Gas

On every transaction you send to the network NEAR charges you a fee (akagas fee). This fee is used to indirectly pay thepeople that keep the network infrastructure, and to incentivize developers of smart contracts.

Gas is a fascinating topic that touches everyone in the NEAR ecosystem, here is a brief summary:

1. A small fee is charged on every transaction to indirectlypay the validators
2. by burning a part of the total token supply.
3. This feeprevents spamming
4. the network with useless transactions.
5. Read-only methods do notresult in fees for the user
6. , instead, the validator absorbs the cost.
7. In transaction involving a contract,30% of the fee
8. goes to the contract as adeveloper incentive
9. .
10. Fees are measured ingas units
11. but paid in NEAR.
12. Gas units are deterministic: thesame transaction
13. costs thesame gas units
14. .
15. Gas units are transformed to NEAR by multiplying for agas price
16. .
17. Thegas price variates
18. smoothly from block to block.
19. Gas can be thought as awall time
20. :1 Tgas
21. ~1 ms
22. of compute time.
23. You can attach amaximum of300Tgas
24. to a transaction.
25. Attaching extra gas doesNOT
26. make the transaction faster, unused gas issimply returned
27. .
28. Contract developers canprepay gas
29. for their users.

Introduction

When you send a transaction to the NEAR network differentvalidators process it using their own infrastructure.

Maintaining the infrastructure up and running is important to keep the network healthy, but represents a significant cost for the validator.

As many other networks, NEARpays the validators for their job. Also like many other networks, users have to pay a smallfee (akagas fee) on every transaction. But instead of giving the gas fee to the validators, validators receive their reward independent from the gas fees. This topic is discussed in more details in thevalidators section.

In addition, NEAR implements two unique features with respect to gas fees:

1. Sharing fees with developers
2. Allowing forfree
3. transactions

1. Gas as a Developer Incentive

Something unique to NEAR is that the GAS is not used to pay validators. In transactions where calling a contract would incur a gas fee, the fee is actually divided as follows:

- 30% goes to the smart contract.
- 70% is burned.

In this way, NEAR uses the gas to alsoincentive development of dApps in the ecosystem.

2. Free Transactions

Another unique feature of NEAR is that it allows to callread-only methods in smart contracts for free, without even needing a NEAR account.

In such case, it is the validators who absorb the gas cost.

Gas Units & Gas Price

On every transaction NEAR users get charged a small NEAR fee, which has to be paid upfront. However, transaction fees are not computed directly in NEAR.

Gas Units

Internally, the computation is done usinggas units which aredeterministic , meaning that a given operation will always cost thesame amount of gas .

Gas Price

To determine the actual NEAR fee the gas of all operations done by the transaction are added up are multiplied by agas price .

The gas price is not fixed: it isrecalculated each block depending on network demand. If the previous block is more than half full the price goes up, otherwise it goes down.

The price cannot change by more than 1% each block and bottoms out at a price that's configured by the network (currently 100 million yocto NEAR).

Translating Gas to Computational Resources

Gas units have been carefully calculated to work out to some easy-to-think-in numbers:

- 1 TGas
- (10^{12} gas units) ≈1 millisecond
- of "compute" time.
- This represents0.1 milliNEAR
- (using theminimum gas price
-).

This1ms is a rough but useful approximation. However, gas units encapsulate not only compute/CPU time but also bandwidth/network time and storage/IO time.

Via a governance mechanism, system parameters might be tweaked, shifting the mapping between TGas and milliseconds in the future.

1s Block Production NEAR imposes amaximum amount of gas per block to ensure that a block is generated approx. every second.

The cost of common actions

To give you a starting point for what to expect for costs on NEAR, the table below lists the cost of some common actions in TGas and milliNEAR (at theminimum gas price).

Operation TGas fee (mN) fee (Ⓝ) Create Account 0.42 0.042 4.2×10^{-5} Send Funds 0.45 0.045 4.5×10^{-5} Stake 0.50 0.050 5.0×10^{-5} Add Full Access Key 0.42 0.042 4.2×10^{-5} Delete Key 0.41 0.041 4.1×10^{-5} Where do these numbers come from? NEAR isconfigured with base costs. An example:

transfer_cost: { send_sir: 115123062500, send_not_sir: 115123062500, execution: 115123062500 } 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 to facilitate NEAR's asynchronous, sharded design (if you're coming from Ethereum, forget what you know about Ethereum's receipts, as they're completely different). 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 appropriatesend 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 directly buy gas; you attach tokens to transactions.

If you're coming from Ethereum, you may be used to the idea of paying a higher gas price to get your transaction processed faster. In NEAR,gas costs are deterministic , and youcan't pay extra .

For basic operations like transfers the gas needed is easy to calculate ahead of time, so it'sautomatically attached for you .

Function calls are more complex and need you to attach an explicit amount of gas to the transactions, up to a maximum value of 3×10^{14} gas units (300 Tgas).

info This maximum value of prepaid gas is subject to change but you can query this value by using theprotocol_config RPC endpoint and search formax_total_prepaid_gas . Details How many tokens will these units cost? Note that you are greenlighting a maximum number of gasunits , not a number of NEAR tokens or yoctoNEAR.

These units will be multiplied by the gas price at the block in which they're processed. If the function call makes cross-contract calls, then separate parts of the function will be processed in different blocks, and could use different gas prices. At a minimum, the function will take two blocks to complete, as explained inwhere those numbers come from .

Assuming the system rests at minimum gas price of 100 million yoctoNEAR during the total operation, a maximum attached gas of 3×10^{14} would seem to allow a maximum expenditure of 3×10^{22} yN. However, there's also a pessimistic multiplier of about 6.4 toprevent shard congestion .

Multiplying all three of these numbers, we find that maximum attached gas units allow about 0.2Ⓝ to be spent on the operation if gas prices stay at their minimum. If gas prices are above the minimum, this charge could be higher.

What if the gas price is at the minimum during the starting block, but the operation takes several blocks to complete, and subsequent blocks have higher gas prices? Could the charge be more than ~0.2Ⓝ? No. The pessimistic multiplier accounts for this possibility.

Attach extra gas; get refunded

The amount of gas required to call a contract depends on the method's complexity and the contract's state. Many times this ishard to predict ahead of time.

Because of this, if you attach more tokens than needed to cover the gas, you'll get refunded the unused fee!

This is also true for basic operations. In thecost section we mentioned that NEAR fees are automatically calculated and attached. Since the gas price could be adjusted while these operations are being applied, a slight amount extra is attached, and any beyond what's necessary gets refunded.

What about Prepaid Gas?

The NEAR Team understands that developers want to provide their users with the best possible onboarding experience. To realize this vision, developers can design their applications in a way that first-time users can draw funds for purchasing gas directly from an account maintained by the developer. Once onboarded, users can then transition to paying for their own platform use.

In this sense, prepaid gas can be realized using a funded account and related contract(s) for onboarding new users.

So how can a developer pay the gas fee for their users on NEAR?

- A user can use the funds directly from the developer's account suitable only for the gas fees on this dApp. Then the developer has to distinguish users based on the signers' keys instead of the account names.
- Using function calls, you can allow a new user without an account to use your dApp and your contract on-chain. The back-end creates a new access key for the user on the contract's account and points it towards the contract itself. Now the user can immediately use the web app without going through any wallet.

NEAR Protocol does not provide any limiting feature on the usage of developer funds. Developers can set allowances on access keys that correspond to specific users -- oneFunctionCall access key per new user with a specific allowance. Edit this page Last updatedonJan 31, 2024 bygagdiez Was this page helpful? Yes No

Previous Overview Next Gas - Advanced