Gas (Execution Fees)

On every transaction the NEAR network charges a tiny fee known asgas . This fee is a simple mechanism that allows us to:

1. Prevent
2. bad actors fromspamming
3. the network with useless transactions
4. Burn
5. a minuscule fraction of thetoken supply
6. on each transaction
7. Incentivize developers
8. by giving contracts 30% of the gas they burn while executing
9. Implement awall time
10. by capping transactions to300Tgas
11. (~300ms
12. of compute time)

Gas in NEAR is computed ongas units and charged using NEAR tokens based on the network'sgas price .

Did you know? In NEAR, attaching extra gas to a transaction doesnot make it faster. Actions cost a fixed amount of gas, and any extra gas attached is simply sent back to the user

Understanding Gas Fees

For every transaction, users get charged a small NEAR fee which has to be paidupfront . This fee is calculated using deterministicgas units , and transformed into a cost in NEAR using the network'sgas price .

Gas Units

Every action in NEAR costs a fixed amount ofgas units , meaning that the same operation will always cost thesame amount of gas units .

Gas units were engineered in such a way that they can be translated into compute resources, where1Tgas gets you approx.1ms of compute time.

Transactions can use a maximum of300Tgas , meaning they should be processed in less than300ms , allowing the network to produce a new block approximatelyevery second .

tip Gas units encapsulate not only compute/CPU time but also bandwidth/network time and storage/IO time

Gas Price

To determine the actual NEAR fee, the cost of all actions in the transaction are multiplied by agas price .

The gas price isrecalculated each block based on the network's demand and floors at1Tgas = 0.0001Ⓝ .

If the previous block ismore than half full the price goes up by 1%, otherwise it goes down by 1% (until it reaches the floor).

What is the gas price now? You can query how much a gas unit costs inyoctoNEAR (1Ⓝ =1e24 yocto) through theRPC . To convert inTgas perNEAR simply divide by1e12 .

Loading...

Cost for Common Actions

Knowing that actions have a fixed cost in gas units, we can calculate the cost of common operations at the minimum gas price of1Tgas = 0.0001Ⓝ .

Action TGas Fee (Ⓝ) Create Account 0.42 0.000042 Transfer NEAR 0.45 0.000045 Add Full Access Key 0.42 0.000042 Delete Key 0.41 0.000041 Function Call *≤ 300 ≤ 0.03 Deploying a16 kb contract 2.65 0.000265 Deploying aX kb contract** 0.58 + 0.13X Note that the fee is in NEAR, to obtain the cost in dollars multiply by the current price of NEAR

Function Calls *The cost of calling a function will depend on how complex the function is, but will be consistent across function calls. Learn more bellow. Deploying a Contract** Note that this covers the cost of uploading and writing bytes to storage, but doesnot cover the cost ofholding these bytes in storage Where do these numbers come from? NEAR isconfigured with base costs. An example:

transfer_cost: { send_sir: 115123062500, send_not_sir: 115123062500, execution: 115123062500 }, deploy_contract_cost:

184765750000, deploy_contract_cost_per_byte: 64572944 The "sir" here stands for "sender is receiver". Yes, these are all identical, but that could change in the future.

When you make a request to transfer funds, NEAR immediately deducts the appropriatesend amount from your account. Then it creates areceipt , an internal book-keeping mechanism. Creating a receipt has its own associated costs:

action_receipt_creation_config: { send_sir: 108059500000, send_not_sir: 108059500000, execution: 108059500000 } You can query this value by using theprotocol_config RPC endpoint and search foraction_receipt_creation_config .

The appropriate amount for creating this receipt is also immediately deducted from your account.

The "transfer" action won't be finalized until the next block. At this point, theexecution amount for each of these actions will be deducted from your account (something subtle: the gas units on this next block could be multiplied by a gas price that's up to 1% different, since gas price is recalculated on each block). Adding it all up to find the total transaction fee:

(transfer_cost.send_not_sir + action_receipt_creation_config.send_not_sir) * gas_price_at_block_1 + (transfer_cost.execution + action_receipt_creation_config.execution) * gas_price_at_block_2

How Do I Buy Gas?

You don't buy gas, instead, the gas fee is automatically removed from your account's balance when the transactionis first proccesed based on the action's gas cost and the network's gas price.

The only exception to this rule is when you make a function call to a contract. In this case, you need to define how many gas units to use, up to a maximum value of300Tgas . This amount will be converted to NEAR using the network's gas price and deducted from your account's balance.

Since many transactions will take morethan 1 block to execute, and the gas price is recalculated on each block and could go up, you will be charged a pessimistic estimate of NEAR (see details bellow).

If the transaction ends up using less gas than the amount deducted, the difference will simply berefunded to your account .

Pessimistic Estimate While actions have a fixed cost in gas units, the gas price might change block to block. Since transactions can take more than 1 block to execute, the gas price might go up during the transaction's execution.

To avoid the need to recalculate the gas price for each block, the network will charge you upfront a pessimistic estimate of the gas fee.

Lets take as an examplethis transaction calling a contract method. The transaction was submitted with 10Tgas attached.

- 10Tgas would cost 0.001Ⓝ at the price when the transaction was submitted
- The transaction used:* 2.4Tgas to convert thetransaction into a receipt
-
    - : 0.00024Ⓝ
-
    - 3.2Tgas to execute the function in the contract: 0.00032Ⓝ
-
    - Total: 5.6Tgas or 0.00056Ⓝ
- In the end, the user was returned 0.00104Ⓝ

Since the system returned0.00104Ⓝ , and the transaction expended0.00056Ⓝ , the user was charged upfront0.0016Ⓝ , this is 60% more than what the user expected to pay (0.001Ⓝ).

This 60% up comes from assuming that the price of gas will go up by 1% on each block, and the transaction will take 50 blocks to execute (1.01**50 ~ 1.64). tip In other chains paying a higher gas price to get your transaction processed faster. In NEAR,gas costs are deterministic , and youcan't pay extra . Any extra gas attached to a transaction is simply sent back to the user.

Gas as a Developer Incentive

In NEAR, 30% of the gas fees burn while executing a contract go to the contract's accounts. This is a powerful incentive for developers to create and maintain useful contracts.

For example, inthis transaction the user calls a function in theguestbook.near-examples.testnet contract.

Executing the function call burned a total of ~0.00032Ⓝ, from which 30% goes to the contract's account. This means that the contract's account received 0.000096Ⓝ.

Notice that the fee comes from the gas burned during the function execution, and not from the total gas used.

Estimating Costs for a Call

warning This section will soon be moved from here to the build documentation . If you developing a smart contract, you might want to estimate how much gas a function call will consume. This is useful to estimate limits for your function and avoid running into out-of-gas errors.

One of the most accurate ways to estimate gas costs is by running your function intestnet . To know exactly how much gas a specific part of your function uses, you can use theused_gas method from our SDK.

Another option is to useSandbox Testing (available in Rust and JavaScript), which simulates the NEAR network. There you can access the gas burnt after each function call.

Finally, you can obtain gas cost estimates for a given function call usingapi.gasbuddy.tech . This API is experimental and may be removed in the future. One can obtain a gas cost estimate for a given function call by sending a POST request tohttps://api.gasbuddy.tech/profile with the following JSON body:

{ "contract_id": "", "method": "", "args": { "arg1": "value1", "arg2": "value2" } }Edit this page Last updatedonMar 25, 2024 bygagdiez Was this page helpful? Yes No