Lifecycle of a Transaction

Transactions are constructed by users to express the intent of performing actions in the network. Once in the network, transactions are converted intoReceipts , which are messages exchanged between network nodes.

In this page we will explore the lifecycle of a transaction, from its creation to its final status.

Recommended Reading To dig deeper into transaction routing we recommend to read thenearcore documentation

Receipts & Finality

Lets walk through the lifecycle of a complex transaction, and see how it is processed by the network using blocks astime units .

Block #1: The Transaction Arrives

After a transaction arrives, the network takes one block to validate it and transform it into a singleReceipt that contains all theactions to be executed.

While creating theReceipt , thesigner gets NEAR deducted from its balance topay for the gas andany attached NEAR .

If thesigner andreceiver coincide - e.g. thesigner is adding a Key - theReceipt is immediately processed in this first block and the transaction is considered final.

Block #2: The Receipt is Processed

If thesigner andreceiver differs - e.g. thesigner transfers NEAR to thereceiver - theReceipt is processed in a second block.

During this process aFunctionCall could span across-contract calls , creating one or multiple newReceipts .

Block #3...: Function Calls

EachReceipt created from the function call take an additional block to be processed. Notice that, if thoseReceipts areFunctionCall they could spawn newReceipts and so on.

Final Block: Gas is Refunded

A finalReceipt is processed in a new block, refunding any extra gas paid by the user.

info A transaction is consideredfinal when all its receipts are processed. tip Most transactions will just spawn a receipt to process the actions, and a receipt to refund the gas, being final in 1-3 blocks (~1-3 seconds):

- One block
- if thesigner
- andreceiver
- coincide - e.g. when adding a key
- Three blocks
- if thesigner
- andreceiver
- differ, since the first block creates theReceipt
- , and the last reimburses gas

Function calls might take longer, as they can spawn multiple receipts. Network congestion can also increase the time to process a receipt, and thus a transaction.

Transaction Status

As theReceipts of aTransaction are processed they get an status:

- Success
- : the actions on the receipt were executed successfully
- Failed
- : an action on the receipt failed
- Unknown
- : the receipt is not known by the network

If an action in aReceipt fails, all the actions in thatReceipt are rollback. Notice that we are talking about theReceipt status,

and not theTransaction status.

The status of a transaction is determined by its first receipt, i.e. the one that contains all its actions. If any of the actions in the first receipt fail, the transaction would be mark as failed.

Notice that, it could happen that a transaction is marked as successful, but some of its receipt fails. This happens when aFunctionCall successfully spawns a new receipt, but the consequent function call fails. In this case, the transaction is marked as successful because the original function call was successful.

See the examples below for more details.

Status Examples

Example: Transaction with Transfer

1. bob.near
2. creates a transaction to transfer 10 NEAR toalice.near
3. The transaction is converted into a receipt
4. The conversion fails becausebob.near
5. does not have enough balance
6. The transaction is marked as failed

Example: Deploying a Contract

1. bob.near
2. creates a transaction to:* create the accountcontract.bob.near
3.
    - transfer 5 NEAR tocontract.bob.near
4.
    - deploy a contract incontract.bob.near
5. The transaction is transformed into one receipt
6. The account is created, the money transfer and the contract deployed
7. The transaction is marked as successful ✅

Example: Deploying a Contract Fails

1. bob.near
2. creates a transaction to:* create the accountcontract.bob.near
3.
    - transfer 5 NEAR tocontract.bob.near
4.
    - deploy a contract incontract.bob.near
5. The transaction is transformed into one receipt
6. The account is created, but the transfer fails becausebob.near
7. does not have enough balance
8. The whole process is reverted (i.e. no account is created)
9. The transaction is marked as successful

Example: Calling a Function

1. bob.near
2. creates a transaction to call the functioncross-call
3. incontract.near
4. The transaction is transformed into one receipt
5. The functioncross-call
6. creates a promise to call the functionexternal-call
7. inexternal.near
8. The function finishes correctly and the transaction is marked as successful ✅
9. A new receipt is created to call the functionexternal-call
10. inexternal.near
11. The functionexternal-call
12. fails
13. The original transaction is still marked as successful ✅ because the first receipt was successful tip You can check the status of a transactions using theNearBlocks explorer Edit this page Last updatedonMar 25, 2024 bygagdiez Was this page helpful? Yes No

Previous Gas (Execution Fees) Next NEAR Data Flow