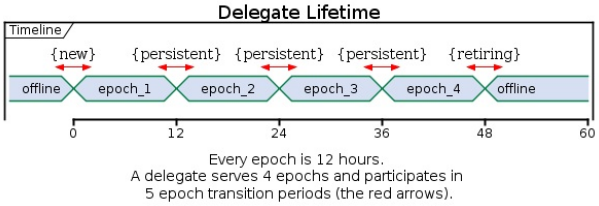
Epoch Transition Component

Overview

The time in the Logos network is divided into distinct, successive time periods (12 hours) called epochs. Every epoch has a set of 32 delegates. During the period when the network transitions from one epoch to the next one, 8 oldest delegates will be replaced by 8 new delegates. Every delegate serves 4 epochs and participates in 5 transition periods (if the delegate is not re-elected, also assuming no recall happened), as the figure below shows. A delegate is “new” in its first transition period, then is “persistent” in the following three transition periods, and finally is “retiring” in its last transition period before it goes offline. This document specifies the design of the three types of transition periods.

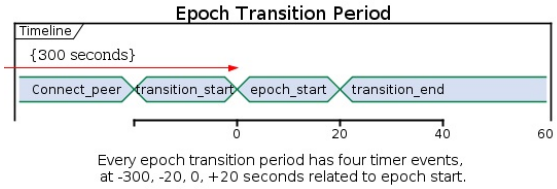


The Epoch Transition component is responsible for all epoch transition related data structure, algorithms, and operations, this includes:

- Maintain timers for proposing epoch transitions.
- Switch to the new epoch number at the right time, so that there will not be any circular dependency among different kinds of blocks.
- Create an EpochManager and all the objects it contains every epoch, such as the ConsensusManagers and the ConsensusNetIOManager. During an epoch transition period, a node could have two EpochManagers. Only one of them can propose pre-prepare messages.
- Make TCP connections to peers, and disconnect from them when needed. In the current design, during the transition period, two persistent delegates will have two TCP connections between them. The protocol must be able to distinguish a reconnection attempt from a new connection between the two peers.

Execution Concept

The Epoch Transition Component uses timers as well as network messages to trigger events of the epoch transitions. The figure and the table below show the four timer events.



Event	Time
Connect_peer	300 seconds before epoch start.
Transition_start	20 seconds before epoch start.
Epoch_start	epoch start time as agreed by the network.
Transition_end	20 seconds after epoch start.

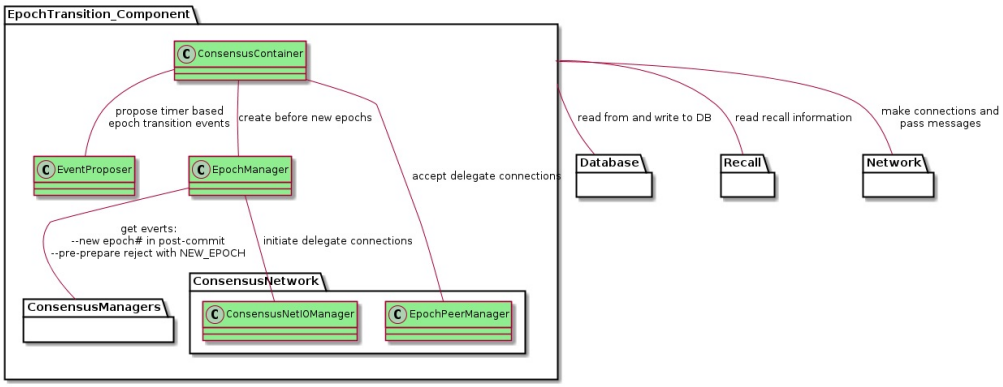
Note that the 20 seconds are introduced to accomendate processing delay, network delay, and clock drift.

The connect_peer is the first event, when it is triggered, one of the boost thread pool thread will call the proper callback function to start make TCP connections to peer delegates of the coming epoch. The delegates has quite sometime to make the connections before the transition_start. At the transition_start and epoch_start, delegates have different behavour depending on if they are new, persistent, or retiring. At the last event transition_end, delegates release unneeded resources and network connections.

There are also two network message triggered events that could happen before Epoch_start: (1) a post-commit message with the new epoch number is received and (2) a pre-prepare reject message with error code NEW_EPOCH is received. The rest of the document will detail how the events are processed.

Interfaces

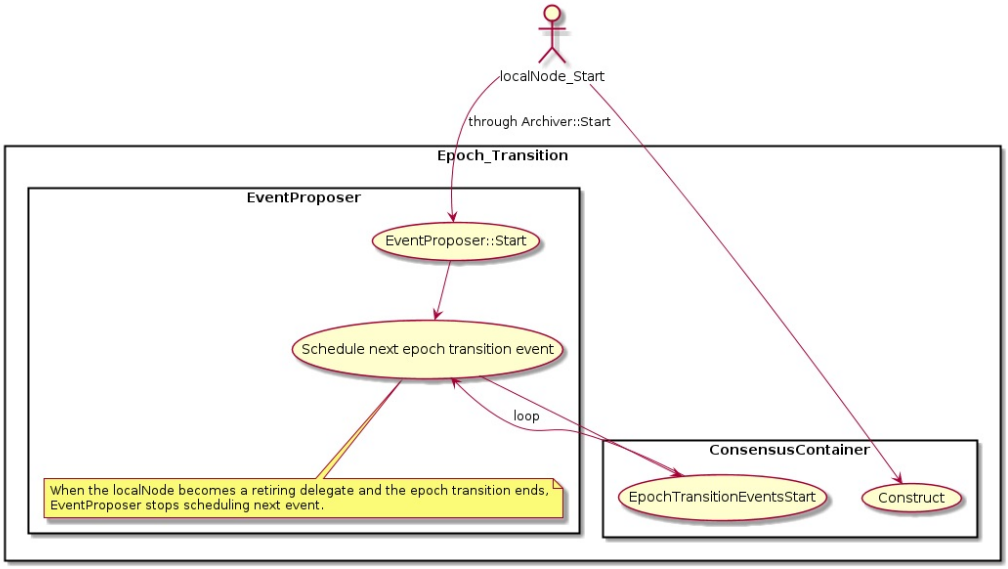
Please note that most of the classes involved in the epoch transition process have other responsibilities. For example, a ConsensusManager’s main purpose is to handle consensus related tasks. In this document, we will only describe epoch transition related design. The figure below shows how the Epoch Transition Component interfaces with other component, as well as its main classes and sub-components.



Note that in the implementation, the EventProposer is contained in an Archiver, as shown in the class diagram in the Classes and Sequence Diagrams section.

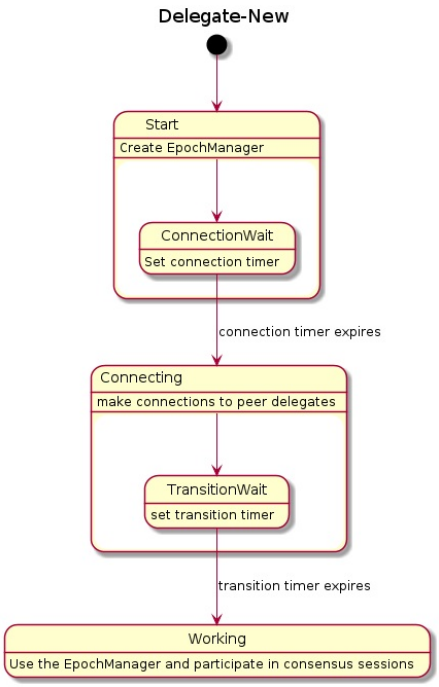
Use Case Diagrams

When a delegate node starts, it creates a ConsensusContainer and starts the EventProposer (through Archiver::Start in the code). The EventProposer schedules a timer based epoch transition event. When the timer expires, a callback in the ConsensusContainer will handle the event and next transition event will be scheduled. Eventually, the delegate node retires and the EventProposer and ConsensusContainer become idle till the node is re-elected as a delegate again.

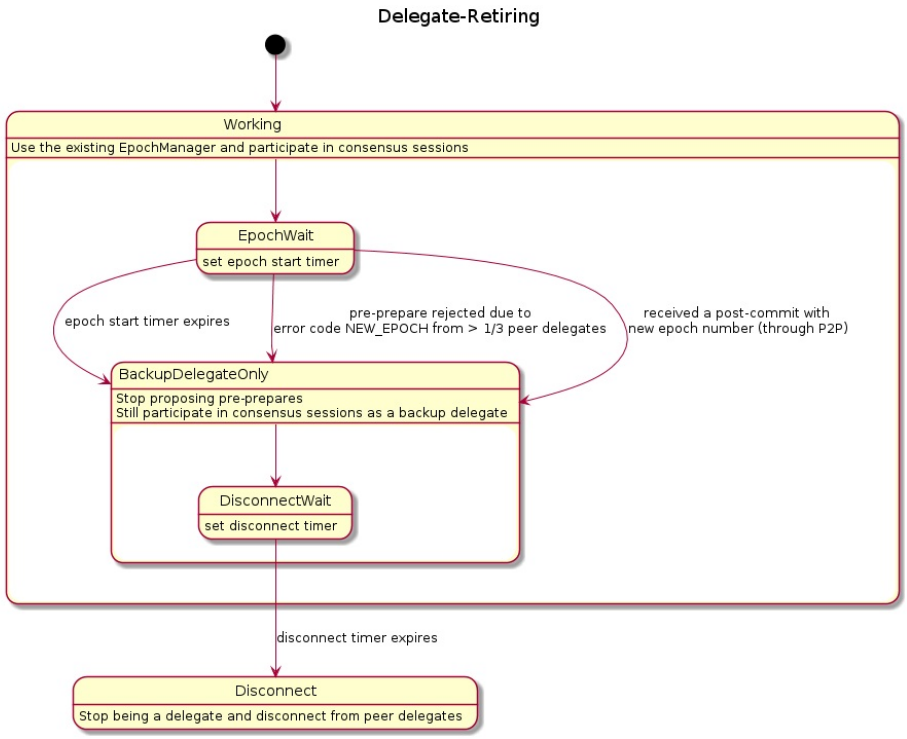


State Diagrams

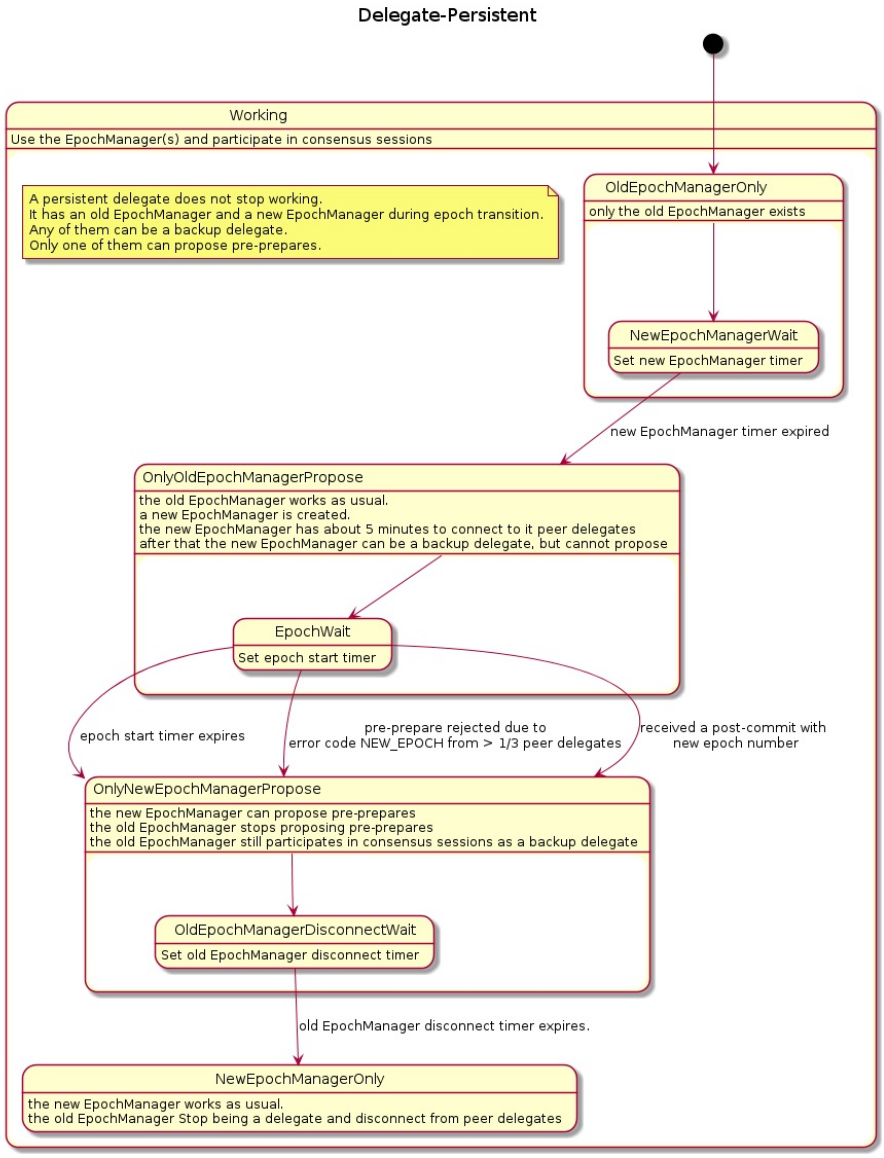
Every delegate participates in 3 types of epoch transitions, namely new, persistent, and retiring, as shown in the state diagrams below.



Being elected, a delegate follows the “Delegate-New” diagram and changes its states. After start, a EpochManager is created. With two timer based events, it connects with its peer delegates in the same epoch and then starts working.



A retiring delegate eventually goes to the “Disconnect” state from “Working” state at the end of its epochs, as shown in the “Delegate-Retiring” diagram. An important state transition to note is from “EpochWait” to “BackupDelegateOnly” while “Working”.

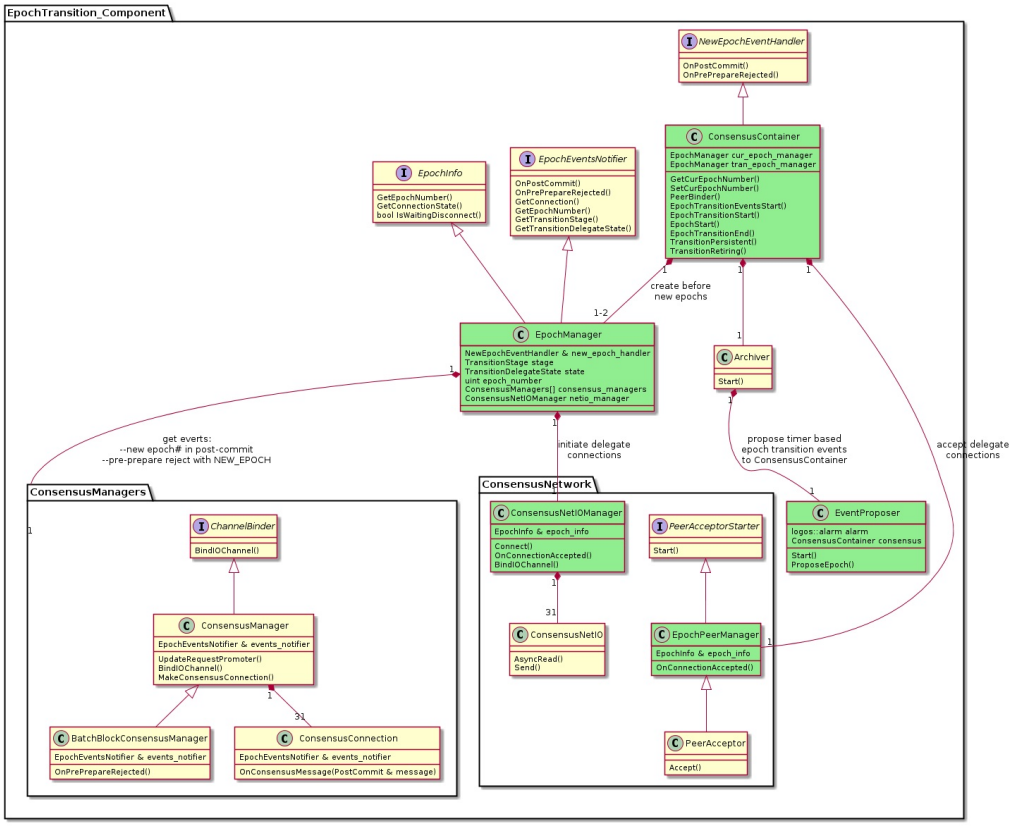


As a persistent delegate in both the old epoch and new epoch, it does not stop working. During the epoch transition period, it has two EpochManagers, the OldEpochManager and the NewEpochManager. The NewEpochManager gradually takes over the responsibility of participating in the consensus sessions from the OldEpochManager. Please note that the three conditions that transition the state from OnlyOldEpochManagerPropose to OnlyNewEpochManagerPropose.

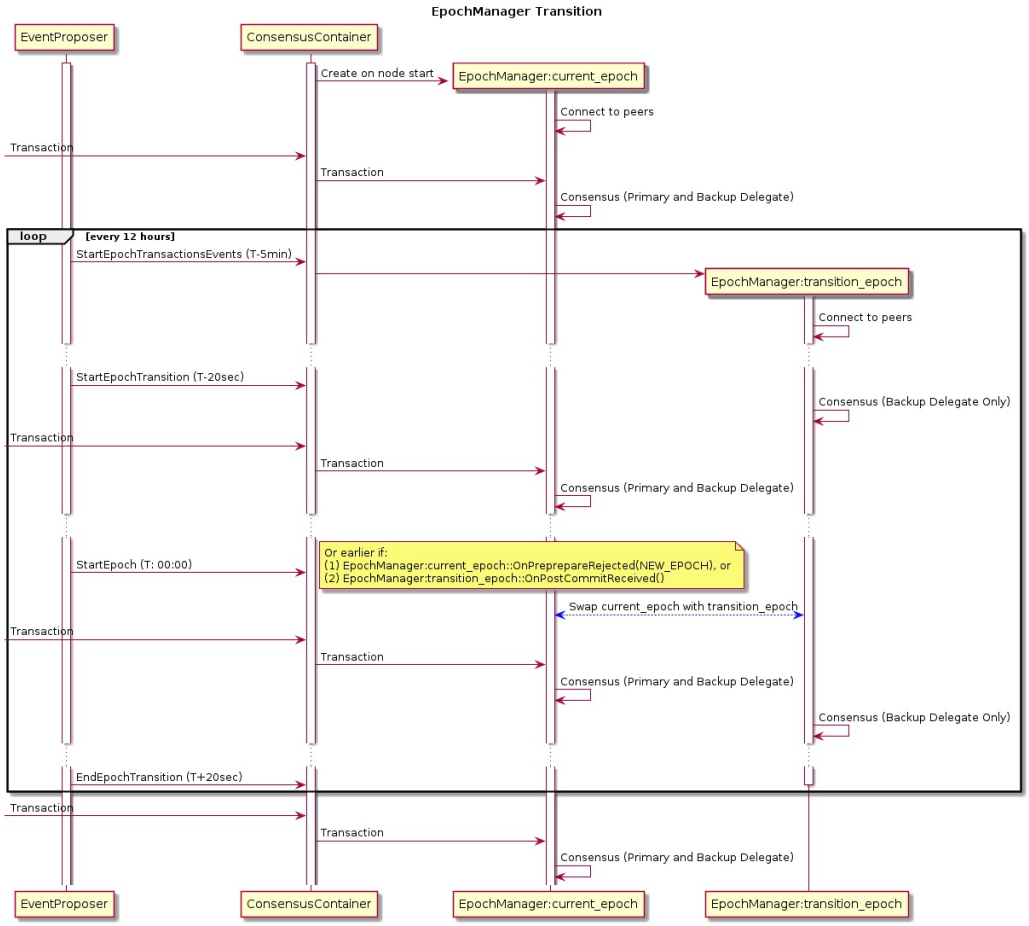
Classes and Sequence Diagrams

The “Epoch Transition Classes” diagram shows the classes involved in the epoch transition process. Most of the classes have other responsibilities. In this section, we focus on their epoch transition related functionalities, and the relationship among the classes. Two most interesting aspects of epoch transition, namely “EpochManager Transition” and “Epoch transition network connection”, are shown in the sequence diagrams.

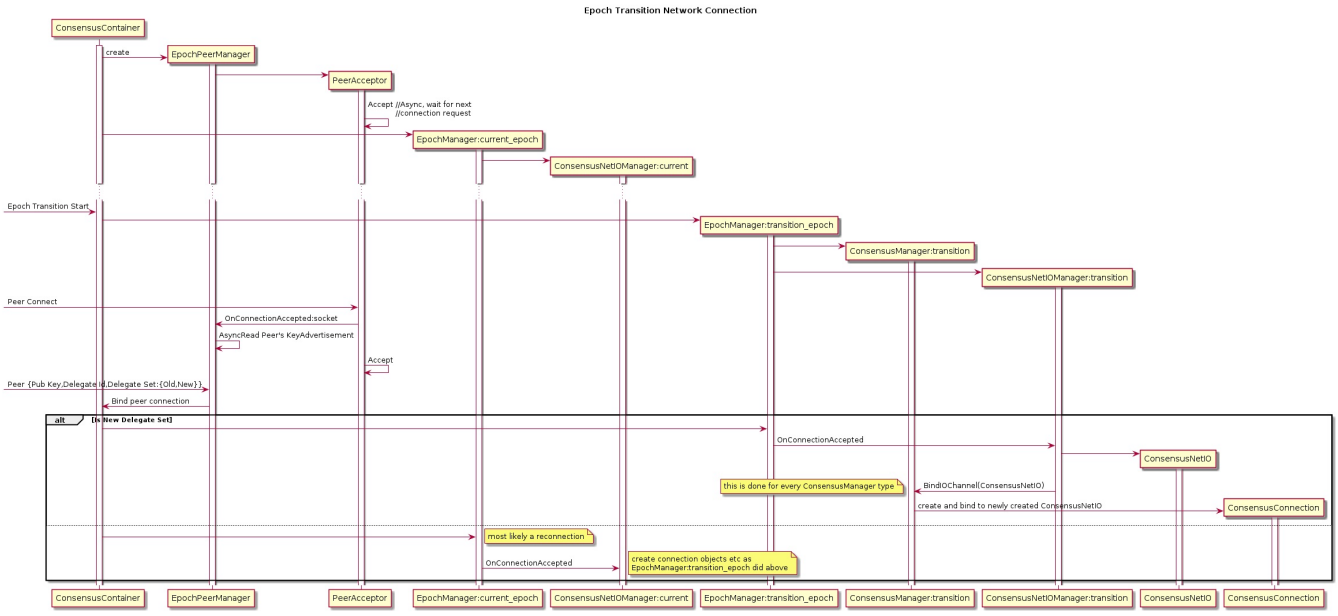
Epoch Transition Classes



The ConsensusContainer manages the epoch transition process. It contains a EpochPeerManager which is a TCP server that accepts peer delegates' connection requests. The ConsensusContainer also contains a EpochEventProposer that provides timer base events. The first timer will trigger the callback to EpochTransitionEventsStart(), in which a new EpochManager will be created, which will create a ConsensusNetIOManager. Once created, the ConsensusNetIOManager will make TCP connections to peer delegates. (Two delegates determine the TCP client-server relationship between them basing on their delegate_indices defined in the EpochBlock. The one with lower index is the server.) The EpochManager will also create ConsensusManagers and related objects. Those objects provide two important network events, namely (1)new epoch# in post-commit and (2) pre-prepare reject with error code NEW_EPOCH, to the ConsensusContainer through the EpochManager.



As the “EpochManager Transition” diagram shows, client transactions are sent to the ConsensusContainer and processed by the current EpochManager. During the epoch transition period, a new EpochManager transition_epoch is created and swaps with the current_epoch at the right time. After the swap, when epoch transition period ends, the transition_epoch (which was the current_epoch before the swap) is distructed. (Note that in the code, when the two EpochManagers swap with each other, the new current_epoch also takes over the secondary waiting list by SecondaryRequestHandler::UpdateRequestPromoter.)



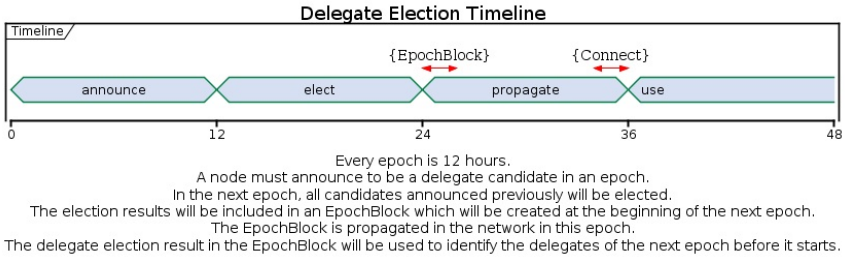
The “Epoch Transition Network Connection” diagram shows how a persistent delegate accepts TCP connection requests and eventually creates a ConsensusConnection object which will be used by ConsensusManagers to send and receive network messages. As a persistent delegate, a PeerAcceptor is already started and accepting connection requests. It must accept connection requests to the new epoch and accepts re-connection requests to the old epoch. The two different kinds of requests are distinguished by the first protocol message (ConnectedClientIds) sent through the underlining TCP connection once it is established. After that, the OnConnectionAccepted() function of ConsensusNetIOManager is called to create a ConsensusNetIO object. The ConsensusManagers are notified to create a ConsensusConnection object which binds to the newly created ConsensusNetIO object.

Resources

This component use the network and database.

Author defined Topics

Delegate Election Timeline



Every epoch is 12 hours.
A node must announce to be a delegate candidate in an epoch.
In the next epoch, all candidates announced previously will be elected.
The election results will be included in an EpochBlock which will be created at the beginning of the next epoch.
The EpochBlock is propagated in the network in this epoch.
The delegate election result in the EpochBlock will be used to identify the delegates of the next epoch before it starts.