BLOCKCHAIN: SMART CONTRACTS LECTURE 2 - ETHEREUM – THE BASICS

FLORIN CRACIUN

IMPORTANT

Some of the following slides are the property of

Dr. Emanuel Onica & Dr. Andrei Arusoaie

Faculty of Computer Science,

Alexandru Ioan Cuza University of Iași

and are used with their consent.

CONTENTS

WHAT IS ETHEREUM?

Quick history:

- Initial idea by Vitalik Buterin in 2013 – add programmability support to blockchain
- Launched as another Proof-of-Work based blockchain platform started in 2015
- Yellow paper by Gavin Wood constantly updated at: https://ethereum.github.io/yellowpaper

Same baseline characteristics as majority of PoW blockchains:

- Decentralized – no single point of trust
- Transactions grouped in chained blocks
- Mining (i.e., finding a specific hash value) used for validating and integrating blocks into blockchain
- Transparent and immutable blockchain ledger

First platform classified as part of the Blockchain 2.0 evolution:

not limited anymore to currency exchanges

ACCOUNTS AND TRANSACTIONS

```
17
18      /// Create a new ballot with $(_numProposals) different proposals.
19      constructor(uint8 _numProposals) public {
20          chairperson = msg.sender;
21          voters[chairperson].weight = 1;
22          proposals.length = _numProposals;
23      }
24
25      /// Give $(toVoter) the right to vote on this ballot.
26      /// May only be called by $(chairperson).
27      function giveRightToVote(address toVoter) public {
28          if (msg.sender != chairperson || voters[toVoter].voted) return;
29          voters[toVoter].weight = 1;
30      }
31
```

Main novelty: support for *programmable transactions*

- Transactions can trigger custom code execution: essentially a transaction can be considered as a function invocation
- Transactions execution can be affected by implemented conditions
- Functions invoked by transactions can have related logic and are grouped into small programs: *smart contracts*
- Main language to code smart contracts: Solidity (more on it in the next lecture)
- Smart contracts can interact between them and be organized as a backend for decentralized applications (DApps)

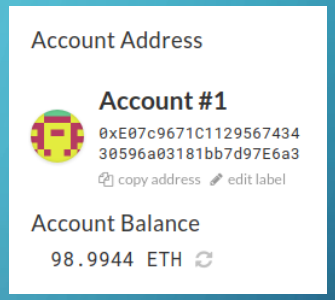
Ethereum still retains the use of a cryptocurrency: transactions have a cost and can also be simply used for transferring funds

The Ethereum currency is named *Ether* (more on it later).

ACCOUNTS AND TRANSACTIONS

Accounts in Ethereum: model *users* and *contracts* as entities that interact through transactions

Two types of accounts:



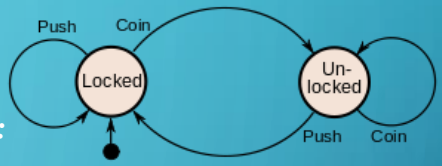
- Externally owned Accounts (EOA) – associated to users
  - A public/private key pair is generated at creation
  - The private key is required to initiate transactions
  - Do not have any code associated with them
  - Typically used through *wallet* applications (keys maintenance and connection to network; user operations: contract deployments, transactions initiation)
- Contract accounts – associated to contracts
  - Controlled by their contract code
  - Cannot initiate transactions except as triggered by received transactions

Each account is identified by a 20 bytes address (40 hexa chars):

- EOAs: address derived from the public key
- Contract accounts: address derived from the contract creator address

ACCOUNTS AND TRANSACTIONS

Ethereum functions like a *transactions-based replicated state machine:*



- A state is maintained both for its users accounts as well as for the contract accounts
- The „world state" is formed of all account states and is stored in a backend database (typically leveldb) having a particular tree structure
- Only a root hash of this world state is effectively kept in each block of the blockchain
- Executed transactions update the accounts states (and consequently also the root hash)

ACCOUNTS AND TRANSACTIONS

Account states information include:

- *Nonce* – the number of transactions initiated for an EoA or the number of other contracts creations for a contract account
- *Balance* – the currency value owned by this account (in Wei, an Ether subdivision)
- *StorageRoot* – the hash of the root node of another tree structure keeping the storage for the account
- *CodeHash* – the hash of the account's associated code in case of a contract account

ACCOUNTS AND TRANSACTIONS

Several types of transactions are possible:

- Transactions from EoA and contract accounts to EoAs transferring balance value
- Transactions from EoA and contract accounts to contract accounts, invoking a function and/or transfering balance value
- Transactions from EoA and contract accounts to contract accounts for invoking a function without balance value transfer
- Transactions from EoA and contract accounts for new contracts creation

By definition, any transaction modifies a state and has an execution cost (will come back to this)

Calls that do not modify a state (i.e., read-only functions or reading a value in a contract) are not considered transactions and are not included in blocks and verified by the network

ACCOUNTS AND TRANSACTIONS

Each executed transaction generates a *transaction receipt* including various information about it:

- the initiator account address
- the address of the destination account (empty in case of new contract construction transactions)
- the balance value transferred with the transaction
- the execution cost of the transaction
- the hash of the transaction
- the number of the block where the transaction was included
- and other logged information about the transaction

ACCOUNTS AND TRANSACTIONS

Who effectively executes the transactions?

The Ethereum Virtual Machine (EVM):

- Resides on each Ethereum node capable of mining
- Executes the functions invoked by transactions in its own low-level stack-based language – the EVM bytecode
- Smart contracts are developed in high level languages compiled to EVM bytecode
- Most popular high level languages (currently):
  - Solidity – focus in this course
  - Vyper

ETHER AND GAS

Ether (ETH) is the currency of Ethereum having many division units as well as higher denominations. The smallest and most used in practice is Wei (1 ETH = 10^{18} Wei).

Name	Value (in Wei)	
Wei	**1**	**10^0**
Kwei/Ada	1000	10^3
Mwei/Babbage	1000000	10^6
Gwei/Shannon	1000000000	10^9
Microether/Szabo	1000000000000	10^{12}
Milliether/Finney	1000000000000000	10^{15}
Ether (ETH)	**1000000000000000000**	**10^{18}**
Kether/Einstein	1000000000000000000000	10^{21}
Mether	1000000000000000000000000	10^{23}
Gether	1000000000000000000000000000	10^{26}
Tether	1000000000000000000000000000000	10^{29}

ETHER AND GAS

Each transaction in Ethereum has a cost. Why?

The basic answer: to prevent network abuse

- Flooding the network with bogus transactions
- Executing very demanding code might result in Denial of Service (remember that transactions are function invocations)
- Preventing excessive storage on the network

Transaction cost is expressed in *gas* units.

Each operation triggered by a transaction has a specific gas cost.

Informally the total transaction cost can be split in two parts:

- Cost of submitting the transaction
- Cost of executing the transaction (includes storage cost)

ETHER AND GAS

The cost of submitting a transaction:

- A base fee:
  - 21000 gas for regular transactions
  - 32000 gas for contract creation transactions
- A fee for any data associated to the transaction (e.g., byte encoding of invoked function signatures and arguments in contracts, or contract code for deployment):
  - 4 gas for each zero byte
  - 68 gas for each non-zero byte

The cost of executing the transaction:

- Varies based on the base cost of the operations executed by the EVM

ETHER AND GAS

Appendix G. Fee Schedule

The fee schedule G is a tuple of 31 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_{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}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	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.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	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 $\lceil \log_{256}(exponent) \rceil$ 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.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	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_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.
$G_{quaddivisor}$	20	The quadratic coefficient of the input sizes of the exponentiation-over-modulo precompiled contract.

ETHER AND GAS

- Typically development environments are able to estimate the complete gas cost for a transaction
- The effective transaction price is computed based on the price per unit of gas:

$$TX\_price = TX\_gas\_cost \times gas\_unit\_price$$

- When submitting a transaction a user is typically required to provide two values:
  - Gas limit – an estimation of the maximum amount of gas to be spent in executing the transaction
  - Gas price – the price per unit of gas that the user allocates

ETHER AND GAS

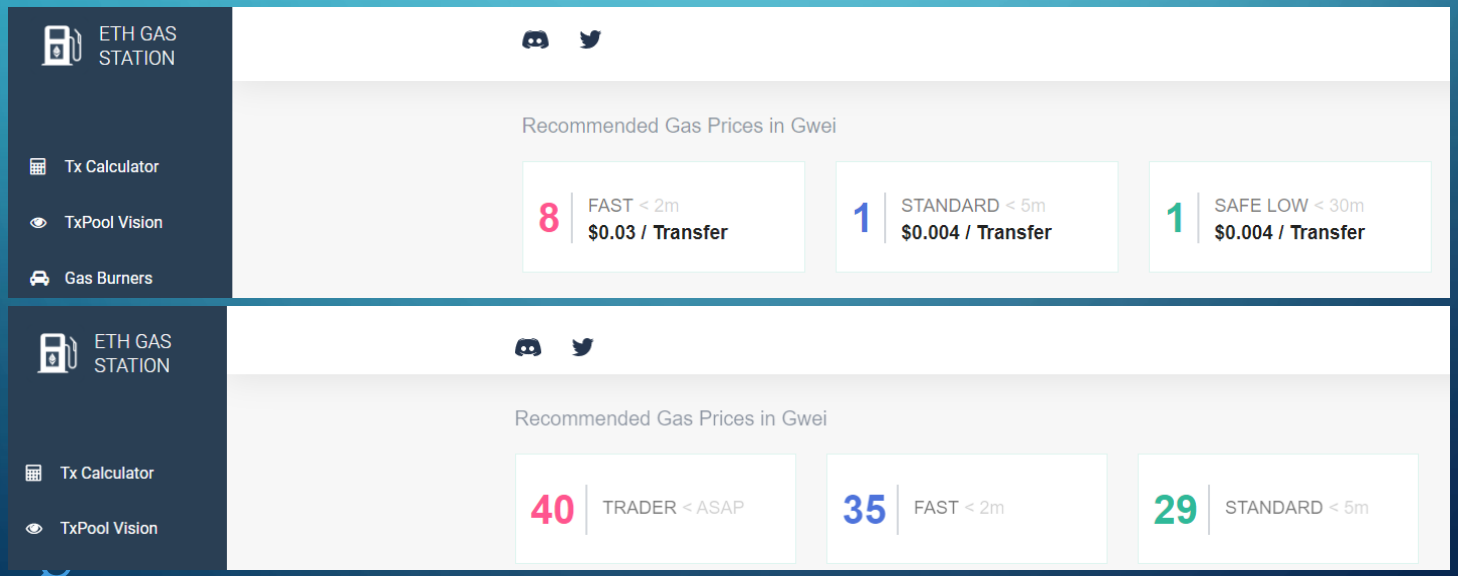
- The gas used for transaction submitting and by each operation will be deducted during execution from the the gas limit provided by the user

The following situations are possible:

- TX_gas_cost > gas_limit
  - Transaction fails due to insufficient gas allocated
  - Any steps executed for the transaction will be reverted
  - The price of spent gas will be deducted from the initiating account balance
- TX_gas_cost <= gas_limit
  - The transaction will succeed (if no other error appears)
  - Any leftover gas up to the the gas_limit will be returned to the initiating account balance at the price rate used initially

ETHER AND GAS

- The gas unit price is not a fixed value and fluctuates depending on the market
- The gas fees paid for transactions are received by miners that include the transaction in a successfully mined block (+ 2 ETH as block reward at the current time)
- It's more likely to have the transactions picked quicker for mining if the allocated gas price is higher
- If the total demand to mine transactions is raising, the gas unit price will likely grow



Gas price October 2019 vs October 2020 - Figure source: *http::/ethgasstation.info*

ETHER AND GAS

- Calls that are not transactions per se (are read-only and do not modify state), might still require computation
- A gas limit and a gas price must still be provided as in the case of transactions
- Read-only calls are, however, normally processed on a local copy of the blockchain state
- Therefore, not being state-changing and „mined", the call price is returned
- Read-only calls can still fail though due to „out-of-gas" exception if the gas limit set is too low

- Why proceed like this for „free" calls?
  - Read-only calls can also be initiated as subsequent calls to effective transactions (i.e., invoked as part of a function)
  - In such case their price for execution is charged normally

ETHER AND GAS

- Mined blocks have a gas limit set for the cost sum of all included transactions
- A miner cannot bypass this limit (i.e., include more transactions in a block)

- Various reasons:
  - Diminish impact of varying mining power distribution (i.e., limit the gains/influence of more powerful miners)
  - Mined blocks that are too large take longer to be propagated into the blockchain network and validated

BLOCKS IN ETHEREUM

- The blockchain in Ethereum uses an efficient storage structure with the form of a Merkle-Patricia tree (or trie) to store the accounts state
- In essence, each node of such a tree is a hash computed over its child nodes, the effective information being stored in leafs
- A first tree includes in leafs the accounts state information (presented earlier)
- The hash root of this tree is the only part included in the effective block (referred sometimes as the „block header")
- Each account state leaf includes also the hash root of another similar tree holding the storage information for that account

BLOCKS IN ETHEREUM

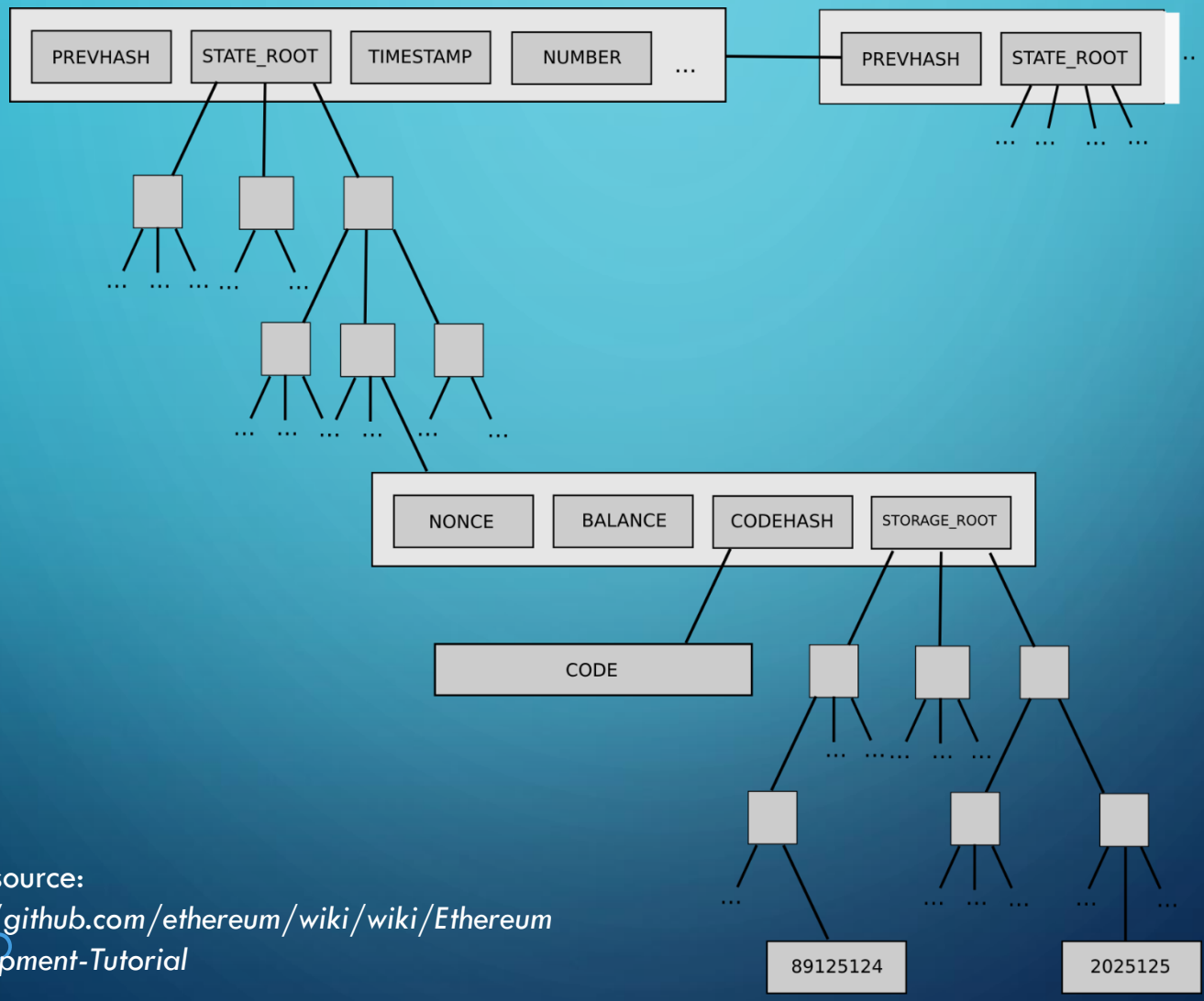


Figure source:
https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial

BLOCKS IN ETHEREUM

- The particular structure of the Patricia tree ensures small differences following each transaction and resulted new blocks
- Most of the tree remains unchanged between two generated blocks
- Only the root in block header is always updated
- Most of tree nodes are pointing back to the memory addresses of the similar nodes in the previous tree
- The historic states (e.g., values in storage nodes of an account were modified) are normally periodically pruned to save space (these can be re-created by running the transactions from the beginning of the chain)

BLOCKS IN ETHEREUM

- A more complete overview of the information included in a block header is the following:

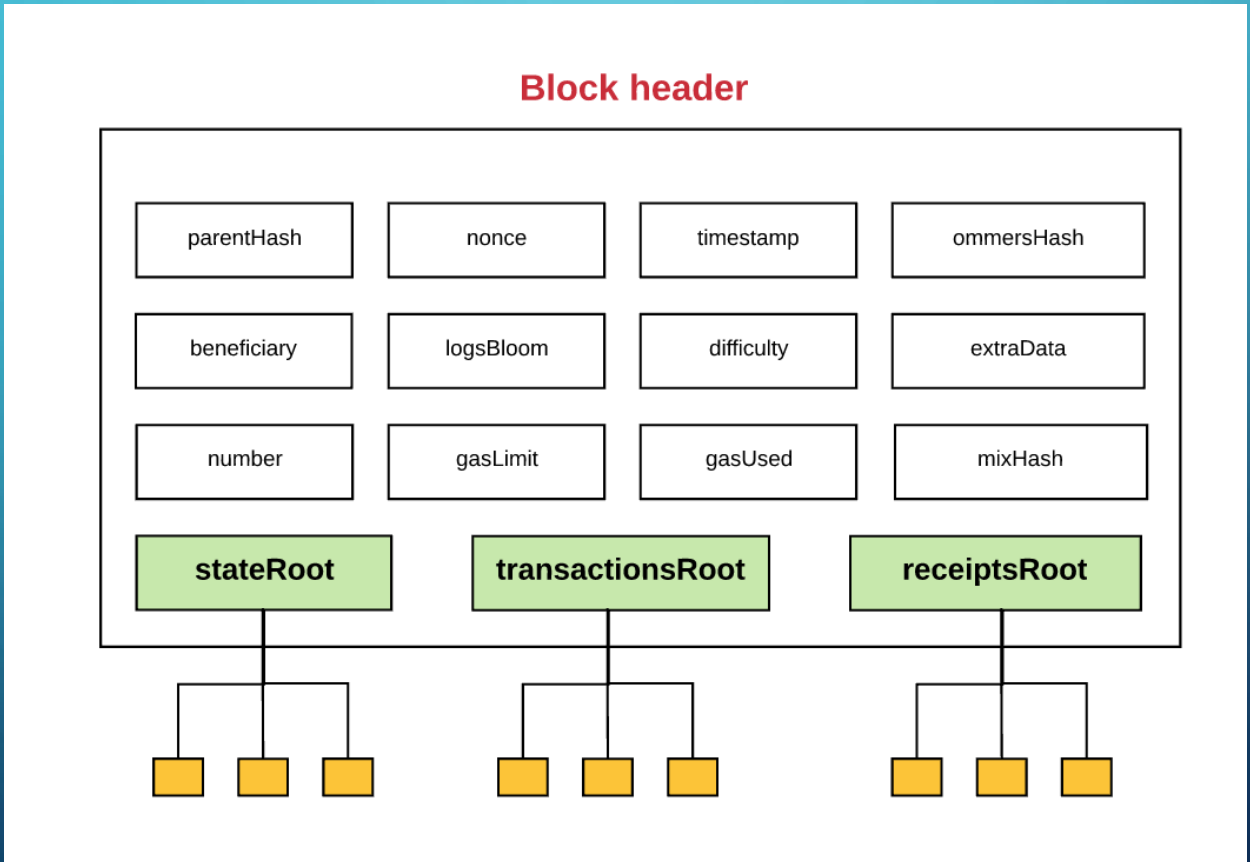


Figure source: *How does Ethereum Work Anyway?* (Preethi Kasireddy – *https://medium.com*)

BLOCKS IN ETHEREUM

- parentHash: the hash of the previous block header
- nonce and mixHash: used in the Proof-of-Work validation of the block
- timestamp: the timestamp indicating when the block was mined
- ommersHash: the hash of the block's ommers list (an ommer is a block having as parent the current block's grandparent – i.e., a brother of the parent that didn't get into the chain); more about it in a future mining lecture
- beneficiary: the address receiving the reward and fees for mining the block
- logsBloom: a Bloom filter summarizing log information of transactions in the block

BLOCKS IN ETHEREUM

- difficulty: difficulty level of mining the block
- extraData: extra data related to the block (max 32 bytes)
- number: a block counter starting from 0 – genesis block
- gasLimit: the gas limit for this block as previously described
- gasUsed: the total gas used in the transactions included in this block
- stateRoot: the root hash of the accounts state tree
- transactionsRoot: the root hash of a similar Merkle-Patricia tree used for storing the transactions
- receiptsRoot: the root hash of a similar Merkle-Patricia tree used for storing transactions receipts

NODES AND NETWORKS

The following types of nodes can be part of an Ethereum network depending on the way they store the blockchain structure:

- Full node:
  - stores the full blockchain data
  - can participate in mining, execute transactions and verify all blocks and states
  - can derive all previous states from the blockchain structure
  - implementations sometimes have optimized modes for full synchronization when joining the network (i.e., by fetching headers first and filling block bodies – the trees – later)
- Light node:
  - stores only the headers and initially just for a limited amount of the most recent blocks
  - cannot mine or execute transactions
  - can perform verifications over the state roots in headers and request more chain data on demand
- Archive node: a full node that also builds and stores hystorical states

NODES AND NETWORKS

There is one public main Ethereum network and multiple public Ethereum test networks:

- The main Ethereum network has the blockchain ID 1, and currently uses Proof-of-Work for consensus
- The most important active test networks are:
  - Ropsten (ID 3) using Proof-of-Work for consensus
  - Rinkeby (ID 4) using Proof-of-Authority for consensus
  - Kovan (ID 42) using Proof-of-Authority for consensus
- Some test networks support only some client implementations
- The ETH mined on test networks do not have market value

https://ethstats.net/