Consensus on Solana

Actionable Insights

- The role of Proof-of-History (PoH) in synchronization

: PoH is not a consensus algorithm, but rather a tool used by Solana's consensus mechanism for synchronization. Similarly, Proof-of-Stake (PoS) is not consensus, but actually sybil resistance.

- Vote transactions are necessary for consensus

: Votes within blocks are not extraneous transactions to artificially inflate TPS metrics. If votes were merely propagated through gossip (informal, peer-to-peer communication), it could lead to discrepancies in how validators perceive the Tower's (vote) state.

- Solana has two primary confirmation rules

: one for short-term fork selection (optimistic confirmation

) and another for full PoS consensus for finality (finalized/rooted

). Clients and users are able to adhere to these confirmation rules for desired security properties and customized UX selections. This is reflected by two commitment levels: "confirmed" and "finalized".

- Understanding censorship risk

: Validators and developers should be aware of the potential for censorship attacks, where a validator attempts to disrupt the sequence of block production. Understanding the mechanics of such attacks and the role of computational power and stake in executing them.

- Future protocol upgrades on the horizon

: Validators and developers should actively prepare for upcoming changes in Solana's consensus mechanism such as asynchronous execution and programmatic slashing.

Introduction

As activity picks up on Solana, various layers of its stack are being tested at unprecedented levels. Much has been written and discussed about "hot" topics, such as local fee markets, but consensus on Solana has long been overlooked. However, increased activity and interest demand a thorough understanding of consensus by the community as the incentives for malicious attacks and the profit from potential exploitations increase.

Consensus is one of the most important aspects of Solana for the community at large to understand, as it determines how thousands of validators agree on a canonical ordering of transactions.

Consensus has been studied in distributed systems for many years. The Byzantine Generals Problem, written by Lamport, Shostak, and Pease, was published in the early 1980s. Consensus algorithms such as RAFT have long been used in web2. In crypto, most consensus algorithms are differing implementations of BFT consensus, including Gasper (Ethereum), Tendermint (Cosmos), MonadBFT (Monad), HotShot (Espresso), and Narwhal/Tusk (Sui).

This article does not attempt to formally prove Solana's consensus mechanism (TowerBFT). Instead, it aims to explain how consensus works on Solana to developers and the broader community, as this is a topic that, so far, is mostly in the minds of Solana Labs and Firedancer contributors. Some commentary around tradeoffs and limitations is also discussed.

A Brief Primer on Consensus

The objective of a consensus protocol is to provide an agreement on transactions and their relative orderings within a block. There are two main types of consensus protocols for validators in a network to come to an agreement on the canonical ordering of transactions:

- Longest-chain protocols

: these protocols (such as Bitcoin's Nakamoto consensus) canonicalize the chain that has required the most computational effort to construct. While this often correlates with the chain having the largest number of blocks, the more accurate description is the chain embodying the greatest amount of cumulative work or computational power.

- BFT-type protocols

: most PoS protocols implement a version of a BFT consensus algorithm. These protocols (such as pBFT) rely on thresholds for liveness and security. Consistency and availability are two guarantees of the CAP theorem, which states that any distributed data store can only offer two out of the following three guarantees: consistency, availability, and partition tolerance.

Both longest-chain and BFT-type consensus protocols derive security from their respective confirmation rules.Security, consisting of both safety and liveness, is derived from a given confirmation rule and is not a property of a chain. As defined by the Ethereum Foundation, a confirmation rule is “an algorithm run by nodes, outputting whether a certain block is confirmed. When that is the case, the block is guaranteed to never be reorged, under certain assumptions, primarily about network synchrony and about the percentage of honest stake.”

It’s social consensus all the way down, ultimately defined by people writing client code that expresses what defines security via certain confirmation rules.

Proof-of-Stake introduces additional layers to the BFT model, requiring participants to put forward their own stake. Participants are rewarded for adhering to a set of rules but can be slashed in cases of proven misconduct (e.g., double signs). This is referred to as accountable safety and allows the protocol to identify and punish malicious nodes with no externalities to honest nodes. This does not replace the fundamental security mechanism of BFT consensus protocols: the network still needs less than one-third of dishonest nodes to avoid halting and less than two-thirds to prevent validating false transactions. The stake-based system imposes consequences for actions that might disrupt or degrade network efficiency.

There are two critical thresholds in such a BFT network:

1. 1/3

: If dishonest nodes make up one-third or more of the total, the network may “halt.” In this scenario, these nodes might simply opt out of participation, leaving the rest unable to attain the two-thirds supermajority needed for consensus. Consequently, the network doesn’t produce incorrect transactions. Rather, it ceases to produce any transactions at all. One popular (albeit crude metric) is the Nakamoto Coefficient, which represents the minimum number of nodes required to produce a liveness failure (halting block production).

1. 2/3

: If dishonest nodes constitute two-thirds or more of the total, they can collude to validate any transactions they choose. This represents the worst-case scenario, where the network ceases to function correctly and instead processes transactions as dictated by the dishonest supermajority. In a scenario where an adversarial party controls over 67% of the stake, they could potentially isolate an honest node, such as a major exchange’s node (e.g., Binance). This isolation could be achieved through collusion with a data center to restrict the node’s network traffic. The malicious entity could then have this isolated node finalize a block that it has crafted while simultaneously distributing a conflicting block to the rest of the network. This sort of attack exploits the isolated node’s limited perspective, leading to potential double-spending as the rest of the network and the isolated node have different views of the on-chain state.

A similar attack is feasible even with a lower stake, over 33%, but this would require creating a network partition rather than mere isolation. In this situation, the Byzantine stakeholder can exploit the network partition to manipulate different segments of the network with conflicting information, again risking double spending and other safety failures.

Typically, a larger number of nodes increases the difficulty for any party trying to corrupt a significant portion to reach these thresholds (assuming geographical separation). However, this desire for a larger network often conflicts with efficiency. Higher numbers of nodes can slow down consensus due to the increased data transmission demands (more time required for vote propagation), with some protocols having a hard cap on the maximum number of nodes.

Proof-of-History (PoH)

While Proof-of-Stake (PoS) ensures consensus in a network, Solana incorporates Proof of History (PoH) into its PoS consensus mechanism, enabling synchronization for continuous block production. This is achieved by Solana skipping slots with slow or unresponsive slot leaders without waiting for a synchronized round of consensus. PoH is not concerned with proving the exact time an event occurred but rather with proving the sequence and the passage of time between events.

Contrary to popular belief, Proof-of-History (PoH) itself is not a consensus mechanism or algorithm. While

[the current implementation

](https://notes.ethereum.org/@dankrad/BJywSZY9t)of consensus utilizes aspects of PoH, one could theoretically strip out PoH and make some minor implementation changes such that consensus on Solana still operates.

The core of PoH is a simple hashing algorithmakin to a Verifiable Delay Function (VDF) but isnot technically a VDF. Solana implements this using a sequential pre-image resistant hash function (SHA-256) that continuously runs, using the output of one iteration as the input for the next. This computation runs on a single core of each validator.

While the sequence generation is sequential and single-threaded, the output can be verified in parallel, allowing for efficient verification on multi-core systems. While there exists an upper bound on hashing speed, improvements in hardware can potentially offer additional performance benefits.

Consider an example involving four validators on the Solana network: Validators A, B, C, and D. In this example, theleader schedule might dictate the sequence of block production as A - B - C - D. Validator A starts by generating a block in turn. To do so, Validator A uses the PoH mechanism, which involves running the SHA-256 hash function iteratively, forming a timescale of “ticks”. This hashing process creates a unique and verifiable record of time spent, ensuring that Validator A’s block reflects the accurate passage of time. Once Validator A completes its block, it’s Validator B’s turn to produce the next block, followed by Validator C.

Suppose Validator C attempts to disrupt the sequence by emitting a block out of turn, aiming to follow Validator A directly, thus bypassing Validator B. For Validator C to convincingly take Validator B’s place, it would need to replicate the PoH hash sequence that Validator B would have produced, which means it must generate a sequence of hashes that represents the time Validator B would have taken to produce a block. There exists a physical maximum on single-core hashing performance, meaning that we know a certain amount of time has elapsed as there exists a maximum number of hashes a computer can generate in a given time interval.

Validator C can attempt to censor Validator B in the leader schedule by generating a chain of empty blocks, starting from the end of a previous legitimate block (Validator A’s block), without including any transactions. To successfully censor B, C must fulfill two conditions. Firstly, C needs the computational power to grind the empty PoH chain. Secondly, C must rapidly disseminate its block to a sufficient number of staked nodes to ensure its block is accepted during its own designated slot, thereby effectively censoring B. This is faster as a validator with high stake-weight thanks to Turbine, which prioritizes information flow by validator stake-weight.

This attack scenario is feasible, but the window for successful censorship is limited. It requires the attacking node to possess substantial computational resources (for SHA-256 hashing) and a significant amount of stake, as the number of blocks it can generate is proportional to its stake.

Besides needing to reach the network more rapidly than node A, C must also generate the hash at a high speed. This attack primarily targets the scenario where a validator with a designated leader at slot n

aims to censor validators with previous leader slots (previous to n

). The extent of censorship achievable by this validator depends on its stake, as its ability to create leader slots is directly tied to the amount of stake it holds.

The PoH mechanism also ensures that blocks are produced at a consistent rate. Since the PoH sequence can be independently verified by each validator, there’s no need for external time synchronization. Ethereum, for example, uses the Network Time Protocol (NTP) in each block for consensus. On Solana, each validator independently verifies that each block was produced in the correct time slot endogenously, without the use of an external protocol.

Tower BFT (Solana’s Consensus Mechanism)

Tower BFT, Solana’s consensus mechanism, operates after shreds (partial blocks) have been propagated to other validators:

As mentioned earlier, Solana runs Tower BFT alongside Proof-of-History. Tower BFT is a pBFT-like consensus algorithm designed to take advantage of the synchronized clock computations from Proof-of-History. This sets up a universal clock across the network, enabling it to efficiently bypass slots assigned to leaders who are either slow or not responding. This process eliminates the need for the network to undergo a synchronous consensus round for each slot and enables continuous block production, as validators do not have to wait for previous blocks to arrive before building the next block.

Another misconception is that Solana’s consensus mechanism currently implements programmatic slashing.

While slashing is on the roadmap, the network currently halts after a safety violation, relying on social consensus for slashing as needed.

Solana’s consensus mechanism provides varying degrees of influence to different nodes. Votes in the network are not equal but are weighted according to each node’s stake, functioning under the same principles of stake-weighted QoS and Turbine. All else equal, a node with a larger stake has more influence on determining the canonical consensus than one with a smaller stake.

For example, in a network with four nodes holding a total of 100 stake, the distribution and influence might be as follows:

- Node A has 10 units of stake.
- Node B has 20 units of stake.
- Node C has 30 units of stake.
- Node D has 40 units of stake.

In this setup, the capability of each group of dishonest nodes varies. A single node, like Node D, could halt the network despite being only 25% of the total number of nodes due to its large stake. Similarly, a combination like Nodes C and D could approve incorrect transactions, representing only 50% of the nodes but holding enough stake to influence the outcome.

The one-third threshold for halting the network can be reached through dishonest, compromised, or blocked nodes. The two-thirds threshold for validating incorrect transactions; however, this requires active collusion and cannot rely on merely blocking honest nodes. Therefore, achieving the latter is significantly more challenging. This is because it involves corrupting or compromising nodes to actively participate in the dishonest scheme rather than blocking honest ones.

Context within Slots

On Solana, slot leaders are assigned four consecutive slots lasting approximately 1.6 seconds in total (4 blocks x 400 ms per block). Leaders are chosen at the beginning of each epoch (432,000 slots or ~2-3 days). The leader schedule is chosen randomly, relative to a validator's stake-weight.

The leader is assigned with building and proposing a new block for each of their assigned slots (there is no proposer-builder separation as it exists in the Ethereum world). Other validators attest to the validity of a given block via vote transactions by applying the fork-choice rule to their own local view of the head of the Solana network.

Leaders must publish blocks within a given PoH tick range in order for that block to be valid; a block not published within this range is considered skipped.

Take the following example of a leader schedule with four participants (A, B, C, D), where D tries to disrupt the sequence. If D attempts to intervene during C's turn, it must create a sequence of PoH ticks that effectively skips C's block, leading to a chain that looks like:

A - B - [C missing] - D.

In a situation where C's block is missing, an honest D would have to generate a PoH sequence that covers the entire duration of C's slot before starting its own. This would make D's block appear valid since it follows B's block after the time allocated for C's slot.

While D produces the PoH sequence for C's slot, C would normally be streaming its block, connected to B's with an appropriate PoH sequence. This leads to two possible outcomes:

- If D's PoH computation is not faster than C's

: As C completes its block around the same time D starts broadcasting its own, the network, having seen C's block, will reject D's block.

- If D is faster at computing PoH than C

: D begins broadcasting its block before C completes its own. Even so, D would need to be substantially quicker than C to significantly affect the sequence, which is unlikely given the high speed and (relatively) similar upper bound of capabilities of the CPUs used by validators for SHA-256 computation.

In both situations, D's chances of successfully replacing C are minimal. Furthermore, if D's attempt to replace C fails, it forfeits the opportunity to emit a block in its own slot. Emitting a block right after B and then trying another after C constitutes a violation (creating two blocks for D's slot), which would result in a slashable offense. Each failed attempt leads to D missing its own chance to produce a block and suggests it is an unlikely strategy to be pursued by D.

The intentions of D are indeterminate from the network's perspective. It is difficult, if not impossible, to differentiate between D simply not seeing C's block (with no intention to censor C) or D had the intention of censoring B. As a result, this is not easily detectable and not punishable by the network.

Vote Transactions

Solana's consensus mechanism relies on vote transactions to achieve consensus. Vote transactions live within a block but are prioritized so as not to be drowned out by regular transactions. Currently, the majority of transactions within a given block are vote transactions:

However, this may not always be the case, as the percentage of vote transactions within a given block represents the ratio between activity and the number of validators participating in consensus. In the future, vote transactions may be the minority of a given block.

Vote transactions are necessary requirement for consensus – they are not extraneous transactions to artificially inflate TPS metrics.

If votes were merely propagated through gossip (informal, peer-to-peer communication), it could lead to discrepancies in how validators perceive the Tower's (vote) state. Such discrepancies might result in validators having different perspectives on which fork is more likely to be the correct one, leading to potential divergence as validators make greedy choices based on incomplete or inconsistent information. Continuous block production requires continual voting, especially while previous blocks are still being confirmed. This demands a reliable and consistent view of the Tower's state.

Like regular transactions initiated by users, vote transactions must also pay the base fee (0.000005 SOL). This is paid by the validator identity, which must reside in a hot wallet for constant signing, while the vote account is used for account lookup and delegation.

Each vote, signed by the validator, includes the validator's public key and the hash of the block they are voting for.

When a validator receives multiple blocks for the same slot, it tracks all possible forks until it can determine the "best" one. Validators run the relevant state transition function locally (this will change under asynchronous execution) and subsequently vote on new blocks after replay. Validators express their votes for a given fork via on-chain vote transactions.

In each vote transaction, a validator publishes votes with a lockout period. This lockout acts as a commitment mechanism, binding validators to their chosen fork and imposing opportunity costs for their decisions. Today, lockouts are not enforced by the runtime but are enforced via social consensus – violating voting lockouts consistently will very likely be manually slashed. Programmatic slashing for violating lockout periods is likely to be added in the near future, as not receiving vote credits does not adequately provide enough of a disincentive.

The validator also signs a hash of the block for every ancestor block between their previous vote and rooted/finalized blocks. This is necessary to detect duplicate blocks. If a leader were to send distinct blocks to each node, every node would end up voting on a different predecessor for the same slot. However, in an epoch consisting of 65,000 slots, the size of each vote could reach up to 2 MB in the most extreme scenario. This is because it might require hashes of 32 bytes for each of the 65,000 slots. This will be explored more in a future post.

The validators manage a "Vote Tower", a sequential stack of votes where each vote reinforces a fork and is an ancestor to the fork above it in the tower. The addition of a new vote to this tower results in the doubling of lockouts for all previous votes in the stack, progressively increasing the commitment and lockout period for earlier decisions. The voting process itself is governed by several checks: validators must respect the lockout periods of previous votes; they need to ensure that a significant portion of the network (usually two-thirds) is also committed to the same fork; and a substantial majority (over 38%) of votes on alternative forks is required before switching to a new fork.

For a block at slot n

, votes outside of the leader begin appearing on-chain as early as n+1

. There are no vote subcommittees – all validators are eligible to vote on all blocks. These votes first appear in blocks from other validators close (one hop away) in the Turbine tree. Votes can appear for slot n

at up to slot n+512

because a validator is allowed to vote on any slot whose "slot hash" is still known, and slot hashes are retained for 512 slots (h/t Shinobi). There is no predetermined upper bound at a certain block for slot n

; however, rooted slots built upon at least 32 continuous blocks will no longer accept votes and become finalized.

There are some quirks in the voting scheme that are not fully enforced by the runtime. For example, validators are currently incentivized to only vote for "rooted" slots (ignoring the tip of the chain), and this is not punished by consensus. This allows a validator to continually earn vote credits for forks with an extremely high likelihood of being canonical while not contributing to the tip of the chain for actual consensus operations. This is occurring today, with a validator having an average "vote latency" of over 68 slots. A proposed feature called Timely Vote Credits aims to mitigate this incentive misalignment.

Solana's Fork Choice Rule

Like Ethereum (LMD Ghost and Casper FFG), Solana has two confirmation rules – one for short-term fork selection and another for full PoS consensus for finality. This enables users and clients to make customized UX selections under different confirmation rules. This is reflected by two commitment levels: "confirmed" and "finalized".

"Confirmed" blocks (also known as optimistic confirmation) require at least 2/3 of validators to have voted via vote transactions on a particular block. ~4.6% of the current stake would have to be slashed for finality violations to be possible."Finalized" blocks require votes on at least 32 subsequent slots or a supermajority (>2/3) of votes. A malicious attack would require over 1/3 of the stake to be slashed. This takes substantially more time than "confirmed" blocks as it must wait for rooted blocks 32 blocks previously for full finality.

A fork-choice rule allows the network to reach consensus about the head of the chain. The Solana Labs implementation primarily handles fork choice by a ~4600 line Rust file aptly named heaviest_subtree_fork_choice.rs. This provides the necessary logic for validators to determine which fork is most likely to become canonical. A more detailed code breakdown will be explored in a future post, but the very high-level mechanics of fork-choice works as follows:

1. Votes are added with add_votes():

This iterates through the new votes, subtracts vote account stakes from old slots if needed, and adds stake to the new voted slots.

1. generate_update_operations() is called to create a batch of fork update operations:

This:

- Subtracts stake from old fork
- Adds stake to new fork
- Aggregates stake up each fork to the root
- process_update_operations():
- Calls mark_fork_valid()/invalid() if needed
- Calls aggregate_slot() up each fork to update fork weights
- Calls add_slot_stake()/subtract_slot_stake() to update stakes
- aggregate_slot():
- Sums up stake for all child slots
- Finds the heaviest child fork by stake weight
- Finds the deepest child fork by height
- Propagates these values up to find the heaviest/deepest fork from this slot
- select_forks():
- Returns the heaviest overall fork for block production
- Returns the heaviest fork descending from the last vote

Stake is added/subtracted based on votes, aggregation recalculates weights bottom-up for each fork, and the root with the heaviest weighted subtree is chosen as the best fork to build and vote on.

Likelihood of Inclusion

The likelihood of transaction inclusion changes over the lifecycle of a transaction after achieving the necessary requirements for the respective confirmation rules.

While slots can be “skipped”, such as when the leader is offline, that transaction can either land in future blocks or not land at all.

Slots here are rough approximations but attempt to give the reader an idea of when votes tend to accumulate. Furthermore, this is unique for each block (blocks may arrive at the “confirmed” or “finalized” confirmation level at varying lengths). The risk of reversion monotonically decreases throughout the lifecycle of the transaction. Even after a block has been finalized (rooted), it can theoretically be forked via social consensus, removing it from the “canonical” chain.

Future Research Areas and Conclusion

This post has highlighted the inner workings of Tower BFT, Solana’s consensus mechanism, and other relevant mechanisms. We explored the role of Proof-of-History, vote transactions, and fork choice within the context of Solana for creating a canonical fork for the ordering of transactions.

There are numerous additional directions for research and formalization around these mechanisms to be explored, such as:

- Ethereum researchers have quantified the value of playingTiming Games, where block producers wait until the end of the slot to submit their blocks. What do these games quantitatively look like on Solana, and are they occurring today?
- Asynchronous execution is on the roadmap to become the main protocol change for 2024. What are the potential attack vectors and relevant security considerations for nodes that are simply voting on existing forks rather than computing the state transitions locally?
- Today, a significant minority of validators (<20%) run modifications to the Solana Labs or Jito-Solana client. What thresholds are they changing that are not governed by the runtime or confirmation rules?

- Implementing programmatic slashing. Currently, there is little economic incentive outside of social consensus to not execute highly profitable trading strategies without slashing.

- What does performance with respect to Solana’s consensus mechanism look like when running two completely different clients (even if they attempt to be defined by the same confirmation rules)?

- What are the expected performance/state reduction benefits ofvote subcommittees?

- What are the potential attack vectors when restarting from an optimistically confirmed slot as opposed to one that has been finalized? How should third-party off-chain solutions like exchanges think about using optimistic confirmation or full finality?

- Should slashed funds be used as insurance for transactions included in safety violations? What are the mechanics and incentives for a SOL-denominated insurance pool?

In this post, we’ve explored some of the mechanisms and thresholds Solana implements with its consensus mechanism and how it relates to typical block production by leaders within given slots. The recent increase in activity on Solana provides real-world liveness and safety tests of these mechanisms, with increased incentives for malicious attacks.

Thanks to anoushk (Tinydancer), dubbel06 (Overclock), Prithvi (Helius), Mert (Helius), and Jarry (Ellipsis Labs) for discussions and comments.