

Gas (Execution Fees)

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

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

Gas in NEAR is computed on [gas units](#) and charged using NEAR tokens based on the network's [gas price](#).

Did you know? In NEAR, attaching extra gas to a transaction does not 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 paid upfront. This fee is calculated using deterministic gas units, and transformed into a cost in NEAR using the network's gas price.

Gas Units

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

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

Transactions can use a maximum of 300 Tgas, meaning they should be processed in less than 300ms, allowing the network to produce a new block approximately every 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 a gas price.

The gas price is recalculated each block based on the network's demand and floors at $1 \text{ Tgas} = 0.0001 \text{ ₿}$.

If the previous block is more 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 in yoctoNEAR ($1 \text{ ₿} = 1 \text{e}24$ yocto) through the [RPC](#). To convert in Tgas per NEAR simply divide by $1 \text{e}12$.

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 of $1 \text{ Tgas} = 0.0001 \text{ ₿}$.

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 $\leq 300 \leq 0.03$ Deploying a 16 kb contract 2.65 0.000265 Deploying a X 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 below. Deploying a Contract** Note that this covers the cost of uploading and writing bytes to storage, but does not cover the cost of [holding these bytes in storage](#) Where do these numbers come from? NEAR is [configured](#) 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 appropriate send amount from your account. Then it creates [a receipt, 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 the [protocol_config](#) RPC endpoint and search for action_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, the execution 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:

$(\text{transfer_cost.send_not_sir} + \text{action_receipt_creation_config.send_not_sir}) * \text{gas_price_at_block_1} +$
 $(\text{transfer_cost.execution} + \text{action_receipt_creation_config.execution}) * \text{gas_price_at_block_2}$

How Do I Buy Gas?

You don't buy gas, instead, the gas fee is automatically removed from your account's balance when the transaction is [first processed](#) 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 of 300Tgas. 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 more [than 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 below).

If the transaction ends up using less gas than the amount deducted, the difference will simply be refunded 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 example [this 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 the [transaction 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 returned 0.00104[Ⓝ], and the transaction expended 0.00056[Ⓝ], the user was charged upfront 0.0016[Ⓝ], 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 ($1.01^{50} \sim 1.64$). tip In other chains paying a higher gas price to get your transaction processed faster. In NEAR, gas costs are deterministic, and you can'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, in [this transaction](#) the user calls a function in the `guestbook.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 in testnet . To know exactly how much gas a specific part of your function uses, you can use the `used_gas` method from our SDK.

Another option is to use Sandbox 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 using [api.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 to <https://api.gasbuddy.tech/profile> with the following JSON body:

```
{ "contract_id": "", "method": "", "args": { "arg1": "value1", "arg2": "value2" } }
```

[Edit this page](#) Last updated on Mar 25, 2024 by gagdiez Was this page helpful? Yes No

[Previous Anatomy of a Transaction](#) [Next Lifecycle of a Transaction](#)