided.

The validator applies the direct decision rule starting with the latest slots. For L_{4d}, it observes $2f + 1 = 3$ certificate patterns and marks it as to-commit. Similarly, L_{4c} and L_{4b} are marked to-commit. For L_{4a}, a skip pattern is observed (3 blocks do not support it), so it is marked to-skip.

For L_{2c}, the direct decision rule fails. The indirect decision rule finds anchor L_{5a} which is undecided, so L_{2c} remains undecided. For L_{1d}, the anchor L_{4b} is to-commit with a certified link, so L_{1d} is marked to-commit. The final commit sequence is $L_{1a}, L_{1c}, L_{1d}, L_{2a}$.

B Detailed Neuron-FPC Protocol

This appendix provides additional details on fast path execution and finality.

Fast Path Execution: A validator safely executes a fast path transaction T once it observes blocks from $2f + 1$ validators containing votes for T. The execution effects are known immediately, and subsequent transactions on the same object can proceed.

Fast Path Finality: Transaction T is finalized when either: (1) $2f + 1$ certificate patterns exist, each containing $2f + 1$ votes for T, or (2) a single certificate pattern with $2f + 1$ votes is referenced by a committed consensus block.

Mixed-Object Transactions: Transactions with both owned and shared object inputs require votes from $2f + 1$ validators AND inclusion in a committed consensus block for finalization.

C Additional Security Proofs

Lemma 18. *The round-robin schedule of proposers ensures that in any window of $3f + 3$ rounds, there are three consecutive rounds with honest primary proposers.*

Proof. There are $3f + 1$ groups of three consecutive rounds. Due to round-robin, each honest validator is primary proposer in exactly 3 groups. As there are $2f + 1$ honest validators, by pigeonhole principle, one group must contain $\lceil 3(2f+1)/(3f+1) \rceil = 3$ honest proposers. □

Lemma 19. *After GST any undecided slot eventually gets decided.*

Proof. Let there be an undecided slot s in round r. After GST, due to the previous lemma, there will eventually be honest proposers for three consecutive rounds $k, k+1, k+2$ where $k > r$. By the Sufficient Votes lemma, these proposers will have $2f+1$ certificates and be scheduled for commit. All slots in rounds $\leq k - 1$ then get decided by induction. □

Theorem 20 (Agreement)**.** Neuron-C *satisfies the agreement property of Byzantine Atomic Broadcast.*

Proof. If a correct validator outputs $a\_deliver_i(b, r, v_k)$, then it must have committed a sequence of proposer blocks such that b is delivered from linearizing their causal histories. Another correct validator Y that has not delivered b will eventually see a proposal from an honest proposer in round $r' > r$. Due to Consensus Liveness, Y will commit this proposer's block. Due to the consistency lemma, Y will also commit the same sequence before this block. Since Y follows identical deterministic linearization, it also delivers b eventually. □

D Reproducing Experiments

We provide orchestration scripts for reproducing our experiments. The settings file includes AWS regions, instance types (`m5d.8xlarge`), and benchmark duration. Key commands:

`cargo run --bin orchestrator -- testbed deploy --instances 2` creates instances. `cargo run --bin orchestrator -- benchmark --committee 50 fixed-load --loads 1000` runs benchmarks.

Node parameters: leader_timeout = 1s. Client parameters: initial_delay = 400ms. The simulator has proven indispensable in identifying correctness defects, while the orchestrator has been instrumental in pinpointing performance bottlenecks.

Algorithm 5 NEURON-C Main Algorithm

```
committeeSize, waveLength ← 3
numOfProposers ← 2                                   ▷ Configurable

procedure TryDecide($r_{\text{committed}}, r_{\text{highest}}$)
    sequence ← []
    for $r \in [r_{\text{highest}}$ down to $r_{\text{committed}} + 1]$ do
        for $l \in$ [numOfProposers $-$ 1 down to 0] do
            $i \leftarrow r \mod$ waveLength
            $c \leftarrow$ DirectDecider(waveLength, $i, l$)
            $w \leftarrow c.\textsc{WaveNumber}(r)$
            if $c.\textsc{ProposerRound}(w) \neq r$ then
                continue
            end if
            status ← $c.\textsc{TryDirectDecide}(w)$
            if status $= \perp$ then
                status ← TryIndirectDecide($c, w$, seq)
            end if
            sequence ← status||sequence
        end for
    end for
    decided ← []
    for status ∈ sequence do
        if status $= \perp$ then break
        end if
        decided ← decided||status
    end for
    return decided
end procedure

procedure TryIndirectDecide($c, w$, sequence)
    $r_{\text{decision}} \leftarrow c.\textsc{DecisionRound}(w)$
    anchors ← [$s \in$ sequence : $r_{\text{decision}} < s$.round]
    for $a \in$ anchors do
        if $a = \perp$ then return $\perp$
        end if
        if $a =$ Commit($b_{\text{anchor}}$) then
            $b_{\text{proposer}} \leftarrow c.\textsc{GetProposerBlock}(w)$
            if $c.\textsc{CertifiedLink}(b_{\text{anchor}}, b_{\text{proposer}})$ then
                return Commit($b_{\text{proposer}}$)
            else
                return Skip($w$)
            end if
        end if
    end for
    return $\perp$
end procedure
```

Algorithm 6 GHOST Fork Choice

```
procedure SelectBestChain(genesis)
    current ← genesis
    while |children(current)| > 0 do
        bestChild ← $\perp$
        bestWeight ← $-1$
        for $c \in$ children(current) do
            $w \leftarrow \textsc{ComputeWeight}(c)$
            if $w >$ bestWeight or ($w =$ bestWeight and $H(c) < H$(bestChild)) then
                bestChild ← $c$
                bestWeight ← $w$
            end if
        end for
        current ← bestChild
    end while
    return current
end procedure

procedure ComputeWeight($B$)
    if $B \in$ cache then
        return cache[$B$]
    end if
    $w \leftarrow 1$
    for $c \in$ children($B$) do
        $w \leftarrow w + \textsc{ComputeWeight}(c)$
    end for
    cache[$B$] ← $w$
    return $w$
end procedure
```

Algorithm 7 Proposal-First Consensus Flow

```
procedure ProposerFlow(slot)
    $P \leftarrow$ CreateProposal($slot$)    ▷ No certificate yet
    Broadcast(P)
    votes $\leftarrow$ [], power $\leftarrow 0$
    threshold $\leftarrow \lfloor 2 \cdot \text{totalPower}/3 \rfloor + 1$
    while now() $<$ deadline do
        $v \leftarrow$ ReceiveVote
        if $v$.proposalHash $= H(P)$ and $v$.validator $\neq$ self then
            votes.append($v$)
            power $+=$ stake[$v$.validator]
            if power $\geq$ threshold then
                $B \leftarrow$ CreateBlockFromProposal($P$)
                $B$.certificate $\leftarrow$ CreateCert(votes)
                Broadcast(B)    ▷ Born finalized!
                return $B$
            end if
        end if
    end while
    return $\bot$    ▷ Timeout—no block created
end procedure

procedure ValidatorVoteFlow(P)
    if not ValidateProposal(P) then return
    end if
    if $P$.proposer = myAddress then return
    end if    ▷ No self-vote
    msg $\leftarrow H(P) \| \text{myAddress}$
    $\sigma \leftarrow$ Sign(sk, msg)
    SendToProposer((msg, $\sigma$))
end procedure

procedure OnReceiveFinalizedBlock(B)
    if VerifyCertificate($B$.certificate) then
        AcceptAsFinalized(B)    ▷ Instant acceptance
    end if
end procedure
```

Algorithm 8 Proposal-Level TX Validation

```
procedure ValidateTxsForProposal(txs)
    validTxs $\leftarrow$ [], proofs $\leftarrow$ []
    for $tx \in txs$ do
        (nonce, balance) $\leftarrow$ GetState($tx$.sender)
        if $tx$.nonce $\neq$ nonce then
            continue    ▷ Invalid nonce
        end if
        if $tx$.value $+$ $tx$.gasCost $>$ balance then
            continue    ▷ Insufficient balance
        end if
        proof $\leftarrow$ CreateRSProof($tx$, nonce, balance)
        validTxs.append($tx$)
        proofs.append(proof)
    end for
    return (validTxs, proofs)
end procedure
```

Algorithm 9 Proposal TX Tracking

```
procedure CreateProposal(slot)
    $txs \leftarrow$ ReapFromMempool    ▷ Skip tracked TXs
    proposal $\leftarrow$ BuildProposal($txs$)
    TrackTxs(txs)    ▷ Mark as in pending proposal
    return proposal
end procedure

procedure OnReceivePeerProposal(proposal)
    ValidateProposal(proposal)
    TrackTxs(proposal.txs)    ▷ Track peer's TXs too
end procedure

procedure OnProposalOutcome(proposal, success)
    if success then
        MarkTxsIncluded(proposal.txs)    ▷ Remove from mempool
    end if
    UntrackTxs(proposal.txs)    ▷ Allow retry if failed
end procedure
```

Algorithm 10 Finality Certificate Verification

```
procedure VerifyCertificate(cert, knownCommittee)
    if cert.CommitteeRoot $\neq H$(knownCommittee) then
        return false
    end if
    totalPower $\leftarrow 0$
    seen $\leftarrow \{\}$
    for $\sigma \in$ cert.Votes do
        ($v$, sig) $\leftarrow \sigma$
        if $v \in$ seen then
            return false    ▷ Duplicate vote
        end if
        if $v \notin$ knownCommittee then
            return false    ▷ Not committee member
        end if
        msg $\leftarrow$ cert.BlockHash$\|$cert.Level
        if not VerifySig($pk_v$, msg, sig) then
            return false
        end if
        totalPower $\leftarrow$ totalPower $+$ stake[$v$]
        seen.add($v$)
    end for
    return totalPower $\geq$ cert.Threshold
end procedure
```