Transaction Flow

This guide explains the transaction flow process for rollup transactions. The process for a rollup transaction has two requirements.

- The transaction needs to be written to L1 (Ethereum).
- This is typically performed byop-batcher
- , but any user can send an L1 transaction to submit an L2 transaction, in which caseop-batcher
- is bypassed.
- The transaction needs to be executed to modify the state (byop-geth
-).
- Afterwards,op-proposer
- writes acommitment(opens in a new tab)
- to the post-transaction state to L1.
- Note thatop-proposer
- does not need to write a commitment after each transaction to L1, it is OK to commit to the state after a block of transactions.

Writing the transaction to L1

op-batcher (opens in a new tab) has two main jobs:

- Compress transactions into batches.
- Post those batches to L1 to ensure availability and integrity.

Compression

The batcher aggregatessequencer batches(opens in a new tab) intochannels(opens in a new tab) . This allows for more data per compression frame, and therefore a better compression ratio. You can read more about this processin the specs(opens in a new tab) .

When a channel is full or times out it is compressed and written.

The maximum time that a channel can be open, from the first transaction to the last, is specified in units of L1 block time (so a value of 5 means 5*12=60 seconds). You can specify it either as an environment variable (OP_BATCHER_MAX_CHANNEL_DURATION) or a command line parameters (--max-channel-duration). Alternatively, you can set it to zero (the default) to avoid posting smaller, less cost efficient, transactions.

A channel is full when the anticipated compressed size is the target L1 transaction size. This is controlled by two parameters:

1. The target L1 transaction size, which you can specify in bytes on the command line (--target-l1-tx-size-bytes
2.) or as an environment variable (OP_BATCHER_TARGET_L1_TX_SIZE_BYTES
3.)
4. The expected compression ratio, which you can specify as a decimal value, again either on the command line (--approx-compr-ratio
5.) or as an environment variable (OP_BATCHER_APPROX_COMPR_RATIO
6.).

You can see the code that implements this process inchannel_manager.go (opens in a new tab) andchannel_builder.go (opens in a new tab) .

Posting to L1

When a channel is full it is posted, either as a single transaction or as multiple transactions (depending on data size) to L1.

Processed L2 transactions exist in one of three states:

- unsafe
- transactions are already processed, but not written to L1 yet.
- A batcher fault might cause these transactions to be dropped.
- safe
- transactions are already processed and written to L1.
- However, they might be dropped due to a reorganization at the L1 level.
- finalized
- transactions are written to L1 in an L1 block that is old enough to be extremely unlikely to be re-organized.

When are transactions irrevocable?

Once a transaction is finalized, you can rely that it has "happened". While the state after the transaction is subject to fault challenges, the transaction itself is fixed and immutable. You can see the code that builds the channels to be written to L1 in channel_out.go (opens in a new tab) and channel_builder.go (opens in a new tab) . The transactions themselves are sent in op-batcher 's main loop(opens in a new tab) , which calls publishStateToL1 (opens in a new tab)

Determining the status of a transaction

This is the procedure to see a transaction's status. The directions here are for Foundry(opens in a new tab) , but the concept is the same regardless of the method you use.

1. Get the number of the L2 block in which the transaction is recorded.
2. export
3. ETH_RPC_URL
4. =<
5. URL
6. to
7. Optimism
8. networ
9. k
10.

11. cast
12. tx
13. <
14. transaction
15. has
16. h
17.

18. blockNumber
19. Get the number of the latest finalized block.
20. If the result is greater than the block number of the transaction, or equal, the transaction is finalized.
21. cast
22. block
23. finalized
24. --field
25. number
26. Get the number of the latest safe block.
27. If the result is greater than the block number of the transaction, or equal, the transaction is safe.
28. cast
29. block
30. safe
31. --field
32. number
33. If the transaction isn't finalized or safe, it's unsafe.

State processing

State processing can be divided into two steps:

1. Applying the transaction to the old state to produce the new state, which is performed by op-geth (opens in a new tab)
2. .
3. Proposing the new Merkle root(opens in a new tab)
4. of the state.
5. Merkle roots are used because the actual state is long and would cost too much to write to L1.
6. This step is performed by op-proposer
7. .

State changes

The state is stored and modified by op-geth (opens in a new tab) . It is a slightly modified(opens in a new tab) version of the standard geth(opens in a new tab) .

State root proposals

The state root proposals are posted by op-proposer (opens in a new tab) to L2OutputOracle (opens in a new tab) on L1.

Output proposals are not immediately valid. They can only be considered authoritative once the fault challenge period (7

days on the production network, less on test networks) has passed.

Deposit Flow Withdrawal Flow