How to estimate gas in Arbitrum

Looking for Stylus guidance? Head over tothe Stylus gas docs for Stylus-specific guidance. This how-to is intended for users and developers interested in understanding how gas operates in Arbitrum, how it's calculated, and how to estimate it before submitting transactions. More detailed information about these calculations can be found in thisMedium article and theGas and Fees page.

Skip the formula, focus on practical know-how

Before diving into the specifics and the formula, if you're looking for a practical way to estimate gas for your transaction, you can rely on the standard gas estimation process. This can be achieved by calling an Arbitrum node'seth_estimateGas , which provides a value (gas limit) that should sufficiently cover the entire transaction fee at the specified L2 gas price.

Multiplying the value obtained frometh_estimateGas by the L2 gas price will give you the total amount of Ether required for the transaction to be successful. It's important to note that, for a specific operation, the result ofeth_estimateGas value may vary over time due to fluctuations in the L1 calldata price, see below to learn why!

Alternatively, to obtain the gas limit for your transaction, you can callNodeInterface.gasEstimateComponents() and then use the first result, which isgasEstimate . Next, to find the total cost, you need to multiply this amount by the L2 gas price, which is available in the third result,baseFee .

Note that when working with L1 to L2 messages (also known asretryable tickets), you can use the functionL1ToL2MessageGasEstimator.estimateAll() of the Arbitrum SDK orNodeInterface.estimateRetryableTicket() to get all the gas information needed to send a successful transaction.

Breaking down the formula

We'll now break down the formula mentioned in the Medium article, moving then to where to get the information of each variable, and finally seeing an example of how to apply the formula in your code as well as other practical ways of estimating gas costs.

However, if you want to jump straight to the code, we have createdthis script in our tutorials repository that goes through all the calculations explained in this how-to.

As explained in the Medium article, the transaction fees to pay at any given moment are the result of the following product:

Transaction fees (TXFEES) = L2 Gas Price (P) * Gas Limit (G) This Gas Limit includes the gas of the L2 computation and an additional buffer to cover the L1 gas to be paid by the Sequencer whenposting the batch including this transaction on L1 .

Gas Limit (G) = Gas used on L2 (L2G) + Extra Buffer for L1 cost (B) This buffer takes into account the cost of posting the transaction, batched and compressed, on L1. The L1 estimated posting cost is calculated by multiplying these two values:

- L1S, which estimates the amount of data the transaction will take up in the batch by compressing the transaction with Brotli.
- L1P, which is the L2's estimated view of the current L1's price of data (per byte), which the L2 dynamically adjusts over time.

More information is availablein this page .

L1 Estimated Cost (L1C) = L1 price per byte of data (L1P) * Size of data to be posted in bytes (L1S) To calculate the buffer, that estimated cost is divided by the L2 Gas Price.

Extra Buffer (B) = L1 Estimated Cost (L1C) / L2 Gas Price (P) Finally, using all of the above elements, the formula can be written as follows:

TXFEES = P * (L2G + ((L1P * L1S) / P))

Where do we get all this information from?

We'll use one resource available in Arbitrum: theNodeInterface .

- P (L2 Gas Price) ⇒ Price to pay for each gas unit. It starts at 0.01 gwei on Arbitrum One (0.01 gwei on Arbitrum Nova) and can increase depending on the demand for network resources.* CallNodeInterface.GasEstimateComponents()
-
    - and get the third element,baseFee
-
    - .
- L2G (Gas used on L2) ⇒ Gas used to compute the transaction on L2. This does not include the“posting on L1”

- part of the calculations. The value of L2G will depend on the transaction itself, but having the data of the transaction, we can calculate it as follows:* CallNodeInterface.GasEstimateComponents()
- with the transaction data and subtract the second element (gasEstimateForL1
- , which estimates the L1 part of the fees) from the first (gasEstimate
- , which includes both the L1 and the L2 parts).
- L1P (L1 estimated price per byte of data) ⇒ Estimated cost of posting 1 byte of data on L1:* CallNodeInterface.GasEstimateComponents()
- , get the fourth elementl1BaseFeeEstimate
- and multiply it by 16.
- L1S (Size of data to be posted on L1, in bytes) ⇒ This will depend on the data of the transaction. Keep in mind that Arbitrum adds a fixed amount to this number to make up for the static part of the transaction, which is also posted on L1 (140 bytes). We can do a small calculation to obtain this value: callNodeInterface.GasEstimateComponents()
- take the second element,gasEstimateForL1
- (this is equivalent toB
- in our formula), multiply it by P and divide it by L1P.* For Arbitrum Nova (AnyTrust), the size of the data is also a fixed value, as only the Data Availability Certificate is posted on L1,as explained here
- .

(Note: for L1P and L1S, you can also callNodeInterface.gasEstimateL1Component() to getl1BaseFeeEstimate andgasEstimateForL1)

An example of how to apply this formula in your code

Finally, we show an example of how to get the values we just described and how to estimate the gas usage of a transaction in Javascript. We'll use ourSDK to connect to theNodeInterface .

We first instantiate a factory object for the NodeInterface, using two methods from the SDK.l2Provider is a regular JSON RPC provider for the L2 network we are using, andNODE_INTERFACE_ADDRESS is the addresses that we need to call to access NodeInterface methods in said network.

```
const

{ NodeInterface__factory }

=

require ( "@arbitrum/sdk/dist/lib/abi/factories/NodeInterface__factory" ) ; const

{

NODE_INTERFACE_ADDRESS

}

=

require ( "@arbitrum/sdk/dist/lib/dataEntities/constants" ) ;

...
```

// Instantiation of the NodeInterface object const nodeInterface = NodeInterface__factory . connect (NODE_INTERFACE_ADDRESS , baseL2Provider) ; For this example, we'll use the methodNodeInterface.gasEstimateComponents() to get the information we need. For thegasEstimateComponents() call, we'll pass adestinationAddress (this should be the address that you intend to call in your transaction) and the data we want to send, to get results as accurate as possible. You can also specify a different block number (in hex) in the object passed as the last parameter.

```
// Getting the gas prices from ArbGasInfo.getPricesInWei() const gasComponents =

await arbGasInfo . callStatic . getPricesInWei ( ) ;

// And the estimations from NodeInterface.GasEstimateComponents() const gasEstimateComponents =

await nodeInterface . callStatic . gasEstimateComponents ( destinationAddress , false , txData , { blockTag :
```

'latest' , } ,) ; With this, we can now get the values of the 4 variables we'll use in our formula:

// Getting useful values for calculating the formula const l1GasEstimated = gasEstimateComponents . gasEstimateForL1 ; const l2GasUsed = gasEstimateComponents . gasEstimate . sub (gasEstimateComponents . gasEstimateForL1) ; const l2EstimatedPrice = gasEstimateComponents . baseFee ; const l1EstimatedPrice = gasEstimateComponents . l1BaseFeeEstimate . mul (16) ;

// Calculating some extra values to be able to apply all variables of the formula // ---------- // NOTE: This one might be a bit confusing, but l1GasEstimated (B in the formula) is calculated based on l2 gas fees const l1Cost = l1GasEstimated . mul (l2EstimatedPrice) ; // NOTE: This is similar to 140 + utils.hexDataLength(txData); const l1Size = l1Cost . div (l1EstimatedPrice) ;

// Setting the basic variables of the formula const

P

= l2EstimatedPrice ; const

L2G

= l2GasUsed ; const

L1P

= l1EstimatedPrice ; const

L1S

= l1Size ; And finally, we estimate the transaction fees applying the formula described in the beginning.

// L1C (L1 Cost) = L1P * L1S const

L1C

=

L1P . mul (L1S) ;

// B (Extra Buffer) = L1C / P const

B

=

L1C . div (P) ;

// G (Gas Limit) = L2G + B const

G

=

L2G . add (B) ;

// TXFEES (Transaction fees) = P * G const

TXFEES

=

P . mul (G) ; Refer to our tutorials repository for a working example of this code.

Final note

Note that gas estimations from the above techniques are approximate and the actual gas fees may differ. We encourage developers to set this expectation explicitly wherever this information is shared with end-users. Edit this page Last updatedonMar 22, 2024 Previous Quickstart: Build a decentralized app (Solidity) Next Arbitrum chains overview