Solana Leader Rotation

At any given moment, a cluster expects only one validator to produce ledger entries. By having only one leader at a time, all validators are able to replay identical copies of the ledger. The drawback of only one leader at a time, however, is that a malicious leader is capable of censoring votes and transactions. Since censoring cannot be distinguished from the network dropping packets, the cluster cannot simply elect a single node to hold the leader role indefinitely. Instead, the cluster minimizes the influence of a malicious leader by rotating which node takes the lead.

Each validator selects the expected leader using the same algorithm, described below. When the validator receives a new signed ledger entry, it can be certain that an entry was produced by the expected leader. The order of slots which each leader is assigned a slot is called aleader schedule .

Leader Schedule Rotation

A validator rejects blocks that are not signed by theslot leader . The list of identities of all slot leaders is called aleader schedule . The leader schedule is recomputed locally and periodically. It assigns slot leaders for a duration of time called anepoch . The schedule must be computed far in advance of the slots it assigns, such that the ledger state it uses to compute the schedule is finalized. That duration is called theleader schedule offset . Solana sets the offset to the duration of slots until the next epoch. That is, the leader schedule for an epoch is calculated from the ledger state at the start of the previous epoch. The offset of one epoch is fairly arbitrary and assumed to be sufficiently long such that all validators will have finalized their ledger state before the next schedule is generated. A cluster may choose to shorten the offset to reduce the time between stake changes and leader schedule updates.

While operating without partitions lasting longer than an epoch, the schedule only needs to be generated when the root fork crosses the epoch boundary. Since the schedule is for the next epoch, any new stakes committed to the root fork will not be active until the next epoch. The block used for generating the leader schedule is the first block to cross the epoch boundary.

Without a partition lasting longer than an epoch, the cluster will work as follows:

1. A validator continuously updates its own root fork as it votes.
2. The validator updates its leader schedule each time the slot height crosses an epoch boundary.

For example:

Let's assume an epoch duration of 100 slots, which in reality is magnitudes higher. The root fork is updated from fork computed at slot height 99 to a fork computed at slot height 102. Forks with slots at height 100, 101 were skipped because of failures. The new leader schedule is computed using fork at slot height 102. It is active from slot 200 until it is updated again.

No inconsistency can exist because every validator that is voting with the cluster has skipped 100 and 101 when its root passes 102. All validators, regardless of voting pattern, would be committing to a root that is either 102, or a descendant of 102.

Leader Schedule Rotation with Epoch Sized Partitions.

The duration of the leader schedule offset has a direct relationship to the likelihood of a cluster having an inconsistent view of the correct leader schedule.

Consider the following scenario:

Two partitions that are generating half of the blocks each. Neither is coming to a definitive supermajority fork. Both will cross epoch 100 and 200 without actually committing to a root and therefore a cluster-wide commitment to a new leader schedule.

In this unstable scenario, multiple valid leader schedules exist.

- A leader schedule is generated for every fork whose direct parent is in the previous epoch.
- The leader schedule is valid after the start of the next epoch for descendant forks until it is updated.

Each partition's schedule will diverge after the partition lasts more than an epoch. For this reason, the epoch duration should be selected to be much much larger then slot time and the expected length for a fork to be committed to root.

After observing the cluster for a sufficient amount of time, the leader schedule offset can be selected based on the median partition duration and its standard deviation. For example, an offset longer then the median partition duration plus six standard deviations would reduce the likelihood of an inconsistent ledger schedule in the cluster to 1 in 1 million.

Leader Schedule Generation at Genesis

The genesis config declares the first leader for the first epoch. This leader ends up scheduled for the first two epochs

because the leader schedule is also generated at slot 0 for the next epoch. The length of the first two epochs can be specified in the genesis config as well. The minimum length of the first epochs must be greater than or equal to the maximum rollback depth as defined inTower BFT .

Leader Schedule Generation Algorithm

Leader schedule is generated using a predefined seed. The process is as follows:

1. Periodically use the PoH tick height(a monotonically increasing counter)to seed a stable pseudo-random algorithm.
2. At that height, sample the bank for all the staked accounts with leader identities that have voted within a cluster-configured number of ticks. The sample is called theactive set
3. .
4. Sort the active set by stake weight.
5. Use the random seed to select nodes weighted by stake to create a stake-weighted ordering.
6. This ordering becomes valid after a cluster-configured number of ticks.

Schedule Attack Vectors

Seed

The seed that is selected is predictable but unbiasable. There is no grinding attack to influence its outcome.

Active Set

A leader can bias the active set by censoring validator votes. Two possible ways exist for leaders to censor the active set:

- Ignore votes from validators
- Refuse to vote for blocks with votes from validators

To reduce the likelihood of censorship, the active set is calculated at the leader schedule offset boundary over anactive set sampling duration . The active set sampling duration is long enough such that votes will have been collected by multiple leaders.

Staking

Leaders can censor new staking transactions or refuse to validate blocks with new stakes. This attack is similar to censorship of validator votes.

Validator operational key loss

Leaders and validators are expected to use ephemeral keys for operation, and stake owners authorize the validators to do work with their stake via delegation.

The cluster should be able to recover from the loss of all the ephemeral keys used by leaders and validators, which could occur through a common software vulnerability shared by all the nodes. Stake owners should be able to vote directly by co-signing a validator vote even though the stake is currently delegated to a validator.

Appending Entries

The lifetime of a leader schedule is called anepoch . The epoch is split intoslots , where each slot has a duration ofT PoH ticks.

A leader transmits entries during its slot. AfterT ticks, all the validators switch to the next scheduled leader. Validators must ignore entries sent outside a leader's assigned slot.

AllT ticks must be observed by the next leader for it to build its own entries on. If entries are not observed(leader is down)or entries are invalid(leader is buggy or malicious), the next leader must produce ticks to fill the previous leader's slot. Note that the next leader should do repair requests in parallel, and postpone sending ticks until it is confident other validators also failed to observe the previous leader's entries. If a leader incorrectly builds on its own ticks, the leader following it must replace all its ticks. Previous Fork Generation Next Managing Forks