Build Decentralized Verifier Networks (DVNs)

This document contains a high level overview of how to implement and integrate a basic third party DVN into the LayerZero V2 protocol.

Fee Quoting, Collection, and Withdrawal

DVN owners should implement and deploy a DVN contract on every chain they want to support. The contract must implement theILayerZeroDVN interface, which specifies two functions:assignJob andgetFee .

interface

ILayerZeroDVN

{ struct

AssignJobParam

{ uint32 dstEid ; bytes packetHeader ; bytes32 payloadHash ; uint64 confirmations ; address sender ; }

function

assignJob (AssignJobParam calldata _param ,

bytes

calldata _options)

external

payable

returns

(uint256 fee) ;

function

getFee (uint32 _dstEid , uint64 _confirmations , address _sender , bytes

calldata _options)

external

view

returns

(uint256 fee) ; } FunctionName Type Description assignJob Payable Called as part of_lzSend . getFee View Typically called by applications before sending the packet to estimate fees.

If your DVN is responsible for a packet, the LayerZero Endpoint will call your DVN contract'sassignJob function.

Building a DVN

The DVN has one off-chain workflow:

1. The DVN first listens for thePacketSent
2. event:
3. PacketSent
4. (
5. bytes
6. encodedPacket
7. ,
8. bytes
9. options
10. ,
11. address
12. sendLibrary
13.)

The packet has the following structure:
struct
Packet
{
uint64
nonce
;
// the nonce of the message in the pathway
uint32
srcEid
;
// the source endpoint ID
address
sender
;
// the sender address
uint32
dstEid
;
// the destination endpoint ID
bytes32
receiver
;
// the receiving address
bytes32
guid
;
// a global unique identifier
bytes
message
;
// the message payload
}
The encoded packet can be deserialized with thePacketSerializer
and the option can be deserialized with theOptionSerializer
.
After thePacketSent
event, theDVNFeePaid
is how you know your DVN has been assigned to verify the packet'spayloadHash
.
DVNFeePaid
(
address
[
]
requiredDVNs
,
address
[
]
optionalDVNs
,
uint256
[
]
fees
)
;
tip
TheDVNFeePaid
returns a list ofall
of the OApp's configured DVNs, so your workflow should filter your specific DVN address from the array to make sure your DVN has been paid.
After receiving the fee, your DVN should query the address of the MessageLib on the destination chain:
getReceiveLibrary
(
_receiver
,

_dstEid
)
;
After your DVN has retrieved the receive MessageLib, you should read the MessageLib configuration from it. In the configuration is the required blockconfirmations
to wait before callingverify
on the destination chain.
function
getUlnConfig
(
address
_oapp
,
uint32
_remoteEid
)
public
view
returns
(
UlnConfig
memory
rtnConfig
)
;
This will return theUlnConfig
, which you can use to read the number ofconfirmations
:
struct
UlnConfig
{
uint64
confirmations
;
// ...
Your DVN should next do an idempotency check:
ULN
.
_verified
(
_dvn
,
_headerHash
,
_payloadHash
,
_requiredConfirmation
)
;
This returns a boolean value:

- If the state istrue
- , then your idempotency check indicates that you already verified this packet. You can terminate your DVN workflow.
- If the state isfalse
- , then you must callULN.verify
- :

ULN
.
_verify
(
_packetHeader
,
_payloadHash

142. ,
143. _confirmations
144.)
145. ;
146. tip
147. To know your workflow has successfully fulfilled its obligation, your DVN should perform an idempotency check at the end of the DVN workflow. Edit this page

Previous Integration Checklist Next Build Executors