Accounts, Transactions, Gas, and Block Gas Limits in Ethereum

Posted on June 27, 2017

This article is meant to help people understand some of the basic mechanics behind accounts, transactions, gas, and the role miners play in setting the block size in Ethereum. Corrections are welcome :)

What are accounts?

EOA vs contract accounts

There are two types of accounts in Ethereum

- Externally Owned Accounts
- Contracts Accounts

This distinction will be abstracted in the upcoming Metropolis upgrade (https://github.com/ethereum/EIPs/pull/208).

Externally owned accounts (EOAs)

An externally controlled account

- has an ether balance,
- can send transactions (ether transfer or trigger contract code),
- is controlled by private keys,
- has no associated code.

Contract accounts

A contract

- has an ether balance,
- has associated code,
- code execution is triggered by transactions or messages (calls) received from other contracts.
- when executed - perform operations of arbitrary complexity (Turing completeness) - manipulate its own persistent storage, i.e. can have its own permanent state - can call other contracts

All action on the Ethereum block chain is set in motion by transactions fired from accounts. Every time a contract account receives a transaction, its code is executed as instructed by the input parameters sent as part of the transaction. The contract code is executed by the Ethereum Virtual Machine on each node participating in the network as part of their verification of new blocks.

What are transactions and messages?

Transactions

The term “transaction” is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account to another account on the blockchain.

Transactions contain:

- the recipient of the message,
- a signature identifying the sender and proving their intention to send the message via the blockchain to the recipient,
- `VALUE` field - The amount of wei to transfer from the sender to the recipient,
- an optional data field, which can contain the message sent to a contract,
- a `GASLIMIT` value, representing the maximum number of computational steps the transaction execution is allowed to take,
- a `GASPRICE` value, representing the fee the sender is willing to pay for gas. One unit of gas corresponds to the execution of one atomic instruction, i.e. a computational step.

Messages

Contracts have the ability to send “messages” to other contracts. Messages are virtual objects that are never serialized and exist only in the Ethereum execution environment. They can be conceived of as function calls.

A message contains:

- the sender of the message (implicit).
- the recipient of the message
- `VALUE` field - The amount of wei to transfer alongside the message to the contract address,
- an optional data field, that is the actual input data to the contract
- a `GASLIMIT` value, which limits the maximum amount of gas the code execution triggered by the message can incur.

Essentially, a message is like a transaction, except it is produced by a contract and not an external actor. A message is produced when a contract currently executing code executes the `CALL` or `DELEGATECALL` opcodes, which produces and executes a message. Messages are also sometimes called “internal transactions.” Like a transaction, a message leads to the recipient account running its code. Thus, contracts can have relationships with other contracts in exactly the same way that external actors can. Many times people use the term transaction when they are referring to a message, so it is possible that this term is being phased out via community consensus by not using it.

What is gas?

Ethereum implements an execution environment on the blockchain called the Ethereum Virtual Machine (EVM). Every node participating in the network runs the EVM as part of the block verification protocol. They go through the transactions listed in the block they are verifying and run the code as triggered by the transaction within the EVM. Each and every full node in the network does the same calculations and stores the same values. The fact that contract executions are redundantly replicated across nodes, naturally makes them expensive, which generally creates an incentive not to use the blockchain for computation that can be done offchain. For every executed operation there is a specified cost, expressed in a number of gas units. Every operation that a contract is able to take advantage of has an associated gas value. Here is an outdated list of gas costs per operation code (opcode) (https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCcNK950dUzMQPMJBxRtGCqs/edit#gid=0).

Gas and transaction costs

Every transaction is required to include a gas limit (sometimes called `startGas` (https://media.consensys.net/ethereum-gas-fuel-and-fees-3333e17fe1dc)) and a fee that it is willing to pay per gas. Miners have the choice of including the transaction and collecting the fee or not. In reality, today all transactions are picked up by miners eventually, but the amount of transaction fees that a user chooses to send affects how long it will take until the transaction is mined. If the total amount of gas used by the computational steps spawned by the transaction, including the original message and any sub-messages that may be triggered, is less than or equal to the gas limit, then the transaction is processed. If the total gas exceeds the gas limit, then all changes are reverted, except that the transaction is still valid and the fee can still be collected by the miner. The blockchain shows that a transaction was attempted, but it did not provide enough gas and all contract operations were reverted. All excess gas not used by the transaction execution is reimbursed to the sender as Ether. Because gas cost estimates are only approximate, many users overpay in gas to guarentee that their transaction is accepted. This is okay because any excess gas is refunded to you.

Estimating transaction costs

The total ether cost of a transaction is based on 2 factors:

- `gasUsed` : the total gas that is consumed by the transaction
- `gasPrice` : price (in ether) of one unit of gas specified in the transaction

Total cost = gasUsed * gasPrice

gasUsed

Each operation in the EVM was assigned a number of how much gas it consumes. gasUsed is the sum of all the gas for all the operations executed.

For estimating gasUsed, there is an estimateGas API that can be used but has some caveats (http://ethereum.stackexchange.com/q/266/42).

gasPrice

A user constructs and signs a transaction, and each user may specify whatever `gasPrice` they desire, which can be zero. However, the Ethereum clients launched at Frontier had a default `gasPrice` of 0.05e12 wei. As miners optimize for their revenue, if most transactions are being submitted with a `gasPrice` of 0.05e12 wei, it would be difficult to convince a miner to accept a transaction that specified a lower, or zero, `gasPrice` .

Example transaction cost

With permission, I am borrowing this example and analogy from the awesome MyEtherWallet team. Please visit their well-written guide on gas here (https://myetherwallet.groovehq.com/knowledge_base/topics/what-is-gas). They also have an excellent utilities page that allows you to convert amounts of ether into subunits (https://www.myetherwallet.com/helpers.html).

You can think of the gas limit like the amount of liters/gallons/units of gas for a car. You can think of the gas price as the cost of that liter/gallon/unit of gas.

With a car, it's $2.50 (price) per gallon (unit). With Ethereum, it's 20 GWEI (price) per gas (unit). To fill up your "tank", it takes... - 10 gallons at $2.50 = $25 - 21000 units of gas at 20 GWEI = 0.00042 ETH.

Therefore, the total TX fee will be 0.00042 Ether.

Sending tokens will typically take ~50000 gas to ~100000 gas, so the total TX fee increases to 0.001 ETH - 0.002 ETH.

What is “block gas limit”?

Block gas limits are the maximum amount of gas allowed in a block to determine how many transactions can fit into a block. For example, let’s say we have 5 transactions where each transaction has a gas limit of 10, 20, 30, 40, and 50. If the block gas limit is 100, then the first four transactions can fit in the block. Miners decide which transactions to include in a block. A different miner could try including the last 2 transactions in the block (50+40), and they only have space to include the first transaction (10). If you try to include a transaction that uses more gas than the current block gas limit, it will be rejected by the network and your Ethereum client will give you the message “Transaction exceeds block gas limit”. Credit for this example comes from this Ethereum StackExchange post by “ethers” (https://ethereum.stackexchange.com/questions/7359/are-gas-limit-in-transaction-and-block-gas-limit-different).

The block gas limit is currently 4,712,357 gas at the time of writing according to ethstats.net (https://ethstats.net/), meaning around 224 transactions that each have a transaction gas limit of 21000 can fit in 1 block (which is produced every 15-20 seconds on average). The protocol allows the miner of a block to adjust the block gas limit by a factor of 1/1024 (0.0976%) in either direction (https://www.reddit.com/r/ethereum/comments/6g6tww/there_are_hundreds_or_even_thousands_of_pending/dinzrgq/).

Who decides what the block gas limit is?

Miners on the network decide what the block gas limit is. Separate from the adjustable protocol block gas limit is a default mining strategy of a minimum block gas limit of 4,712,388 for most clients (https://github.com/search?q=org%3Aethereum+4712388&type=Code). Miners can choose to change this, but many of them do not and leave the default.

How is the block gas limit changed?

Miners on Ethereum use a mining program, such as ethminer (https://github.com/ethereum-mining/ethminer), which connects to a geth (https://geth.ethereum.org/) or Parity Ethereum client node. geth and Parity have options that miners are able to change. geth’s command line options for mining are list here (https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options) and Parity’s options are here (https://github.com/paritytech/parity/wiki/Configuring-Parity#cli-options).

What is a “DoS” of the Ethereum network?

Recently there have been many comments about the Ethereum network slowing down, becoming clogged, or becoming unusable. These comments describe this slow-down as a “DoS” of the Ethereum network. A denial of service (DoS) incident on the Ethereum network happens when there are consistently full blocks and many pending transactions on the network. Recall that miners can choose to include transactions based on the transaction fee attached. If there are hundreds of thousands of transaction in queue (or as it is technically termed, the transaction pool) it can cause unusual transaction delays of hours. DDoS incidents can be malicious or non-malicious.

Malicious DoS

Last fall Ethereum was attacked by a person or group in what was called a transaction spam attack. The attack is described in this blog post (https://blog.ethereum.org/2016/10/18/faq-upcoming-ethereum-hard-fork/):

> *The attacker performed a DoS attack by repeatedly calling certain operation codes (opcodes) in their smart contracts that are computationally difficult for clients to process, but very cheap to add to the network.*
>
> *During the attack miners were asked to lower the block gas limit to 1.5 million (https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/) and then to 2 million in another instance (https://www.reddit.com/r/ethereum/comments/58aelh/attention_miners_recommending_miners_lower_the/). There have been other instances where there has been requests for the miners to lower the block gas limit during attacks on the network.*

Non-Malicious DoS

Non-malicious DoS incidents are simply when the network has so many pending transactions that it takes an unusually long time to process a transaction. Recently the popularity and proliferation of token distribution events (or initial coin offerings (ICOs) or token sales) have caused the network to become backed up with transactions. The folks at Infura wrote a blog about the technical details (https://blog.infura.io/when-there-are-too-many-pending-transactions-8ec1a88bc87e).

Why is the block gas limit not changing even when blocks are full?

Primary Reason: Miners are not using the adaptive gas limit feature.

The Ethereum protocol has a built in mechanism where miners can vote on the gas limit, and so capacity can be increased without having to coordinate on a hard fork. Originally, this mechanism was coupled with a default strategy where miners would vote on a gas limit which is at least 4.7 million, but which would target 150% of the recent (1024-block exponential moving) average gas used if that amount is higher, allowing capacity to organically increase with growing demand while still including a ceiling for anti-spam purposes.

As described in the "Malicious DoS" section above there were multiple times in the past where miners were suggested to change their settings to set the block gas limit to non-default settings to help deter attacks until fixes were developed. The problem is that some mining pools never changed the settings back even after the attacks subsided. About a month ago miners were asked to change the gas limit and gas price settings to re-introduce an adaptive gas limit feature (https://www.reddit.com/r/ethereum/comments/6ehp60/recommendations_to_miners_to_change_gas_limit_and/) because the recent token sales were quickly filling blocks and causing blockchain transaction congestion.

ETH Gas Station (http://ethgasstation.info/index.php) is an excellent resource for people looking for the latest information on what block gas limits mining pools are voting for (http://ethgasstation.info/minerVotes.php).

What do miners need to do to fix this?

Miners can adjust their settings on their geth or Parity client to re-enable adaptive gas limits. Note: The values below are taken from this Reddit post (https://www.reddit.com/r/ethereum/comments/6ehp60/recommendations_to_miners_to_change_gas_limit_and/) and can actually be set much higher, as explained in this Reddit post (https://www.reddit.com/r/ethereum/comments/6jn6lr/miners_increase_the_capacity_of_the_ethereum/).

Geth

Suggested setting

```
--gasprice 4000000000 --targetgaslimit 4712388
```

Explanation

`--targetgaslimit` Target gas limit sets the artificial target gas floor for the blocks to mine (default: “4712388”) `--gasprice` Minimal gas price to accept for mining a transactions (default: “20000000000”). Note: `gasprice` is listed in wei (https://www.myetherwallet.com/helpers.html).

Parity

Suggested setting

```
--gas-floor-target 4712388 --gas-cap 9000000 --gasprice 4000000000
```

Explanation

`--gas-floor-target` Amount of gas per block to target when sealing a new block (default: 4700000).

`--gas-cap` A cap on how large we will raise the gas limit per block due to transaction volume (default: 6283184).

`--gasprice` Minimum amount of Wei per GAS to be paid for a transaction to be accepted for mining. Note: `gasprice` is listed in wei (https://www.myetherwallet.com/helpers.html). Note 2: `--gasprice` is a “Legacy Option”

Other Mining Options

Please visit the CLI options pages for geth (https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options) and Parity (https://github.com/paritytech/parity/wiki/Configuring-Parity#cli-options) to see the full list of options miners can set to optimally adjust their settings.

Resources and Further Reading

- Eth Gas Station website with Ethereum gas and miner stats. (http://ethgasstation.info)
- Ethereum StackExchange for technical questions of all kinds. (https://ethereum.stackexchange.com/)
- “Ethereum Gas, Fuel and Fees” by Joseph Chow. (https://media.consensys.net/ethereum-gas-fuel-and-fees-3333e17fe1dc)
- “What is Gas?” by MyEtherWallet (https://myetherwallet.groovehq.com/knowledge_base/topics/what-is-gas).
- MyEtherWallet Ether unit conversion tool. (https://www.myetherwallet.com/helpers.html)
- “When there are too many pending transactions” by Infura. (https://blog.infura.io/when-there-are-too-many-pending-transactions-8ec1a88bc87e)

Posted

(ht… (ht… (ht… (ht…