vim-markdown-preview.html

Introduction

The elections component enables functionality for accounts to become representatives, for representatives to become delegate candidates and for representatives to vote for delegate candidates.

During each epoch, there is a fixed set of delegate candidates. Throughout the epoch, each representative will issue one vote request. Each vote request can contain votes for up to 8 different candidates, and is processed through Axios consensus. Each vote is weighted by the casting representative's stake. At the conclusion of the epoch, voting ceases, and the 8 candidates that received the most weighted votes are written into the next epoch block, replacing the 8 oldest delegates who were elected in a previous epoch.

An account that wishes to vote must first become a representative, and can do so by issuing a `StartRepresenting` request. A representative can become a delegate candidate by issuing an `AnnounceCandidacy` request.

Overview

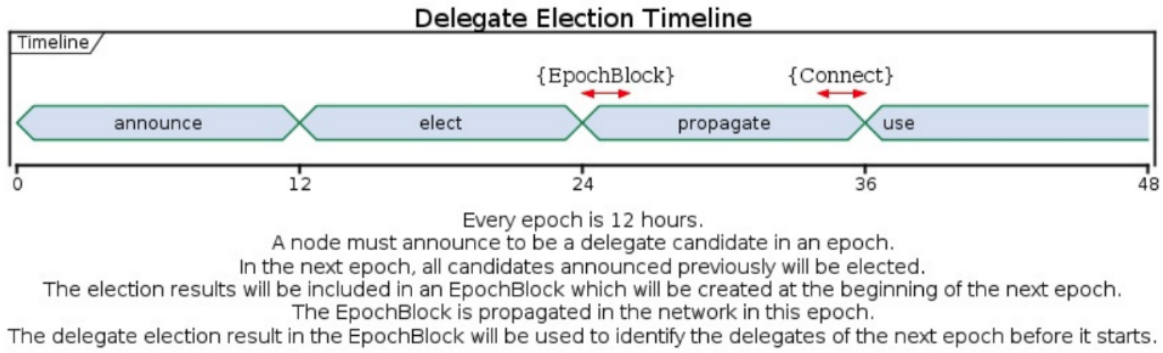
The elections component is not a single component but a collection of additions to existing components.

- Structs that inherit from `Request`
- Logic within `PersistenceManager<R>` to validate and apply these new `Request` types
- Representatives database and candidates database
- Logic within `EpochVotingManager` to select the next epoch's delegates
- Logic within `PersistenceManager<ECT>` to transition elections state to next epoch

Timeline

Candidates `AnnounceCandidacy` in epoch `i`, are voted on in epoch `i+1`, are written into the epoch block at the start of epoch `i+2` (assuming they won the election), serve as delegates from epoch `i+3` to `i+6`, and are eligible for reelection starting in epoch `i+5`. If a candidate is not elected, the candidate remains a candidate in subsequent epochs until they are elected or issue a valid `RenounceCandidacy` or `StopRepresenting`.

The epoch block post-committed in epoch `j` lists the delegates that will be active for epoch `j+1`.



Execution Concept

The processing of Requests related to elections follows the same execution model as processing any other type of Request (Send for instance).

The software keeps track of the current election results by maintaining a database, `candidacy_db`, consisting of the current candidates with their current total weighted votes received so far, which is updated each time an `ElectionVote` is post-committed. The software also maintains another database, `leading_candidates_db`, consisting of the current top 8 candidates with the most weighted votes, which is also updated each time an `ElectionVote` is post-committed.

The software keeps maintains a database, `remove_candidates_db`, consisting of any candidates that issued `RenounceCandidacy` during the epoch, as well as a database, `remove_reps_db`, consisting of any reps that issued `StopRepresenting` during the epoch.

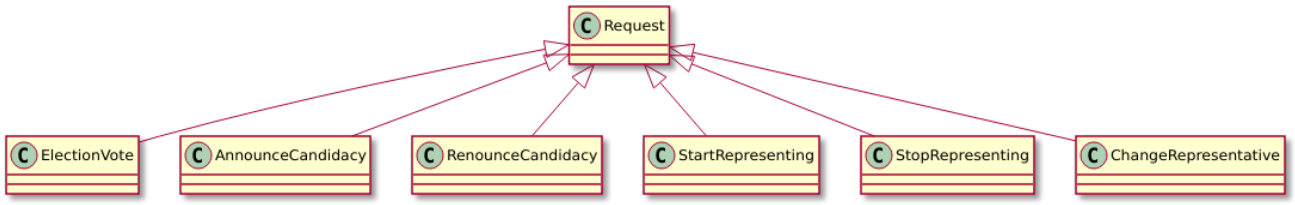
When the epoch ends, voting ceases until the epoch block is post-committed. When the epoch block is proposed, the winners of the election are read from `leading_candidates_db`. When the epoch block is applied, multiple state changes occur:

- Any candidates recorded in `remove_candidates_db`, as well as the election winners, are removed from `candidacy_db`
- Any representatives recorded in `remove_reps_db` are removed from the `representatives_db`.
- `Leading_candidates_db` is cleared. Note, the current total weighted votes for each candidate in `candidacy_db` are not reset on epoch transition, but instead are reset on the first vote that candidate receives in the next epoch.
- `remove_candidates_db` and `remove_reps_db` are cleared.

The time between epoch start and the epoch block being post-committed and written to the database (typically 10 minutes) is known as the elections dead period. No votes are accepted during the dead period; this is because prior to the epoch block being post-committed, the network has not come to consensus on the previous epoch's election results. If a candidate won the election in the previous epoch, that candidate can not receive votes in the next epoch. Delegates cannot be sure that their view of the election results is consistent with the other delegates until after the last microblock.

Interfaces

The interface to the election system is through issuing the proper `Request` which is processed through consensus.



- `StartRepresenting` Become a representative. Includes amount to stake.
- `AnnounceCandidacy` Become a candidate. If not already a representative, the account will become a representative as well. Includes amount to stake, which can be ommitted if account is already staking as a representative.
- `ElectionVote` Cast vote/s for a given epoch. Includes 1-8 current candidates
- `RenounceCandidacy` Remove from candidate list for upcoming epochs.
- `StopRepresenting` Remove from representative list for upcoming epochs. If sending account is also a candidate, this command also removes the account from the candidate list for upcoming epochs.

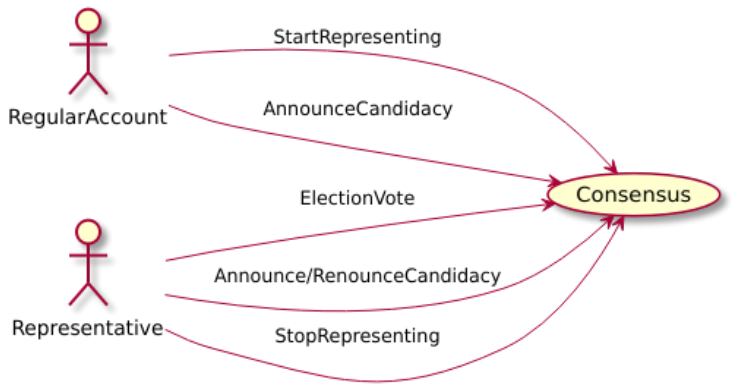
Each request, except `ElectionVote`, takes effect the epoch after that request is post-committed. For example, if an account issued `AnnounceCandidacy` in epoch `i`, then the account is a candidate no earlier than epoch `i+1`. If an account issued `StartRepresenting` in epoch `i`, the account can vote starting in epoch `i+1`. If an account issues `StopRepresenting` in epoch `i`, the account can still vote in epoch `i`, but not in any epoch after `i`. If an account issues `RenounceCandidacy` in epoch `i`, the account is still a candidate in epoch `i` (unless the account is a current delegate not yet eligible for reelection), but will not be a candidate in any epoch after `i`. Note, if a candidate issues a `RenounceCandidacy` in epoch `i` but also wins the election in epoch `i`, then the candidate still must serve as a delegate. However, the candidate will not be eligible for reelection, unless the account issues a subsequent `AnnounceCandidacy` request.

Note, a candidate can issue `StopRepresenting` while a candidate, which will remove that candidate from the candidate list and representative list for subsequent epochs. However, if that candidate wins the current epoch's election, that candidate must still serve as a delegate. This leads to the only situation where a delegate is not also a representative. The delegate can subsequently issue `StartRepresenting` or `AnnounceCandidacy` to become a representative and/or candidate again.

An account can only issue one of the above requests per epoch, except `ElectionVote`. For example, an account cannot issue `AnnounceCandidacy` and `RenounceCandidacy` in the same epoch. If an account wishes to become a representative and a candidate in one command, that account should issue `AnnounceCandidacy`. If an account wishes to stop being a representative and stop being a candidate in one command, that account should issue `StopRepresenting`.

See the IDD for further descriptions of the Requests.

Use Case Diagrams



Classes

`PersistenceManager<R>` (persistence manager for requests) has a `ValidateRequest(...)` method and `ApplyRequest(...)` method for each request type in the elections subsystem. The validation methods must take in the epoch number of the request, to ensure that the request is valid for that given epoch (for example, is the request during the dead period, did the representative already vote this epoch, etc). `PersistenceManager<ECT>` (persistence manager for epoch blocks) has a method `TransitionNextEpoch(...)` which updates the databases pertaining to representatives and candidates for the next epoch. This method calls various helper methods in the same class. `EpochVotingManager` has a method `GetNextEpochDelegates(...)` which determines the list of delegates to be written into the epoch block by swapping the retiring delegates with the election winners.

Resources

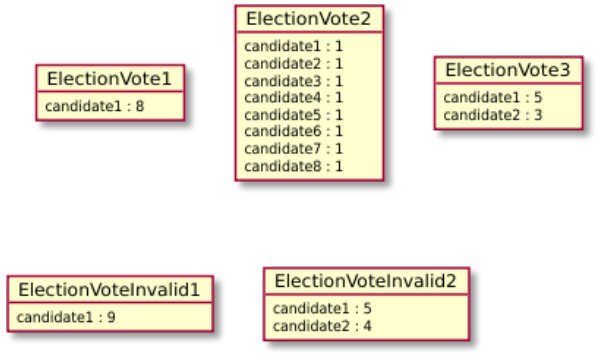
Elections will introduce five new databases. For detailed database format layout, please reference the database format documentation.

- `representative_db`: this database will store all of the representatives and any additional information about those representatives (voting weight, hash of their most recent vote, representative status). The hash of their most recent vote is stored, to ensure representatives only vote once. The hash of their most recent representative action (`StartRepresenting`, `StopRepresenting` or `AnnounceCandidacy`) is stored, as well as their most recent candidacy action (`AnnounceCandidacy`, `RenounceCandidacy` or `StartRepresenting`), to keep track of their status as a representative and candidate and validate future requests.
- `candidacy_db`: this database will store all of the current candidates, along with the amount of weighted votes that they have received so far.
- `remove_reps_db`: this database stores representatives that have issued `StopRepresenting` this epoch and will be removed from `representatives_db` when the epoch block is post-committed (see execution concept)

- `remove_candidates_db` : stores candidates that have issued `RenounceCandidacy` this epoch and will be removed from `candidacy_db` when the epoch block is post-committed (see execution concept)
- `leading_candidates_db` : stores top 8 candidates based on current election votes

Votes and Vote Weighting

Each epoch we are electing `N/L` delegates, where `N` is the total number of delegates and `L` is the term length. `ElectionVote` can contain votes for up to `N/L` different candidates. The votes are represented like a map from candidates to number of votes. In the diagram below, assuming `N/L == 8` , `ElectionVote1` , `ElectionVote2` and `ElectionVote3` are valid, while the others are not. Any `ElectionVote` that lists an account that is not an active candidate is also invalid.

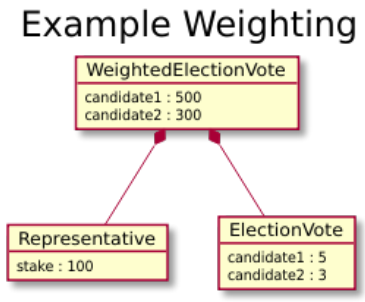


When a representative issues an `ElectionVote` , the vote is weighted by that representative's stake. So if a representative `R` has stake `s` , and issues an election vote with `x` votes for a candidate `C` , `C` receives `s * x` weighted votes from `R` .

Below is pseudocode for how a single `ElectionVote` is weighted and processed.

```
weightVoteAndAddToDb(ElectionVote v)
    validate v
    stake = sender's stake
    for(Candidate c in v)
        weighted_vote = (num_votes for c in v) * stake
        add weighted_vote to current total votes for c in candidate_db
```

Below is an example representative, a vote that they cast and the resulting weighted vote. Note, WeightedElectionVote is not an actual class, but simply a concept used here to illustrate weighting.



Delegate Voting Power

During consensus, each delegate has a specific voting power, fixed for the entire epoch, that is proportional to the number of votes they received during elections. The function is first stated in english, and then specific pseudocode. We are capping voting power at 1/8 of total voting power, and in the rare case a delegate received 0 votes but was elected still, we give them 1 vote for free

English:

```
if a delegate has 0 votes, give it 1 vote

pick an unprocessed delegate
if delegate has greater than cap votes
    redistribute excess votes to unprocessed delegates, proportional to number of votes those delegates hav
mark delegate as processed
repeat until all delegates are processed
```

Pseudocode:

```
votes_i = votes delegate i received when elected, or 1 if delegate received 0 votes
total_votes = sum of all votes_i
votes_remaining = total_votes
cap = total_votes * 1/8
for each votes_i in sorted order:
    if votes_i > cap:
        excess = votes_i - cap
        votes_i = cap
        votes_remaining -= votes_i
        for each votes_j not yet processed:
            votes_j_ratio = votes_j / votes_remaining
            votes_j += excess * votes_j_ratio
        votes_remaining += excess
    votes_i.processed = true
```

This function ensures no delegate receives more than 1/8 voting power, every delegate has greater than 0 voting power, and, if two delegates have less than 1/8 voting power after redistribution, their voting power relative to each other is the same before and after redistribution. When giving 1 vote as a freebie, that delegate is receiving at most 1/32 of voting power, since every other delegate will also have at least 1 vote as well. To change the cap, simply replace 1/8 with a different cap ratio.

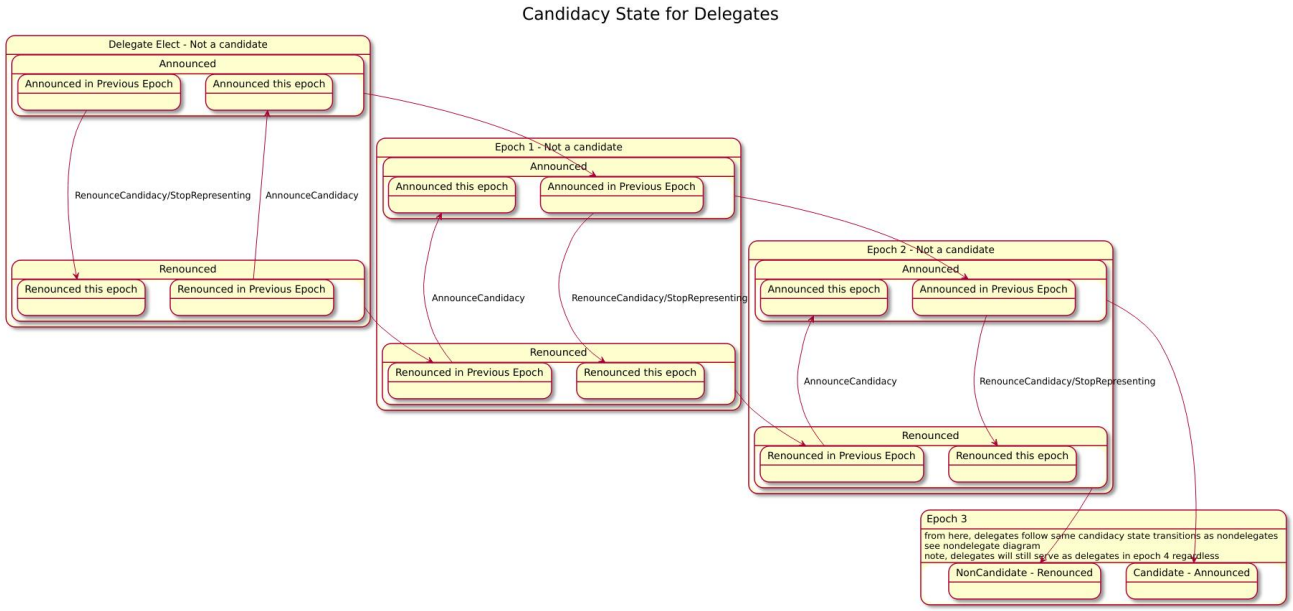
Staking

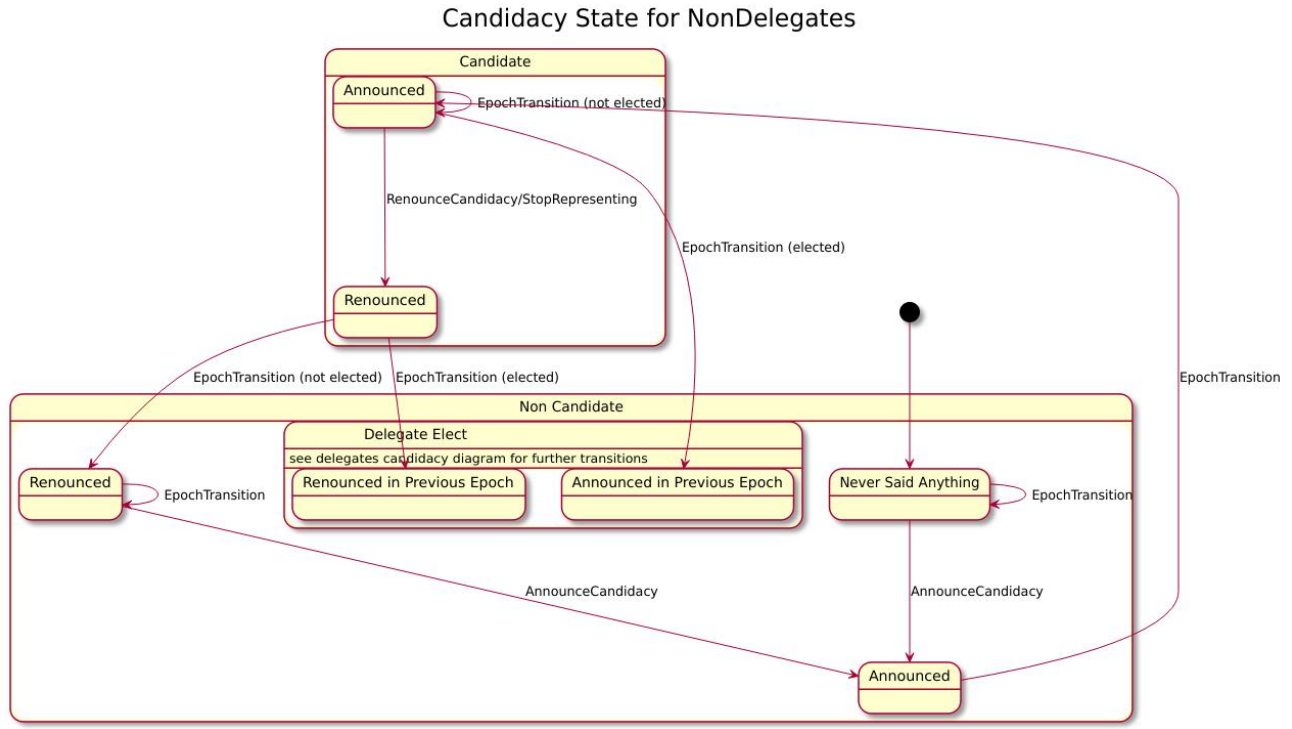
To be completed

Candidacy State

Below is a state diagram detailing the candidacy state transitions. Initially, an account starts in the `Never Said Anything` state for non-delegates. `AnnounceCandidacy` transitions the account to `Announced`, but the account is not a candidate until after epoch transition. After epoch transition, the candidate remains a candidate through subsequent epochs unless either: a) the candidate is elected or b) the candidate issues `RenounceCandidacy` or `StopRepresenting`, in which case they will be removed from the candidate list the epoch after they issue `RenounceCandidacy` or `StopRepresenting`. If a candidate is elected, they will transition to the `Delegate Elect` state in the Candidacy State for Delegates diagram, retaining their `Announced` or `Renounced` state. As a delegate or delegate elect, an account may change their candidacy state, but an account is not a candidate that can receive votes until their 3rd epoch as an active delegate. Note, a candidate is a delegate elect in the epoch after they are elected, and doesn't begin serving as a delegate until 2 epochs after they were elected (Epoch 1 in the diagram). Beginning in epoch 3, delegates become active candidates that can receive votes and follow the same candidacy state transitions as non-delegates. If a delegate is in the `Announced` state immediately prior to transitioning to epoch 3, then the delegate is an active candidate in epoch 3.

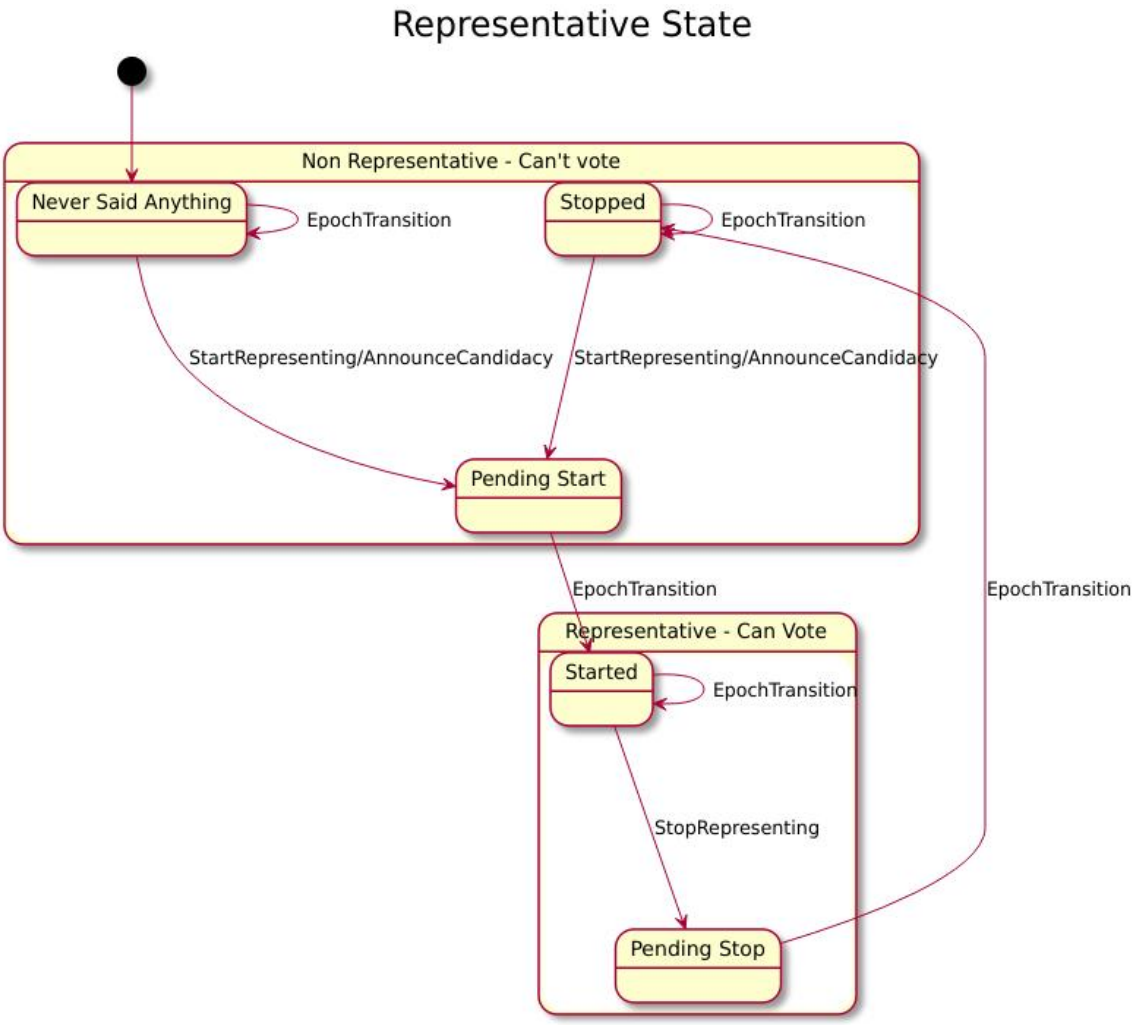
Note, an account can only issue one of `AnnounceCandidacy`, `RenounceCandidacy` or `StopRepresenting` per epoch, which means candidacy state can only change once per epoch.





Representative State

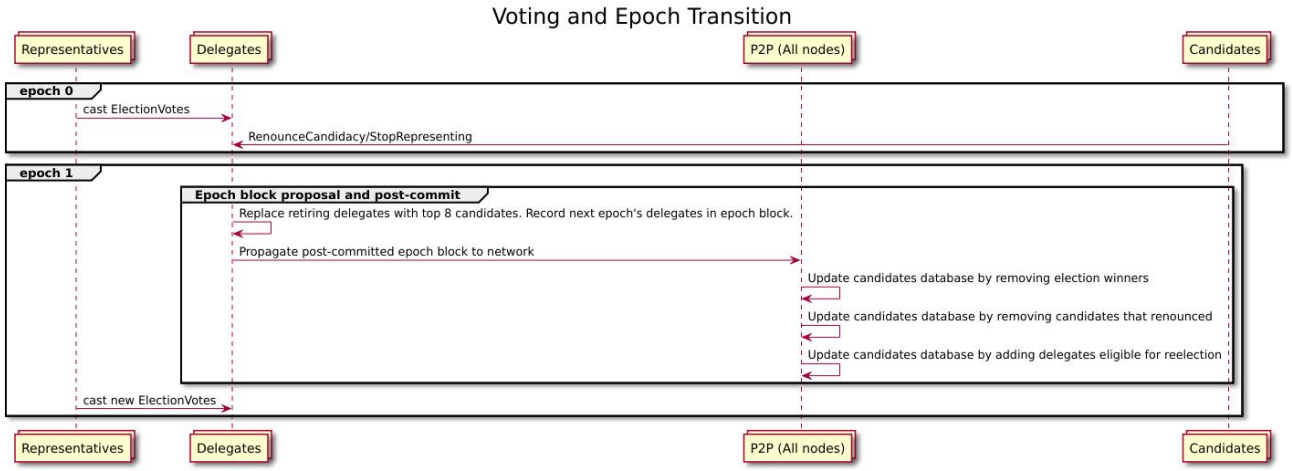
Below is a state diagram detailing the representative state transitions. The representative state determines whether an account can vote in elections. Accounts start in the `Never Said Anything` state of Non Representatives. To become a representative, accounts issue either `StartRepresenting` or `AnnounceCandidacy`, whereas the latter makes the account a rep and a candidate simultaneously. Each of these commands includes how much to stake. Once an account issues either of these commands, the account enters the `Pending Start` state, but is not yet a representative and cannot vote until after epoch transition. Note, an account can only issue one of `StartRepresenting`, `AnnounceCandidacy` or `StopRepresenting` per epoch, which means representative state can only change once per epoch.

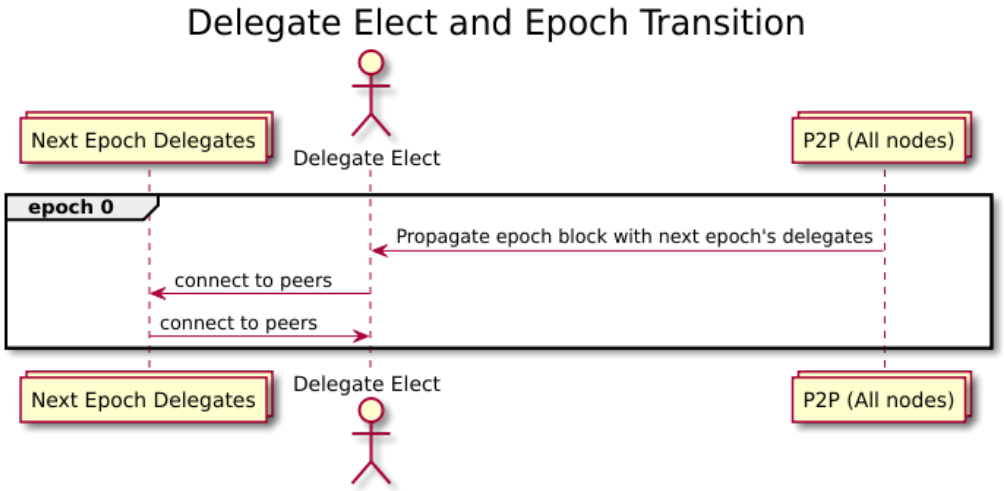


Scenarios and Sequence Diagrams

The below 2 diagrams detail election events that occur surrounding epoch block proposal and post-commit, and serve to transition the elections subsystem to the next epoch. During an epoch, the amount of votes received by each candidate is stored in `candidacy_db`, and the top 8 candidates are stored in `leading_candidates_db`. When the epoch ends, the elections dead period begins and voting ceases. When an epoch block is being built or validated, delegates read from `leading_candidates_db` to determine the candidates that won the election. The 8 retiring delegates are replaced with the 8 election winners. The list of delegates for the next epoch, which includes the election winners, is recorded in the epoch block. The epoch block is post-committed and distributed to the network over P2P. All nodes update their candidate databases upon receiving the epoch block, the dead period ends and elections begin for the new epoch. The nodes update the candidate database by removing delegate elects, removing candidates that issued `RenounceCandidacy` or `StopRepresenting`, and adding any delegates eligible for reelection. Delegates are eligible for reelection if they are in their second to last epoch and have not most recently renounced their candidacy, or if they are in their last epoch and are not delegate elects. See the timeline section and candidacy state section for more details on when delegates are eligible for reelection.

Candidates that won the election, who are now delegate elects, learn that they won the election when the epoch block is propagated to them, and will connect to their peers (the next epoch's delegates) at the end of the epoch.





The below diagram shows a representative announcing candidacy. Representative 1 submits `AnnounceCandidacy` in epoch 0, which all nodes hear about through P2P. The request makes Representative 1 a candidate for any subsequent epochs. In epoch 1, multiple representatives vote for Representative 1. A current tally of the amount of weighted votes that Representative 1 has received so far, recorded in `candidacy_db` and possibly `leading_candidates_db`, is updated after each `ElectionVote` is post-committed. At the beginning of Epoch 2, the election results are calculated. See the above sequence diagrams pertaining to epoch block persistence and transitioning the elections subsystem to the next epoch.