(h/t to @yahgwai

, @EdFelten, et al)

Background

The transaction fee an Arbitrum user pays is derived from two components; the “L2 fee” and the “L1 fee,” each with its own pricing mechanism. The L2 fee pays for L2 gas; it supports the parties expending resources to secure the chain, i.e., validators (this is closely analogous to the gas fee of an L1 Ethereum transaction).

The L1 fee exists to pay for a transaction’s calldata; it’s sent to the transaction’s batch-poster — typically & hereby the sequencer — to reimburse the L1 cost that the transaction contributed to its batch. The design goal here is that by collecting these L1 fees, the sequencer ultimately breaks even; i.e.,

(cost of posting batches) - (L1 fees collected) approaches 0 over time.

The Challenge: L1 Price Unpredictability

When the sequencer initially receives a transaction, it executes it and updates the Arbitrum chain’s off-chain state immediately; this includes extracting the L1 fee as per the Arbitrum chain’s current L1 calldata price. Some time later, the sequencer will include the transaction in a batch on L1 (batches are typically posted every few minutes).

In a naive implementation, this presents a problem: if, say, in the interim between applying the transaction off-chain and a batch being posted on chain, the actual L1 base fee increases, the sequencer ends up under-compensated for its batch and operates at a loss. Even an optimally well architected sequencer can’t predict the future (and simply letting the sequencer set the L1 data price — and in turn, how much it gets paid — would be an unacceptable trust assumption.).

Approach

Part of the Arbitrum Nitro upgrade will include a new mechanism that manages both reimbursing the sequencer for batch posting and for tracking/updating the L2’s L1 calldata price.

The flow works as follows:

1. When L2 transactions are processed off-chain by the sequencer, the L1 portion of the transaction fee is not sent directly to the sequencer, but instead collects on L2 in a sequencer “fee pool.”
2. The sequencer includes the transaction in a batch on L1. This L1 transaction also sends a cross-chain message reporting its cost to L2 (see first “Note” below for details).
3. The cross-chain message is received on L2; the L2 uses this message to calculate the L1 fees the sequencer actually paid on L1 to post the batch. This amount is transferred from the sequencer fee pool to the sequencer (if the fee pool has insufficient funds for this, the pool’s value can “go negative”, i.e., it is in debt to the sequencer, and the sequencer will be able to collect what it’s owed later.)
4. The L1 calldata price adjusts, retargeting as per the difference between the amount the sequencer paid on L1 and the reimbursement it received on L2; i.e. the price increases if the sequencer was undercompensated and decreases if there is a surplus in the fee pool.

Intuitively: the protocol regularly updates its L1 calldata price to stay as in-sync with L1 as it can, and any surplus or deficit in reimbursement fees is amortized over subsequent users of the chain. Tracking this surplus/deficit and adjusting accordingly is how the L2 view of L1 calldata price itself is maintained; i.e., how the protocol knows what to charge users for it in the first place.

Other Benefits

Conveniently, this protocol serves not only to handle the unpredictability of L1 price fluctuations, but covers other sources “L1 cost unpredictability” as well:

- Nitro will support calldata compression of posted batches; no heuristic for predicting a transaction’s contribution to the calldata of the entire compressed batch can be perfect, since we must price it before we have the whole batch.
- Arbitrum AnyTrust (non-rollup) chains optimistically don’t rely on posting data to L1, but under some circumstances they will fallback to posting transaction data in calldata.

The cost reporting / price adjustment / fee amortization process described above equally ensures that the seqeuencer is fairly reimburse for these conditions.

Notes/ Implementation Details

- To determine the gas paid by a sequencer in a batch, the sequencer sends a cross chain message which includes the L1 base fee paid when posting the batch; the L2, which has access to the contents of the batch data, uses this to

determine the amount paid. We need the batch contents (and not just its size) since zero bytes and non-zero bytes are charged differently by Ethereum.

- Note that the units used for charging L1 calldata don't need to be bytes, and indeed, in Nitro, will not be; instead, we use a "score" unit, which estimates the transaction's data footprint within its batch once compressed. The system works the same as described regardless of the unit used.