Introduction

This section introduces and outlines the Charon[kharon] middleware, Obol's implementation of DVT. Please see the key concepts section as background and context.

What is Charon?

Charon is a GoLang-based, HTTP middleware built by Obol to enable any existing Ethereum validator clients to operate together as part of a distributed validator.

Charon sits as a middleware between a normal validating client and its connected beacon node, intercepting and proxying API traffic. Multiple Charon clients are configured to communicate together to come to consensus on validator duties and behave as a single unified proof-of-stake validator together. The nodes form a cluster that isbyzantine-fault tolerant and continues to progress assuming a supermajority of working/honest nodes is met.

Charon Architecture

Charon is an Ethereum proof of stake distributed validator (DV) client. Like any validator client, its main purpose is to perform validation duties for the Beacon Chain, primarily attestations and block proposals. The beacon client handles a lot of the heavy lifting, leaving the validator client to focus on fetching duty data, signing that data, and submitting it back to the beacon client.

Charon is designed as a generic event-driven workflow with different components coordinating to perform validation duties. All duties follow the same flow, the only difference being the signed data. The workflow can be divided into phases consisting of one or more components:

Determinewhen

duties need to be performed

The beacon chain is divided intoslots andepochs , which divides it into deterministically fixed-size time chunks. The first step is to determine when (which slot/epoch) duties need to be performed. This is done by thescheduler component. It queries the beacon node to detect which validators defined in the cluster lock are active, and what duties they need to perform for the upcoming epoch and slots. When such a slot starts, thescheduler emits an event indicating which validator needs to perform what duty.

Fetch and come to consensus onwhat

data to sign

A DV cluster consists of multiple operators each provided with one of the M-of-N threshold BLS private key shares per validator. The key shares are imported into the validator clients which produce partial signatures. Charon threshold aggregates these partial signatures before broadcasting them to the Beacon Chain.But to threshold aggregate partial signatures, each validator must sign the same data. The cluster must therefore coordinate and come to a consensus on what data to sign.

Fetcher fetches the unsigned duty data from the beacon node upon receiving an event fromScheduler . For attestations, this is the unsigned attestation, for block proposals, this is the unsigned block.

TheConsensus component listens to events from Fetcher and starts aQBFT consensus game with the other Charon nodes in the cluster for that specific duty and slot. When consensus is reached, the resulting unsigned duty data is stored in theDutyDB .

Wait

for the VC to sign

Charon is amiddleware distributed validator client. That means Charon doesn’t have access to the validator private key shares and cannot sign anything on demand. Instead, operators import the key shares into industry-standard validator clients (VC) that are configured to connect to their local Charon client instead of their local Beacon node directly.

Charon, therefore, serves theEthereum Beacon Node API from theValidatorAPI component and intercepts some endpoints while proxying other endpoints directly to the upstream Beacon node.

The VC queries theValidatorAPI for unsigned data which is retrieved from theDutyDB . It then signs it and submits it back to theValidatorAPI which stores it in thePartialSignatureDB .

Share

partial signatures

ThePartialSignatureDB stores the partially signed data submitted by the local Charon client's VC. But it also stores all the partial signatures submitted by the VCs of other peers in the cluster. This is achieved by thePartialSignatureExchange component that exchanges partial signatures between all peers in the cluster. All charon clients, therefore, store all partial signatures the cluster generates.

Threshold Aggregate

partial signatures

TheSignatureAggregator is invoked as soon as sufficient (any M of N) partial signatures are stored in thePartialSignatureDB . It performs BLS threshold aggregation of the partial signatures resulting in a final signature that is valid for the beacon chain.

Broadcast

final signature

Finally, theBroadcaster component broadcasts the final threshold aggregated signature to the Beacon client, thereby completing the duty.

Ports

The following is an outline of the services that can be exposed by charon.

- :3600
-
    - The validator REST API. This is the port that serves the consensus layer'sbeacon node API
- . This is the port validator clients should talk to instead of their standard consensus client REST API port. Charon subsequently proxies these requests to the upstream consensus client specified by--beacon-node-endpoints
- .
- :3610
-
    - Charon P2P port. This is the port that charon clients use to communicate with one another via TCP. This endpoint should be port-forwarded on your router and exposed publicly, preferably on a static IP address. This IP address should then be set on the charon run command with--p2p-external-ip
- orCHARON_P2P_EXTERNAL_IP
- .
- :3620
-
    - Monitoring port. This port hosts a webserver that serves prometheus metrics on/metrics
- , a readiness endpoint on/readyz
- and a liveness endpoint on/livez
- , and a pprof server on/debug/pprof
- . This port should not be exposed publicly.

Getting started

For more information on running charon, take a look at ouQuickstart Guides . Edit this page Previous Combine DV private key shares Next Distributed Key Generation