Fork Generation

The Solana protocol doesn't wait for all validators to agree on a newly produced block before the next block is produced. Because of that, it's quite common for two different blocks to be chained to the same parent block. In those situations, we call each conflicting chain a"fork."

Solana validators need to vote on one of these forks and reach agreement on which one to use through a consensus algorithm (that is beyond the scope of this article). The main point you need to remember is that when there are competing forks, only one fork will be finalized by the cluster and the abandoned blocks in competing forks are all discarded.

This section describes how forks naturally occur as a consequence oleader rotation .

Overview

Nodes take turns beingleader and generating the PoH that encodes state changes. The cluster can tolerate loss of connection to any leader by synthesizing what the leaderwould have generated had it been connected but not ingesting any state changes.

The possible number of forks is thereby limited to a "there/not-there" skip list of forks that may arise on leader rotation slot boundaries. At any given slot, only a single leader's transactions will be accepted.

Forking example

The table below illustrates what competing forks could look like. Time progresses from left to right and each slot is assigned to a validator that temporarily becomes the cluster "leader" and may produce a block for that slot.

In this example, the leader for slot 3 chose to chain its "Block 3" directly to "Block 1" and in doing so skipped "Block 2". Similarly, the leader for slot 5 chose to chain "Block 5" directly to "Block 3" and skipped "Block 4".

Note that across different forks, the block produced for a given slot isalways the same because producing two different blocks for the same slot is a slashable offense. So the conflicting forks above can be distinguished from each other by which slots they haveskipped . Slot 1 Slot 2 Slot 3 Slot 4 Slot 5 Fork 1 Block 1

Block 3

Block 5 Fork 2 Block 1

Block 3 Block 4 Fork 3 Block 1 Block 2

Message Flow

1. Transactions are ingested by the current leader.
2. Leader filters valid transactions.
3. Leader executes valid transactions updating its state.
4. Leader packages transactions into entries based off its current PoH slot.
5. Leader transmits the entries to validator nodes(in signed shreds)1. The PoH stream includes ticks; empty entries that indicate liveness of the
6. 1. leader and the passage of time on the cluster.
7. 1. A leader's stream begins with the tick entries necessary to complete PoH
8. 1. back to the leader's most recently observed prior leader slot.
9. Validators retransmit entries to peers in their set and to further downstream
10. nodes.
11. Validators validate the transactions and execute them on their state.
12. Validators compute the hash of the state.
13. At specific times, i.e. specific PoH tick counts, validators transmit votes
14. to the leader.1. Votes are signatures of the hash of the computed state at that PoH tick
15. 1. count.
16. 1. Votes are also propagated via gossip.
17. Leader executes the votes, the same as any other transaction, and broadcasts
18. them to the cluster.
19. Validators observe their votes and all the votes from the cluster.

Partitions, Forks

Forks can arise at PoH tick counts that correspond to a vote. The next leader may not have observed the last vote slot and may start their slot with generated virtual PoH entries. These empty ticks are generated by all nodes in the cluster at a cluster-configured rate for hashes/per/tickZ .

There are only two possible versions of the PoH during a voting slot: PoH withT ticks and entries generated by the current leader, or PoH with just ticks. The "just ticks" version of the PoH can be thought of as a virtual ledger, one that all nodes in the cluster can derive from the last tick in the previous slot.

Validators can ignore forks at other points(e.g. from the wrong leader), or slash the leader responsible for the fork.

Validators vote based on a greedy choice to maximize their reward described inTower BFT .

Validator's View

Time Progression

The diagram below represents a validator's view of the PoH stream with possible forks over time. L1, L2, etc. are leader slots, andE s represent entries from that leader during that leader's slot. Thex s represent ticks only, and time flows downwards in the diagram.

Note that anE appearing on 2 forks at the same slot is a slashable condition, so a validator observingE3 andE3' can slash L3 and safely choosex for that slot. Once a validator commits to a fork, other forks can be discarded below that tick count. For any slot, validators need only consider a single "has entries" chain or a "ticks only" chain to be proposed by a leader. But multiple virtual entries may overlap as they link back to the a previous slot.

Time Division

It's useful to consider leader rotation over PoH tick count as time division of the job of encoding state for the cluster. The following table presents the above tree of forks as a time-divided ledger.

leader slot L1 L2 L3 L4 L5 data E1 E2 E3 E4 E5 ticks since prev

x xx Note that only data from leader L3 will be accepted during leader slot L3. Data from L3 may include "catchup" ticks back to a slot other than L2 if L3 did not observe L2's data. L4 and L5's transmissions include the "ticks to prev" PoH entries.

This arrangement of the network data streams permits nodes to save exactly this to the ledger for replay, restart, and checkpoints.

Leader's View

When a new leader begins a slot, it must first transmit any PoH(ticks)required to link the new slot with the most recently observed and voted slot. The fork the leader proposes would link the current slot to a previous fork that the leader has voted on with virtual ticks. Previous Consensus Commitment Status Next Leader Rotation