Consensus

Introduction

Logos builds on the best components of existing cryptocurrency networks while introducing innovations that enable its primary goals: practical scalability, fully aligned incentive structure, and security in the presence of Byzantine faults. Its base data structure is the chain mesh, a directed acyclic graph (DAG) consisting of a dedicated chain of transactions for each account. Transactions are processed in parallel rather than in blocks, and consensus is achieved via Axios, a delegated Practical Byzantine Fault Tolerance (dPBFT) algorithm that provides both liveness and safety as long as more than ⅔ of nodes by holdings are honest.

Axios is a PBFT-derived algorithm that allows for large, dynamic node sets to achieve consensus on transactions, votes, and epoch blocks. Each instance of PBFT requires a single leader called the primary to propose transactions and manage feedback from the other validators.

Axios has three primary phases to reach consensus:

- **Pre-Prepare:** the primary announces a network operation to be considered for consensus, signed by the primary.
- **Prepare:** every node checks the validity of the operation and send a signed prepare message to the primary if it agrees the operation is safe and valid. The leader waits for prepare messages from more than *2/3* n of nodes to construct a post-prepare message containing the multisignature of the agreeing nodes along with the corresponding bitmap identifying those nodes.
- **Commit:** Upon receiving proof that more than *2/3 n* nodes signed prepare messages for the operation (via the post-prepare message), a node sends a signed commit message to the primary. The leader waits for commit messages from more than *2/3 n* of nodes to construct a post-commit message containing the multisignature of the committing nodes along with the corresponding bitmap identifying those nodes. The post-commit is then disseminated across then network as proof that consensus has been reached, and all nodes update their ledgers accordingly.

Overview

At a very high level, the consensus module essentially boils down to three main components:

- `PrimaryDelegate`

- `ConsensusManager`
- `ConsensusConnection`

PrimaryDelegate

The PrimaryDelegate is responsible for receiving and validating consensus messages sent by backup delegates. It aggregates messages sent by backups and upon receiving approving responses from more than *2/3* of backups, responds with a message containing the aggregated signature of the responding delegates.

ConsensusManager

The ConsensusManager acts as the primary interface for the Consensus module and handles the bookkeeping done before consensus is initiated and after consensus is reached by the delegates.

ConsensusConnection

The ConsensusConnection maintains a connection to a single remote peer. This class is used by the PrimaryDelegate to send messages to remote delegates, and also receives messages sent by remote delegates.

Execution Concept

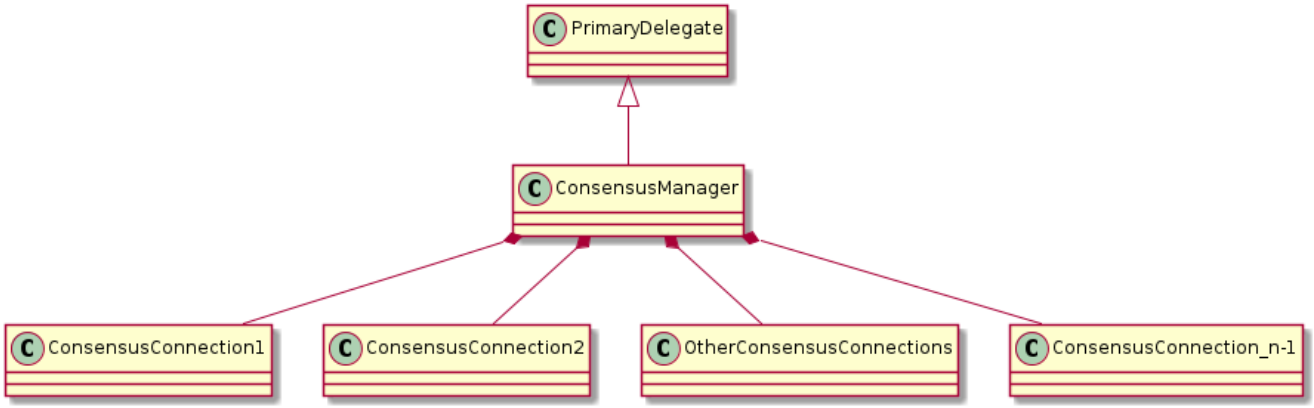
Threading

The consensus module executes on top of the Boost asio threadpool. This threadpool is created by spawning threads at the start of the application and calling `service.run()` on each thread. The module then responds to IO events such as transaction requests from clients received from the network, or messages from other delegates.

Axios Concurrently

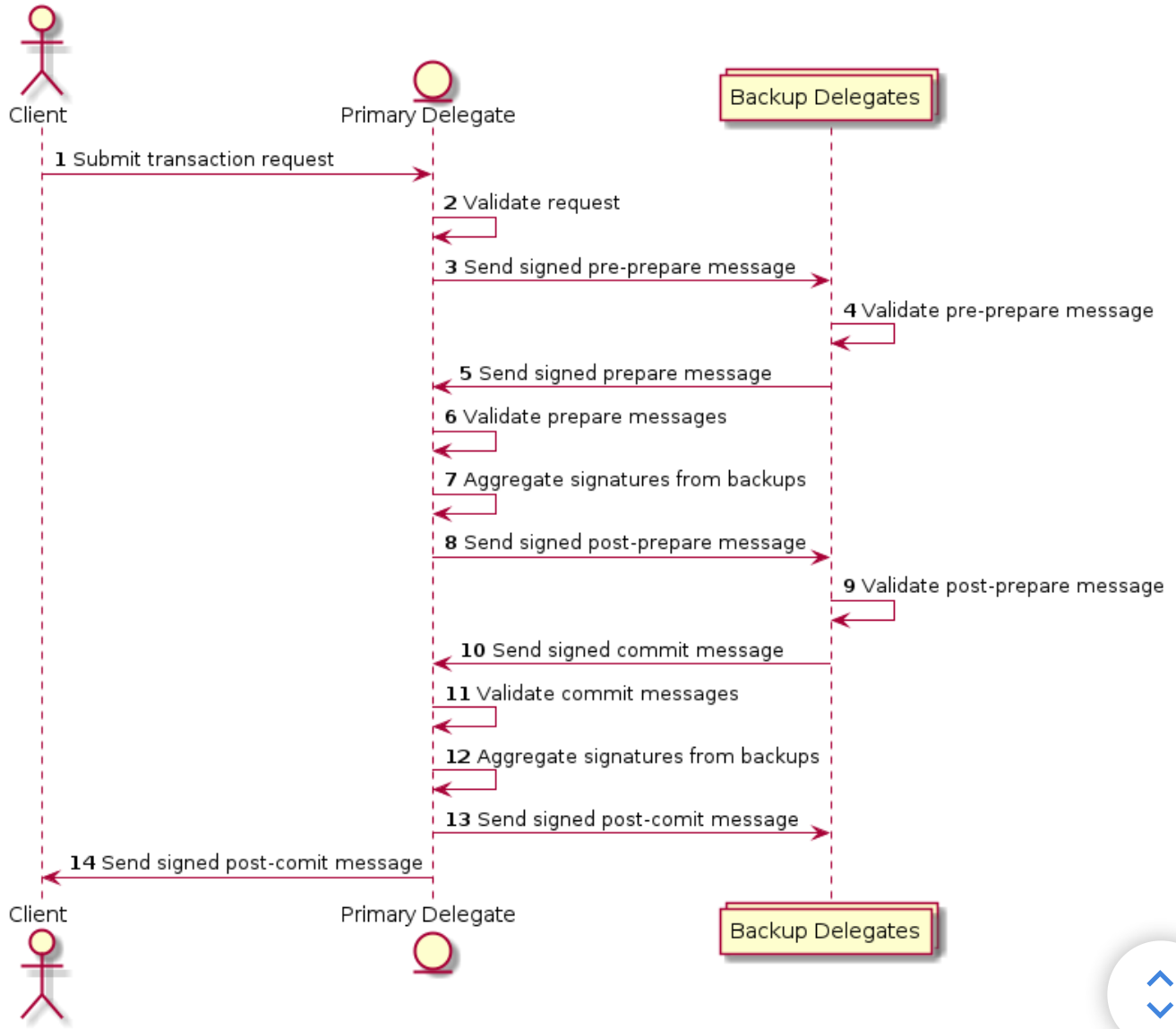
For each type of consensus (BatchStateBlock, MicroBlock, EpochBlock, etc.), a delegate node will have one instance of the `PrimaryDelegate` class and *n-1* instances of the `ConsensusConnection` class (where *n* is the number of delegates in the distributed system). A `PrimaryDelegate` instance can only lead a single instance of consensus at a time. The `PrimaryDelegate` leading consensus and the `ConsensusConnection` instances serving as backups occur in parallel.

Class Overview



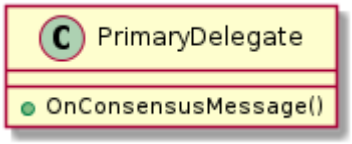
Consensus Sequence

Please note that the designation of one node as a primary and the rest as backups is determined by the client. The recipient of the client's transaction request will be the primary for with respect to that request, and the others will serve as backups.

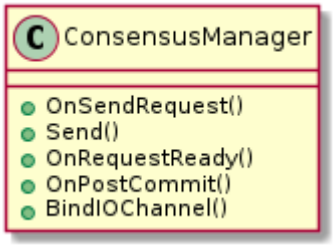


Interfaces

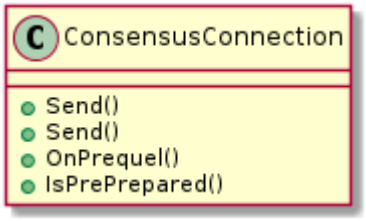
The PrimaryDelegate class presents a public interface to the ConsensusConnection class:



The ConsensusManager class presents a public interface to several classes including but not limited to ConsensusConnection and PrimaryDelegate:

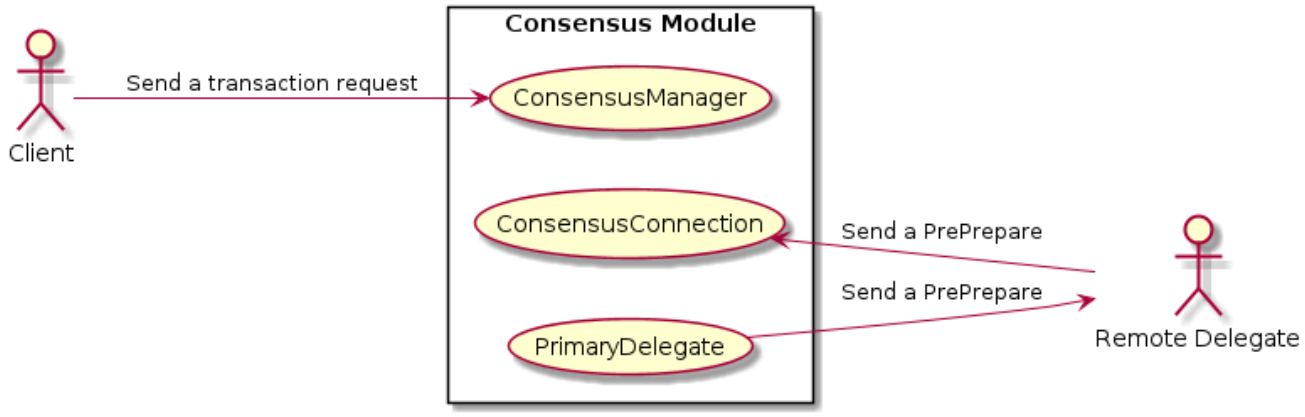


The ConsensusConnection class presents a public interface to the ConsensusManager and ConsensusNetIO classes:



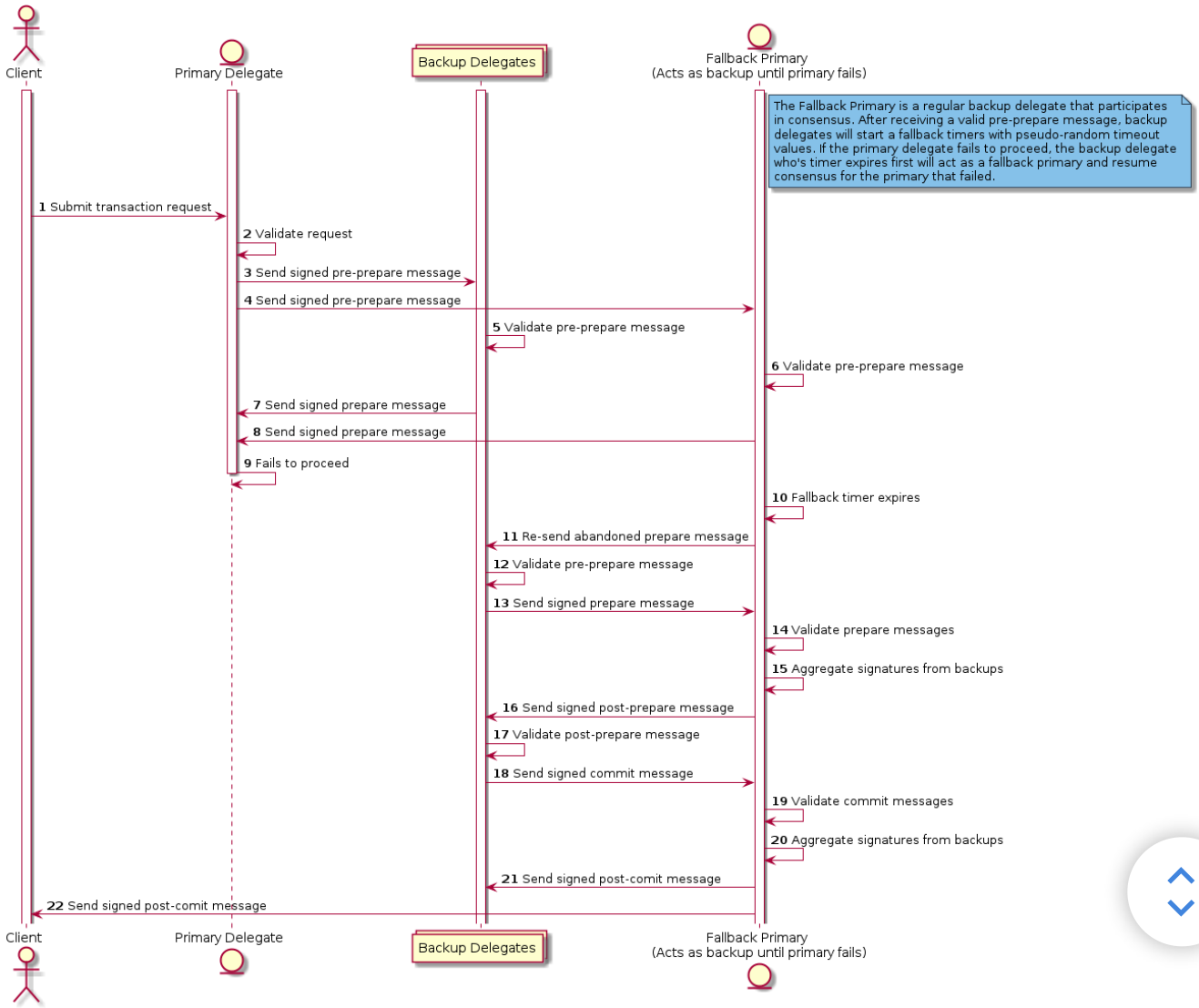
Use Case Diagram

The Consensus Module exposes interfaces to remote clients and remote delegates:



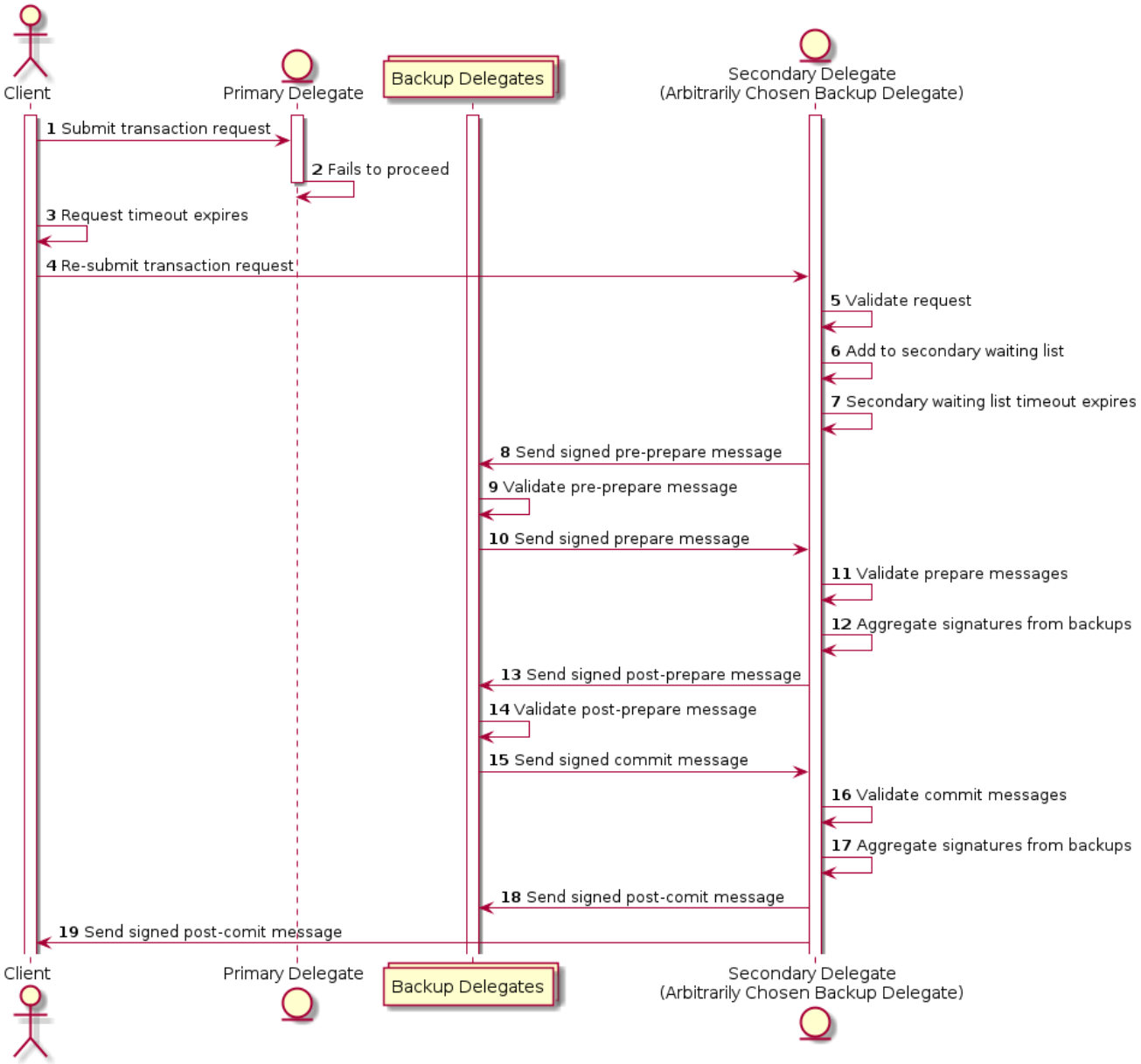
Fallback Consensus

In fallback consensus, a backup delegate resumes a consensus session abandoned by a faulty primary.



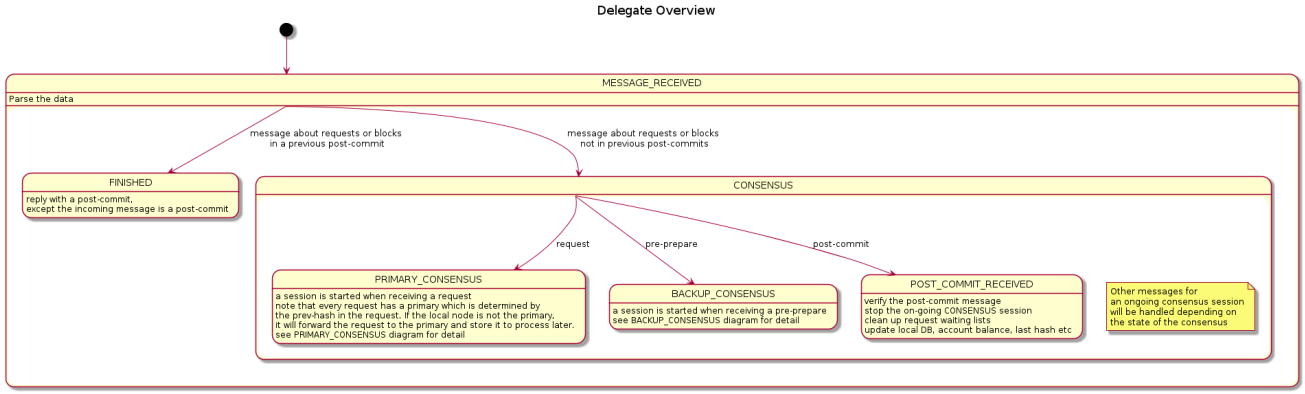
Secondary Waiting List

A transaction request received by a delegate that is not the designated primary for the request will be stored in that delegate's waiting list and a timer will be started. If the timer for this request expires before the request is post-comitted by another delegate, the request will be promoted to the primary waiting list and proposed by the receiving delegate.

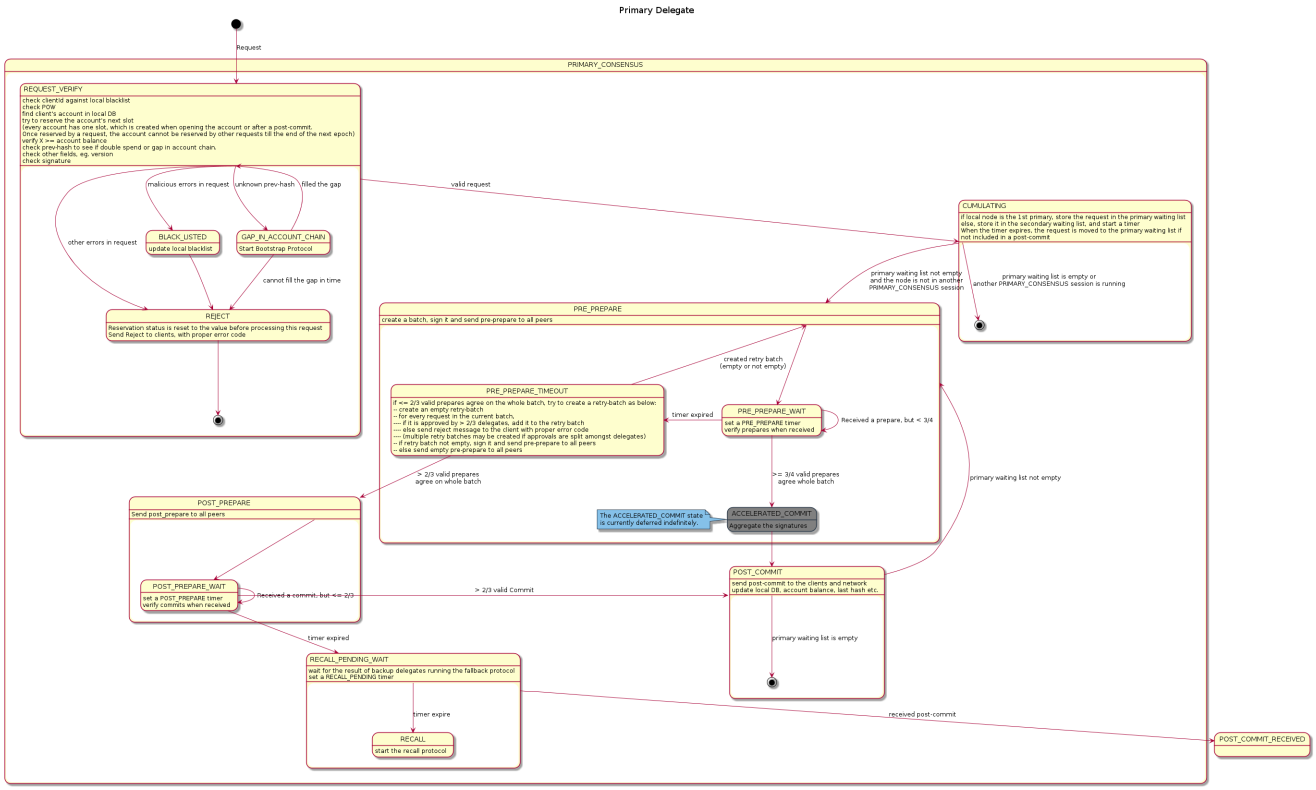


State Diagrams

Delegate Overview

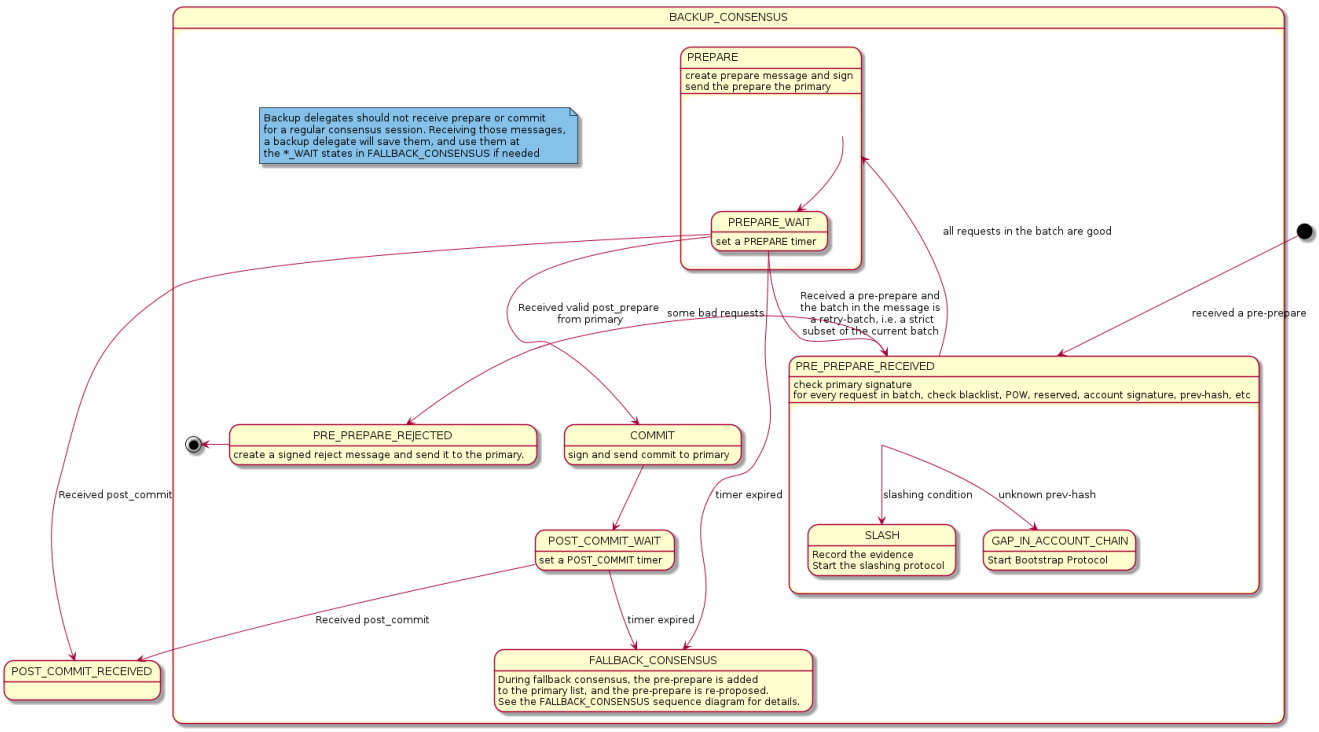


Primary Delegate



Backup Delegate

Backup Delegate



➔ Next to **Tokens**