We have recently done some work overhauling our spec pertaining to gas and fees.

One significant departure between the spec as it exists and prior design is that we no long have the notion of L1 gas, which was intended to cover the costs associated with publishing messages from L2 to L1.

This post provides an explanation of the costs associated with processing cross chain messages, and why we could drop L1 gas.

It will be helpful for the reader to review the currentspec on cross chain messaging.

L1 to L2

Writing the L1->L2 message

When a sequencer publishes a block, the rollup contract on L1 calls consume

on the L1 Inbox. This computes the root of the frontier merkle tree stored in the L1 inbox contract, and computed root against the inHash

which is published as part of the content commitments in the header of the published block.

If the hashes match, then we are convinced that the published block has included all (up to 16) previous (with a delay of 1 block) messages.

Those parenthetical caveats are important: the first caps the work performed to a manageable, fixed amount, and the second prevents DoS.

This means that a sequencer cannot produce a valid block unless it consumes the appropriate L1->L2 messages, and inserts them into the L1_TO_L2_MESSAGE_TREE

, regardless of what, if any, messages were published to the L1 inbox.

Reading the L1->L2 message

Putting the message in the tree is separate from a contract actually reading the message. But this is a standard operation which is done in private (so we aren't concerned about computation) by emitting a nullifier to mark the message as read (which incurs DA gas).

pub fn consume_l1_to_l2_message(&mut self, content: Field, secret: Field, sender: EthAddress) { let nullifier = process_l1_to_l2_message(self.historical_header.state.l1_to_l2_message_tree.root, self.this_address(), sender, self.chain_id(), self.version(), content, secret); self.push_new_nullifier(nullifier, 0) }

L2 to L1

Writing the L2->L1 message

A transaction may have up to 2 L2->L1 messages. These messages must fit in a field element. We have the following on public and private contexts:

pub fn message_portal(&mut self, recipient: EthAddress, content: Field) { // docs:end:context_message_portal let message = L2ToL1Message { recipient, content }; self.new_l2_to_l1_msgs.push(message); }

Then we accumulate the hashes of the 2 (potentially zero/empty) L2->L1 messages per transaction in the base rollup, and finally insert the accumulated hashes across all TXs in the root.

This tree root is published with the block header's content commitments, and simply stored in the L1 Outbox contract.

Importantly:

1. The L2 to L1 message hashes are part of the "TxEffects

" of a transaction, and published to DA. For that reason, users pay DA gas for them.

1. Since the message hashes are part of TxEffects

, they are part of the content commitment for that transaction, so if the sequencer does not include them the published block will be invalid.

1. The sequencer does the work to compute the hashes of the messages and the subsequent state root regardless of whether the messages are empty, i.e. it does effectively no additional work processing non-empty versus empty messages (ignoring bitwise advantages of processing zero bytes).

Reading the L2->L1 message

Contracts on L1 are free to submit membership proofs to the L1 Outbox to consume messages. They pay their own gas in eth to do this as normal, independent of Aztec network operations.

Summary

Sequencers are forced to process cross-chain messages to produce valid blocks, and transactions use (primarily DA gas) to pay for the resources consumed by reading L1->L2 messages and writing L2->L1 messages, so there is no need, as far as we are aware, for a separate dimension of gas tailored to cross-chain messaging.