BlockScholes

20TH JANUARY 2023

Ethereum's Gas Mechanics

The Ethereum network's pivot to a new consensus mechanism in September 2022 caused a dramatic shakeup to the reward system available to block-builders. Maximal extractable value, previously merely a popular side project for Proof of Work miners, has now become the most attractive reward stream for validators, and understanding its nuances is key to a profitable validator operation.

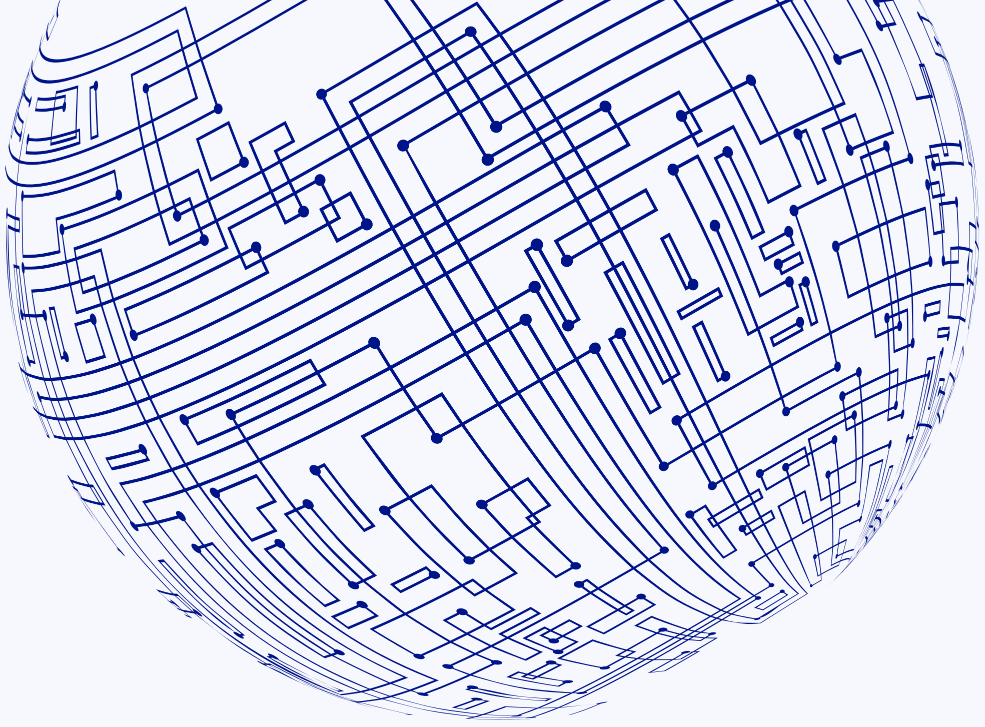


Table of contents

Ethereum's Gas

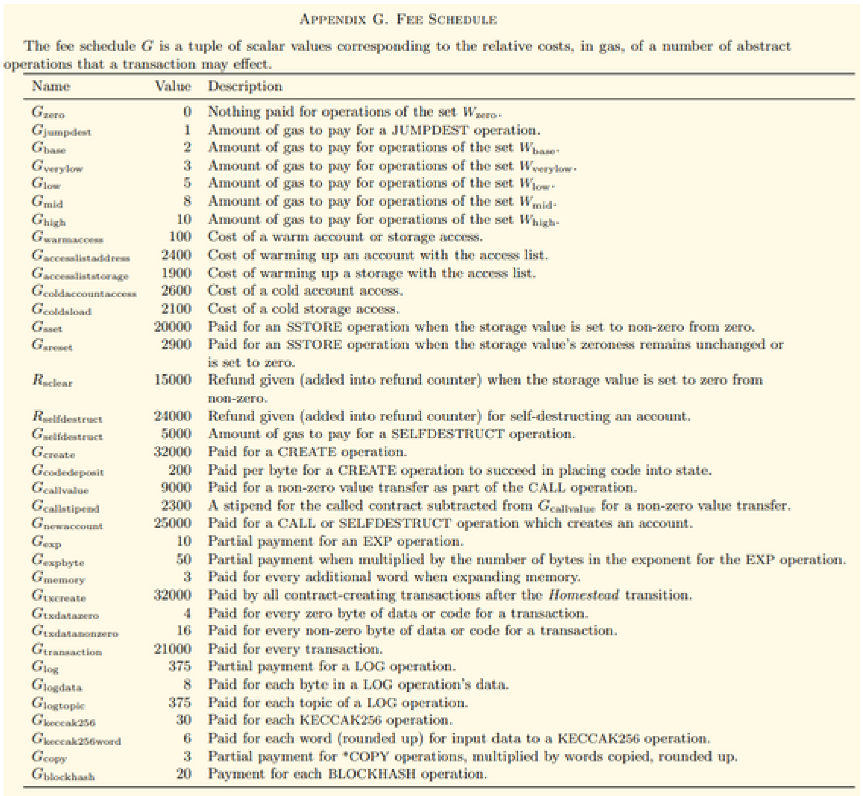
INTRODUCTION

The Ethereum network boasts the biggest Decentralised Finance (DeFi) ecosystem of any blockchain, an ecosystem supported by a network of smart contracts that allow for composable blocks of code to be executed by validators in the form of onchain transactions. The results of those computations are transmitted in the blocks appended to the chain every 12 seconds and dictate necessary updates to the state of the Ethereum Virtual Machine (EVM). The computation required by a transaction is performed by the validators (analogous to Bitcoin's miners) who create new blocks on the chain and is paid for in ETH by the sender.

WHAT DOES GAS MEASURE?

Validators collect fees from users for each transaction they include in a block in addition to their income from staking rewards. The size of those fees is dependent on the amount of computation required by the transaction or call to a smart contract. This is to compensate validators for the computational work they complete on behalf of the user.

That computational effort is measured in "gas", with a set of standardised computational operations assigned a fixed gas value in the Ethereum yellow paper. The greater the complexity of the operation, the greater the number of gas units it will need to be executed. Therefore, complex smart contracts with many operations will require more gas to be executed, so developers are incentivised to keep their smart contract code concise.



Appendix G. Fee Schedule

The fee schedule G is a tuple of scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{jumpdest}	1	Amount of gas to pay for a JUMPDEST operation.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
G_{verylow}	3	Amount of gas to pay for operations of the set W_{verylow}.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{\text{warmaccess}}$	100	Cost of a warm account or storage access.
$G_{\text{accesslistaddress}}$	2400	Cost of warming up an account with the access list.
$G_{\text{accessliststorage}}$	1900	Cost of warming up a storage with the access list.
$G_{\text{coldaccountaccess}}$	2600	Cost of a cold account access.
$G_{\text{coldsload}}$	2100	Cost of a cold storage access.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	2900	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{\text{selfdestruct}}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{\text{selfdestruct}}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{\text{codedeposit}}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
$G_{\text{callvalue}}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{\text{callstipend}}$	2300	A stipend for the called contract subtracted from $G_{\text{callvalue}}$ for a non-zero value transfer.
$G_{\text{newaccount}}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
G_{expbyte}	50	Partial payment when multiplied by the number of bytes in the exponent for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
G_{txcreate}	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{\text{txdatazero}}$	4	Paid for every zero byte of data or code for a transaction.
$G_{\text{txdatanonzero}}$	16	Paid for every non-zero byte of data or code for a transaction.
$G_{\text{transaction}}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
G_{logdata}	8	Paid for each byte in a LOG operation's data.
G_{logtopic}	375	Paid for each topic of a LOG operation.
$G_{\text{keccak256}}$	30	Paid for each KECCAK256 operation.
$G_{\text{keccak256word}}$	6	Paid for each word (rounded up) for input data to a KECCAK256 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{\text{blockhash}}$	20	Payment for each BLOCKHASH operation.

Figure 1 Implied volatility of a 25-delta BTC call divided by the implied volatility of a 25-delta BTC put at a 1 week (orange), 1 month (blue), and 3 month (red) tenor. *Source: Block Scholes*

For example, a regular transaction of ETH between two users would require 21,000 gas to be executed. Alternatively, a transaction that initialises a smart contract onchain requires 32,000 gas to be executed. Some transactions can actually refund gas to the user that calls it. For example, 15,000 gas is refunded to users who send transactions that free up storage on smart contracts by setting non-zero values to zero. This is done to incentivise users to free up storage space on the evergrowing blockchain.

BITCOIN FEES

On the Bitcoin network, the transaction fees are measured in satoshis (0.00000001 BTC) per vbytes (virtual byte), meaning that the minimum payment you must make to have a transaction included inside a block is 1e-8 BTC, the smallest denomination of BTC recorded on its blockchain. Similar to ETH, this fee is paid to miners to incentivise them to include transactions inside a block instead of mining empty blocks.

The number of bitcoin a user pays as a fee depends on 2 things: the size of the transaction, which denotes the actual number of vbytes it will take in a block, and satoshis-per-vbyte. The first portion of the transaction depends on the number of inputs and outputs of the transaction, and the latter is a user-specified amount. The total fee paid is the product of these two factors. When many users are trying to send transactions on the network, miners will choose to only include transactions that have high fees associated with them. Therefore users need to adequately specify the fees in order to outcompete other users and have their transactions added on chain.

HOW IS EACH UNIT OF GAS PRICED?

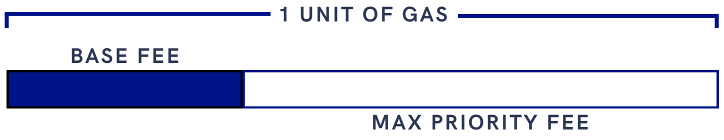
Like BTC on the Bitcoin network, units of gas are paid for in Ethereum's native currency, ETH. However, due to their incredibly small value, they are more commonly quoted in gwei (Giga-wei). Each wei denotes 1 x 10^-18 ETH (the smallest denomination of ETH representable in the EVM's onchain), and hence 1 gwei is equal to 10^-9 ETH.

Blocks have a maximum limit of 30,000,000 gas units worth of transactions, called the gas limit, which places a cap on the computational expenses that a validator must perform in executing the block. Despite this gas limit, Ethereum blocks have a target size of 15,000,000 gas units. This is a target set by the Ethereum Foundation, which they expect to reflect "normal" network congestion. The importance of this target will be discussed later. By limiting the amount of gas expended in a block, the protocol also limits the number of transactions that can fit into a block.

Transaction Fee Structure

As users are required to pay an ETH-denominated fee for each unit of gas that their transaction uses, it is common for users of the network to talk about the ETH "gas fee", or the *Fee per Gas* when discussing transaction fees.

The *Fee Per Gas* parameter is broken down into 2 components: the *Base Fee*, which the user must pay if they want to have their transaction included on the chain, and the *Priority Fee*, which is an optional component set by the user when submitting a transaction.



BASE FEE

The Base Fee is a parameter which is computed by the protocol itself (not by validators) in each block, that changes depending on the congestion of the Ethereum network. Congestion on the network is measured by how close each block's gas size is to the *target size*.

Recall that blocks have a *target size* of 15,000,000 gas units which the developers of the network deem a healthy size for blocks on the chain. If a block contains transactions worth more than the 15,000,000 target size, the base fee of the subsequent block will increase by up to a maximum of 12.5% (which would occur when the previous block is at a full capacity of 30,000,000 gas units). Alternatively, when blocks are below this target size the base fee of the subsequent block will decrease and can do so by a maximum of 12.5% (when the block is empty).

This proportion of the fee is burnt and removed from the circulation of ETH supply. This is in contrast to the system employed by the Bitcoin network, which does not burn tokens (but is set at a hard limit of 21M BTC). The number of tokens burnt via the base fee offsets the new issuance of ETH to validators. Since the Merge in September 2022, the net growth of ETH supply is near zero.

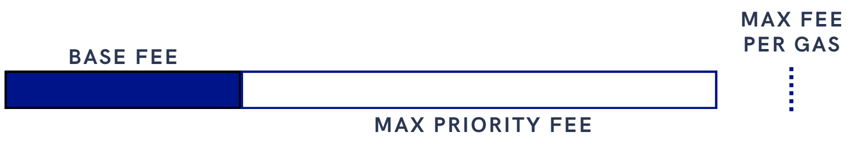
MAX PRIORITY FEE

The *Max Priority Fee* is the second component of the bid made by the user. This is the portion of the gas fee that is paid directly to the validator. Unlike the Base Fee, this is a non-random value that is set by the user submitting the transaction.

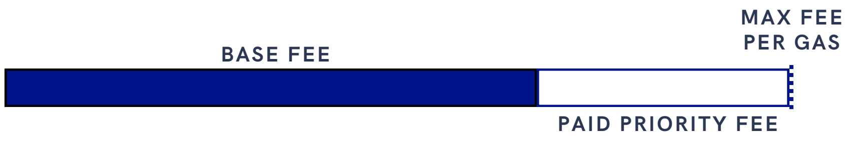
Users are free to make this value as large (or as small) as possible, and larger tips will incentivise validators to include their transactions into a block before other transactions with lower priority fees.

MAX FEE PER GAS

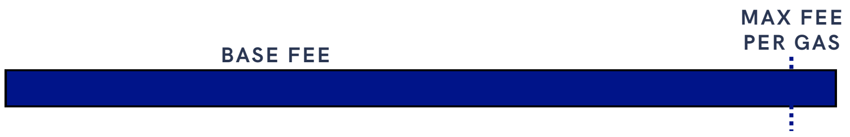
Whilst the *Max Priority Fee* per Gas is set by the user, the unpredictable nature of the *Base Fee* means that the actual price per gas paid for the transaction is not known at the point of sending. To avoid paying fees that are beyond their budget, the *Max Priority Fee* per unit of gas allows the user to set the maximum amount of ETH per unit of gas they are willing to pay to have their transactions included in a block.



However, Base Fees increasing suddenly could cause the sum of the Base Fee and the Priority Fee to exceed the Max Fee per Gas. In this case, the Priority Fee decreases to account for this upper limit. This may cause the actual priority fee paid as a tip to the miner to be lower than the Max Priority Fee specified by the user, which will ultimately make the user's transaction less attractive to validators.



In an extreme case, where the network is highly congested, the *Base Fee* could completely dominate the *Priority Fee*. In this case, the *Base Fee* would be equal to the *Max Fee er Gas* parameter. If the Base Fee exceeds the Max Fee Per Gas then the transaction would remain unprocessed in the mempool until the network becomes less congested, lowering *Base Fees* and making the user's bid competitive once again.



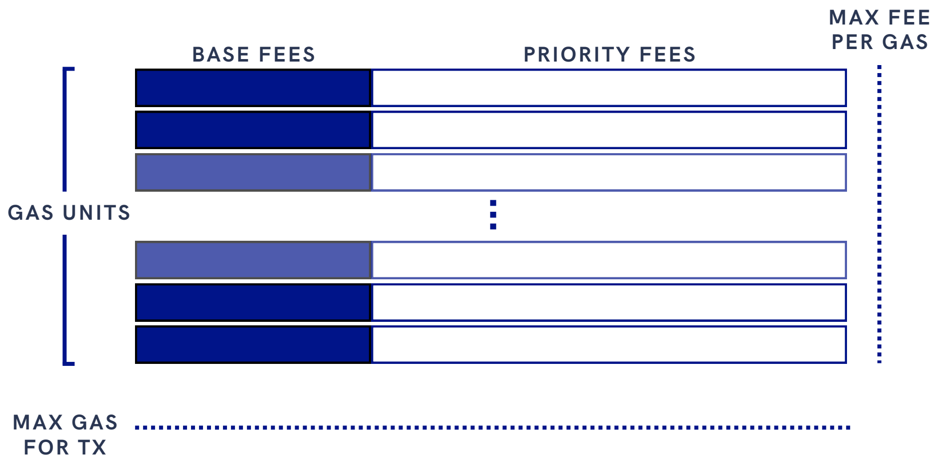
Note that the *Max Fee Per Gas* parameter is not the total fee that the user is willing to pay for their transaction, but rather the maximum amount a user is willing to *pay per unit of gas*. In order to know the final total fee, we must multiply the fee paid per gas by the actual units of gas consumed by the transaction. However, the fee per gas does have other implications for the user's transaction, as transactions are only processed if the *Base Fee* is less than the *Max Fee Per Gas* parameter.

GAS LIMIT

The parameters above allow a user to control the price they pay per unit of gas. That control assumes a known, constant amount of gas for a transaction sent to the mempool. In reality, the gas requirements of a transaction are not immediately known to the user at the point of submission. Calculating the computation required to complete a transaction requires actually computing the transaction.

To mitigate this, users can specify a gas limit that specifies the maximum units of gas that their transaction is allowed to consume. If their transaction requires more gas than was budgeted for by their upper limit, a validator will halt the transaction and move on, regardless of if the computation has been completed. Therefore this gas limit determines the total amount of ETH committed by a user to a transaction, with any unused gas budget resulting in a partial refund to the user.

In order to obtain the total amount of ETH a user submits to process a transaction, we must multiply their specified *Gas Limit* by the *Max Fee Per Gas* determined by the user in the sections above.



If however, the transaction required a larger amount of gas than was specified, the transaction gets reverted, the validator gets the full amount of gas specified by the user and the user receives no refund. This highlights the need for an efficient user to specify a high enough gas limit parameter to avoid paying for an unsuccessful transaction.

Conclusion

Ethereum's gas mechanism is the system that underpins the smooth operation of the network, by translating blockspace demand into a fee that users have to pay to access the network. It is the system that bridges the needs and wants of both validators and users on the network by creating a market for blockspace. This fundamental part of the network has numerous consequences for users wishing to use the network. We hope to explore these in future reports.

CONTACT US

AUTHORS

AHMAD MUSTAFA KIDA
Data Analyst
ahmad.kida@blockscholes.com

Block Scholes Ltd.

27 Old Gloucester Street
London WC1N 3AX

research@blockscholes.com

SUBSCRIBE TO OUR PLATFORM