

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 the `ILayerZeroDVN` interface, which specifies two functions: `assignJob` and `getFee`.

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
<code>assignJob</code>	Payable	Called as part of <code>_lzSend</code> .
<code>getFee</code>	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's `assignJob` function.

Building a DVN

The DVN has one off-chain workflow:

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

```

14. The packet has the following structure:
15. struct
16. Packet
17. {
18. uint64
19. nonce
20. ;
21. // the nonce of the message in the pathway
22. uint32
23. srcEid
24. ;
25. // the source endpoint ID
26. address
27. sender
28. ;
29. // the sender address
30. uint32
31. dstEid
32. ;
33. // the destination endpoint ID
34. bytes32
35. receiver
36. ;
37. // the receiving address
38. bytes32
39. guid
40. ;
41. // a global unique identifier
42. bytes
43. message
44. ;
45. // the message payload
46. }
47. The encoded packet can be deserialized with the PacketSerializer
48. and the option can be deserialized with the OptionSerializer
49. .
50. After thePacketSent
51. event, theDVNFeePaid
52. is how you know your DVN has been assigned to verify the packet'spayloadHash
53. .
54. DVNFeePaid
55. (
56. address
57. [
58. ]
59. requiredDVNs
60. ,
61. address
62. [
63. ]
64. optionalDVNs
65. ,
66. uint256
67. [
68. ]
69. fees
70. )
71. ;
72. tip
73. TheDVNFeePaid
74. returns a list of all
75. 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.
76. After receiving the fee, your DVN should query the address of the MessageLib on the destination chain:
77. getReceiveLibrary
78. (
79. _receiver
80. ,

```

```

81. _dstEid
82. )
83. ;
84. After your DVN has retrieved the receive MessageLib, you should read the MessageLib configuration from it. In the
    configuration is the required blockconfirmations
85. to wait before callingverify
86. on the destination chain.
87. function
88. getUlnConfig
89. (
90. address
91. _oapp
92. ,
93. uint32
94. _remoteEid
95. )
96. public
97. view
98. returns
99. (
100. UlnConfig
101. memory
102. rtnConfig
103. )
104. ;
105. This will return theUlnConfig
106. , which you can use to read the number ofconfirmations
107. :
108. struct
109. UlnConfig
110. {
111. uint64
112. confirmations
113. ;
114. // ...
115. Your DVN should next do an idempotency check:
116. ULN
117. .
118. _verified
119. (
120. _dvn
121. ,
122. _headerHash
123. ,
124. _payloadHash
125. ,
126. _requiredConfirmation
127. )
128. ;
129. This returns a boolean value:
130.
    ◦ If the state istrue
131.
    ◦ , then your idempotency check indicates that you already verified this packet. You can terminate your DVN
        workflow.
132.
    ◦ If the state isfalse
133.
    ◦ , then you must callULN.verify
134.
    ◦ :
135. ULN
136. .
137. _verify
138. (
139. _packetHeader
140. ,
141. _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](#)