Ethereum Blocks

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

September 3, 2019

Ethereum Block Header

Block = (Header, Transactions, Uncle Headers)

Block Header

`parentHash`	32	bytes
`ommersHash`	32	bytes
`beneficiary`	20	bytes
`stateRoot`	32	bytes
`transactionsRoot`	32	bytes
`receiptsRoot`	32	bytes
`logsBloom`	256	bytes
`difficulty`	≥ 1	byte
`number`	≥ 1	byte
`gasLimit`	≥ 1	byte
`gasUsed`	≥ 1	byte
`timestamp`	≤ 32	bytes
`extraData`	≤ 32	bytes
`mixHash`	32	bytes
`nonce`	8	bytes

Simple Fields in Block Header

parentHash	32	bytes
ommersHash	32	bytes
beneficiary	20	bytes
stateRoot	32	bytes
transactionsRoot	32	bytes
receiptsRoot	32	bytes
logsBloom	256	bytes
difficulty	≥ 1	byte
number	≥ 1	byte
gasLimit	≥ 1	byte
gasUsed	≥ 1	byte
timestamp	≤ 32	bytes
extraData	≤ 32	bytes
mixHash	32	bytes
nonce	8	bytes

- parentHash = Keccak-256 hash of parent block header
- beneficiary = Destination address of block reward and transaction fees
- stateRoot = Root hash of world state trie after all transactions are applied
- transactionsRoot = Root hash of trie populated with all transactions in the block
- number = Number of ancestor blocks
- timestamp = Unix time at block creation
- extraData = Arbitrary data; Miners identify themselves in this field

gasLimit and gasUsed

`parentHash`	32	bytes
`ommersHash`	32	bytes
`beneficiary`	20	bytes
`stateRoot`	32	bytes
`transactionsRoot`	32	bytes
`receiptsRoot`	32	bytes
`logsBloom`	256	bytes
`difficulty`	≥ 1	byte
`number`	≥ 1	byte
`gasLimit`	≥ 1	byte
`gasUsed`	≥ 1	byte
`timestamp`	≤ 32	bytes
`extraData`	≤ 32	bytes
`mixHash`	32	bytes
`nonce`	8	bytes

- `gasUsed` is the total gas used by all transactions in the block
- `gasLimit` is the maximum gas which can be used
- $|\texttt{gasLimit} - \texttt{parent.gasLimit}| \leq \frac{\texttt{parent.gasLimit}}{1024}$
- Miner can choose to increase or decrease the `gasLimit`

logsBloom and receiptsRoot

`parentHash`	32	bytes
`ommersHash`	32	bytes
`beneficiary`	20	bytes
`stateRoot`	32	bytes
`transactionsRoot`	32	bytes
`receiptsRoot`	32	bytes
`logsBloom`	256	bytes
`difficulty`	≥ 1	byte
`number`	≥ 1	byte
`gasLimit`	≥ 1	byte
`gasUsed`	≥ 1	byte
`timestamp`	≤ 32	bytes
`extraData`	≤ 32	bytes
`mixHash`	32	bytes
`nonce`	8	bytes

- Bloom filter = Probabilistic data structure for set membership queries
  - **Query:** Is x in the set? **Response:** "Maybe" or "No"
- `receiptsRoot` is the root hash of transaction receipts trie
  - Each transaction receipt contains Bloom filter of addresses and "topics"
- `logBloom` is the OR of all transaction receipt Bloom filters
- Light clients can efficiently retrieve only transactions of interest

Mining

Ethash Mining Algorithm

- An epoch lasts 30,000 blocks
- Epoch index EI = block_number / 30000
- At an epoch beginning
  - A list called cache of size $\approx 2^{24} + EI \times 2^{17}$ bytes is created
  - A list called dataset of size $\approx 2^{30} + EI \times 2^{23}$ bytes is created
- The dataset is also called the DAG (directed acyclic graph)

Block Number	Epoch	Cache Size	DAG Size	Start Date
30000	1	16 MB	1 GB	17 Oct, 2015
3840000	128	32 MB	2 GB	21 Jul, 2017
7680000	256	48 MB	3 GB	30 Apr, 2019
192000000	640	96 MB	6 GB	25 Aug, 2024

Source: `https://investoon.com/tools/dag_size`

- Mining nodes need to store full dataset (ASIC resistance)
- Light nodes store cache and recalculate specific dataset items

Cache Generation

- The cache is initialized by repeatedly hashing a seed (deriving from the block headers)
- Two rounds of a function called randmemohash are applied

```
1   HASH_BYTES = 64
2   CACHE_ROUNDS = 3
3
4   def mkcache(cache_size, seed):
5     n = cache_size // HASH_BYTES
6
7     # Sequentially produce the initial dataset
8     o = [sha3_512(seed)]
9     for i in range(1, n):
10        o.append(sha3_512(o[-1]))
11
12    # Use a low-round version of randmemohash
13    for _ in range(CACHE_ROUNDS):
14        for i in range(n):
15            v = o[i][0] % n
16            o[i] = sha3_512(map(xor, o[(i-1+n) % n], o[v]))
17
18    return o
```

Dataset Generation

```
HASH_BYTES = 64
WORD_BYTES = 4
DATASET_PARENTS = 256

FNV_PRIME = 0x01000193

def fnv(v1, v2):
    return ((v1 * FNV_PRIME) ^ v2) % 2**32

def calc_dataset_item(cache, i):
  n = len(cache)
  r = HASH_BYTES // WORD_BYTES
  # initialize the mix
  mix = copy.copy(cache[i % n])
  mix[0] ^= i
  mix = sha3_512(mix)
  # fnv it with a lot of random cache nodes based on i
  for j in range(DATASET_PARENTS):
      cache_index = fnv(i ^ j, mix[j % r])
      mix = map(fnv, mix, cache[cache_index % n])
  return sha3_512(mix)
```

Ethash Mining Algorithm

`parentHash`	32	bytes
`ommersHash`	32	bytes
`beneficiary`	20	bytes
`stateRoot`	32	bytes
`transactionsRoot`	32	bytes
`receiptsRoot`	32	bytes
`logsBloom`	256	bytes
`difficulty`	≥ 1	byte
`number`	≥ 1	byte
`gasLimit`	≥ 1	byte
`gasUsed`	≥ 1	byte
`timestamp`	≤ 32	bytes
`extraData`	≤ 32	bytes
`mixHash`	32	bytes
`nonce`	8	bytes

- Cache calculation involves hashing previous cache elements pseudorandomly
- Every dataset element involves hashing 256 pseudorandom cache elements
- Mining loop takes partial header hash, `nonce`, and dataset as input
- 128 dataset elements are used to create 256-bit `mixHash`

  Mining output $=$ Keccak256 (Keccak512(`HdrHash`$\|$`nonce`)$\|$`mixHash`)

Mining Difficulty

`parentHash`	32	bytes
`ommersHash`	32	bytes
`beneficiary`	20	bytes
`stateRoot`	32	bytes
`transactionsRoot`	32	bytes
`receiptsRoot`	32	bytes
`logsBloom`	256	bytes
`difficulty`	≥ 1	byte
`number`	≥ 1	byte
`gasLimit`	≥ 1	byte
`gasUsed`	≥ 1	byte
`timestamp`	≤ 32	bytes
`extraData`	≤ 32	bytes
`mixHash`	32	bytes
`nonce`	8	bytes

- Proof of work is valid if `mixhash` and `nonce` lead to

$$\text{Keccak256}\left(\text{Keccak512}(\texttt{HdrHash}\|\texttt{nonce})\|\texttt{mixHash}\right) \leq \frac{2^{256}}{\texttt{difficulty}}$$

- Partial validation of PoW in block can be done without DAG or cache

References

- Yellow paper `https://ethereum.github.io/yellowpaper/paper.pdf`
- Light client protocol `https://github.com/ethereum/wiki/wiki/Light-client-protocol`
- Ethash `https://github.com/ethereum/wiki/wiki/Ethash`
- Randmemohash `http://www.hashcash.org/papers/memohash.pdf`