MicroBitcoin: decentralized peer-to-peer payment platform for the micro-economy

Abstract. MicroBitcoin is decentralized blockchain intended to serve for micro-economy payments. It inherits Bitcoin UTXO set and initially has been implemented as an hard fork. After more than one year after launch some limitations began to arise, in particular extensive size of blockchain inherited from Bitcoin network and poor performance of PoW algorithm during block validation. To solve those issues on 9 October 2019 community switched to new network essentially abandoning old one. New MicroBitcoin network is featuring UTXO set snapshot, smaller block size, new block reward formula and cpu focused Proof-of-Work algorithm.

1. Prerequisites of new network launch

Initially original MicroBitcoin network has been launched 11 July 2018 as an hard fork of Bitcoin network. Main focus was on ASIC[1] resistance and faster block time to be more suitable for micro-payments. To make interaction with currency units easier decimal poin was shifted by 4 places making 1 BTC equal to 10,000 MBC.

First MicroBitcoin block was mined at 11 July 2018 causing hardfork by replacing default sha256d hash function with NIST SHA-3 candidate Groestl[2] algo which didn't had ASIC implementation at the time and because of that was considered ASIC resistant. Time proven that this assumption was wrong after Baikal released[3] BK-G28 featuring Groestl support on 26 October 2018. Since this time BK-G28 miners had been main source of hash power on MicroBitcoin network fundamentally corrupting decentralization. After extensive research we stopped on Rainforest[4] PoW algo by Bill Schneider. On 6 March 2019 MicroBitcoin network hardforked to Rainforest and on 7 May 2019 to second version of Rainforest (also known as RFv2) which fixed some flaws of original algo.

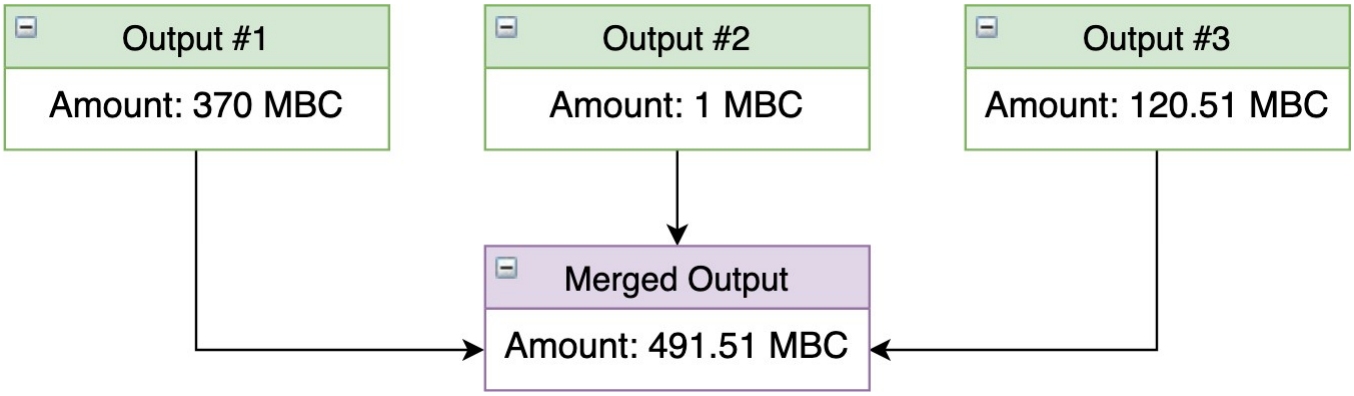
After a while it became clear that Rainforest v2 algorithm is way to slow during PoW validation phase and in combination with more than 200 GB of blockchain size make it very hard to sync/keep full node of MicroBitcoin essentially undermining decentralization. This situation became the main reason behind launch of new network.

2. Snapshot

Since MicroBitcoin network operates on UTXO[5] model where final address balance is basically sum of all unspent outputs, moving balances from one network to another is rather trivial task.

We took all UTXOs starting from block **525,000** (first MBC block) to block **1,137,200**, copied them and merged. For example if address had 3 unspent outputs in old network, they had been merged into one output with sum of amounts.

Example:



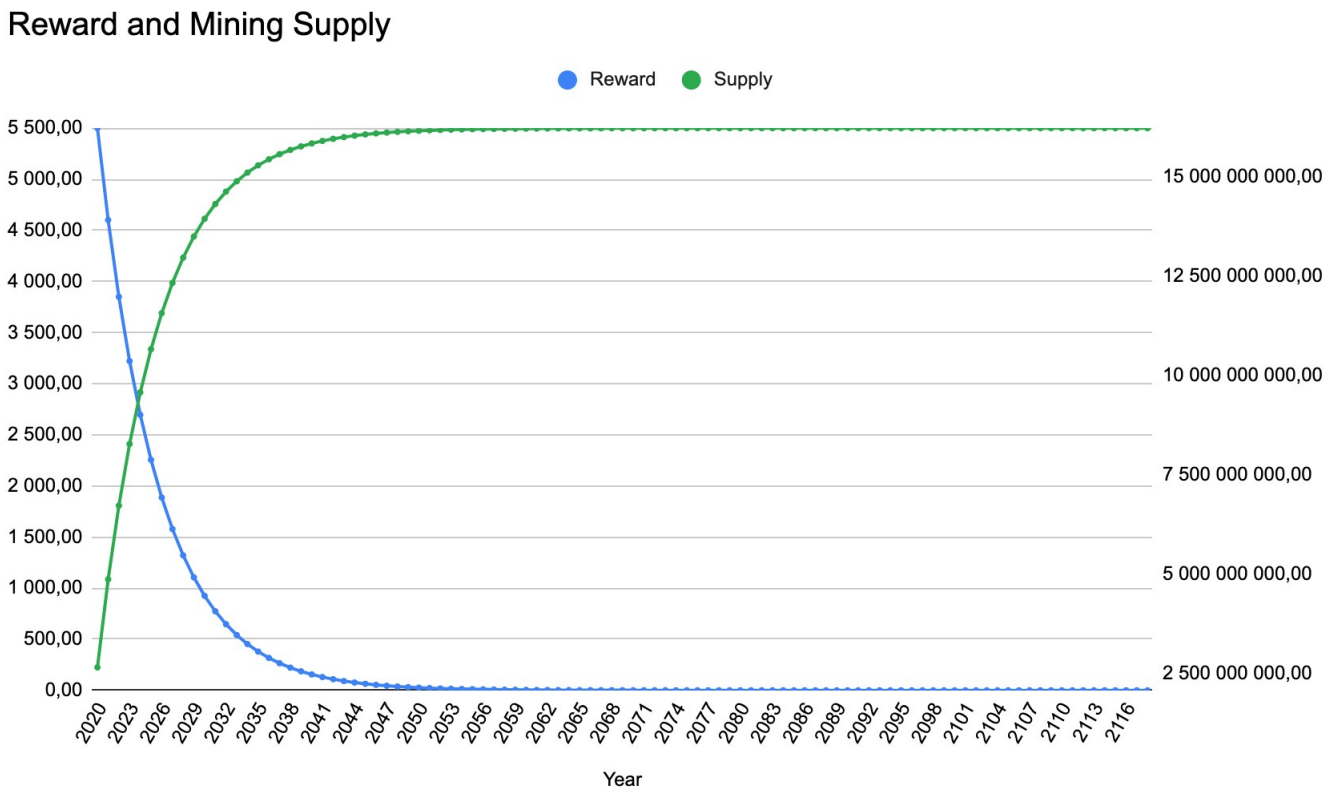
All snapshoted outputs is located in genesis[6] block of new MicroBitcoin network and can be checked in explorer.

3. Supply and emission

At the moment of new network launch total supply was over-minted for the current userbase, big chunk of funds haven't been moved since hard fork. To improve this situation coins which haven't been moved since block **525,000** (initial network launch height) hasn't been snapshoted and essentially burned. In total **44,386,397,362.4252** MBC has been activated. Approximately **2,700,000** BTC has moved since hard fork.

For better distributrion of new coins block emission schedule has been adjusted. Instead of halvings[7] which reduces block reward by 50% each 4 years new reward smoothly decrease each new block reward. Base reward is decaying by 30% each epoch which is around 2 years.

Graph for reward and mining supply:



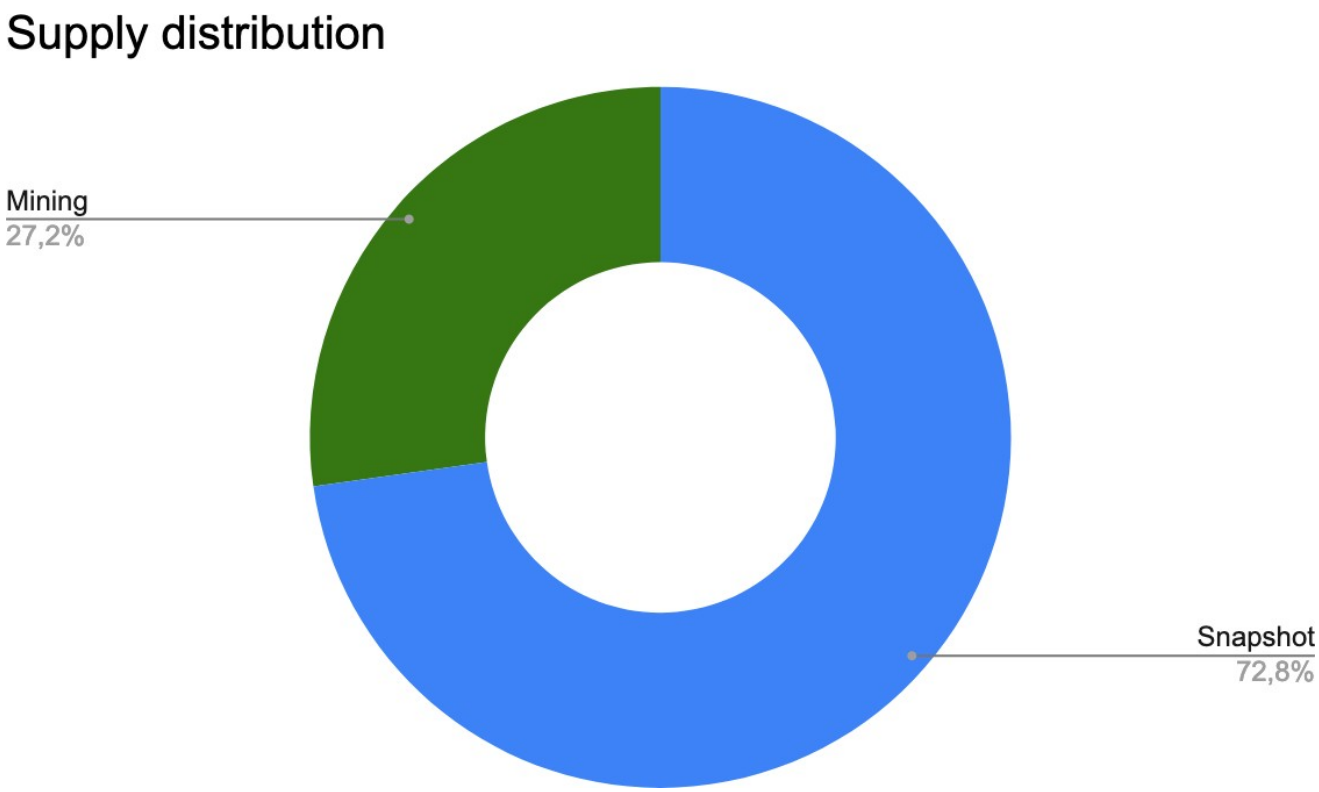
Reward formula implementation in C++.

```
#include <iostream>
#include <cmath>

// Amounts of satoshit per coin
const int64_t COIN = 10000;

int64_t reward(int height) {
  // Initial reward per block
  const int64_t reward = 5500 * COIN;
  // Reward decreasing epoch (2 years)
  const int epoch = 525960 * 2;
  // Decrease amount by 30% each epoch
  const long double r = 1 + (std::log(1 - 0.3) / epoch);
  return reward * std::pow(r, height);
}
```

Total supply is limited to **61,000,000,000 MBC** from which **44,386,397,362.4252** MBC is snapshot amount from old network. The rest **16,613,602,638** MBC will be mined in around next 100 years.



4. Block size

To make network more reliable, prevent block spamming and create better and fair fee market in terms of 1 block per minute model block size has been decreased to **300kb**. Implementation is inspired by Bitcoin Core developer Luke Dashjr proposal[8].

5.Power2B Proof-of-Work algorithm

To encourage decentralization and idea of "one-CPU-one-vote" proposed[9] by Satoshi in original whitepaper of Bitcoin we used modified YesPower[10] hash function called Power2B[11] which was designed to be CPU-friendly, GPU-unfriendly, and FPGA/ASIC-neutral. It combines computationally expensive and sequential memory-hard hashing in a way that slows down GPUs to CPU-like speeds, and limits potential advantages for FPGAs and ASICs. So far YesPower proven to be decent CPU focused algorightm by providing security for dozens different cryptocurrencies.

Our Power2B modification replaces SHA256 based PBKDF2 and HMAC with blake2b[12] based implemetations in essence keeps YesPower original design intact. This has been done to make implemetations of FPGAs and ASICs for original YesPower incompatible with Power2B. This would require developers to create MicroBitcoin specific implementations of software/hardware and strengthening network security overall as an result.

6. Difficulty adjustment algorithm

MicroBitcoin network uses LWMA3[13] difficulty adjustment algorithm authored by zawy12. It sets difficulty by estimating current hashrate by the most recent difficulties and solvetimes. It divides the average difficulty by the Linearly Weighted Moving Average (LWMA) of the solvetimes. This gives it more weight to the more recent solvetimes. It is designed for small coin protection against timestamp manipulation and hash attacks. The basic equation is:

```
next_difficulty = average(Difficulties) * target_solvetime / LWMA(solvetimes)
```

7. Comparison to other Bitcoin hard forks

Here is table chart with comparison MicroBitcoin with other Bitcoin hard forks.

	Bitcoin	BitcoinCash	BitcoinGold	MicroBitcoin
Total Supply	21M	21M	21M	**61B ***
PoW Algorithm	SHA256	SHA256	Equihash	**Power2B**
Mining Hardware	ASIC	ASIC	CPU/GPU	**CPU**
Block Creation Interval	10min	10min	10min	**1min**
Difficulty Adjustment	Bitcoin DAA	SMA	LWMA2	**LWMA3**
SegWit	Yes	No	Yes	**Yes**
Block Size	1mb	32mb	1mb	**300kb**

> Keep in mind that MicroBitcoin have **4** decimal places instead of **8** like in case of Bitcoin. So in terms of Satoshi units[14] supply of MicroBitcoin is only **3x** larger than supply of Bitcoin.

8.Token Layer

Blockchain technology, due to its inherent design, presents a unique set of issues when trying to implement new features. In order to preserve consensus when adding new features, all network peers must agree on a new set of rules - a hard fork. While building the Token Layer for MicroBitcoin, we took into account our previous experiences with hard forks and decided to use another approach: subnetworks built using blockchain data embedding. This is the perfect way to introduce new features like tokens within the existing ecosystem without hard forking or introducing new breaking changes to the existing consensus.

8.1 Overview

As the network grows, new ideas for features and improvements will appear. This usually consists of modifying the underlying consensus rules and requiring the majority of the network peers to update their software. This is, at the very least, inconvenient for network members and requires investing time and effort into maintenance. Furthermore, the community may not accept the newly proposed changes, which would lead to a network split, which is considered undesirable in most circumstances as it fragments community.

Soft fork is one solution to this problem, which was introduced by Bitcoin developers. The gist of it is adding new rules to consensus, making previously valid blocks invalid, for example, by limiting block reward after a certain percentage of the network accepts new rules by updating to a new software version. While pre-soft fork software can still process blocks created by update nodes as they are still part of consensus, new nodes won't accept blocks created by old software as it breaks newly established and agreed-upon rules.
On the other side, hard forking is modifying rules in such a way that previously invalid blocks become valid, for example, by adding token functionality to the network and essentially introducing new kinds of transactions with their own structure[18].

The emergence of layer 2 networks resulted from the limitations imposed on the underlying blockchain networks and the obstacles posed by both soft and hard forks. They introduce novel features in a distinct network that is governed by distinct rules and governed by consensus, which operates independently of the base network. A notable example of such an approach is the Lightning network[19] built on top of the Bitcoin blockchain, which facilitates instant payments off-chain and uses the base network to finalize those payments by broadcasting transactions and closing the payment channel. Omni Layer[20] is another good example of this approach to introducing new features to the underlying network without breaking consensus. It works by embedding its own payloads in `OP_RETURN` output. This allows anyone to go through the Bitcoin blockchain and rebuild the current state of the Omni Layer by processing encoded payloads.

8.2 Design

Considering previous approaches of introducing new features we decided to use blockchain data embedding via `OP_RETURN` output as the basis for our Token Layer. `OP_RETURN` opcode is a standard way of attaching extra data to transactions is to add a zero-value output with a scriptPubKey consisting of `OP_RETURN` followed by data[21].

It's possible to attach up to 83 bytes of encoded payload in single transaction using this approach. For payload encoding, we decided to use Message Pack[22] as it provides a compact and efficient way to serialize and transmit data. Since we are working with rather limited storage capacity per transaction, efficient data storage is crucial.

A protocol is a set of rules and conventions that define how data is formatted, transmitted, and processed over the MicroBitcoin network. It enables all Token Layer clients to understand and interpret the information they exchange, allowing for seamless and standardized communication between them.

Token Layer works by scanning the MicroBitcoin blockchain. During this process, the client goes through each block, looking for `OP_RETURN` outputs in its transactions. If such output is found, the Token Layer client checks the first byte and compares it with the current chain ID, a unique identifier that is used to differentiate between different subnetworks. The chain ID byte is followed by a protocol payload encoded using Message Pack.

Token Layer uses Bitcoin-style satoshis to represent all amounts along with decimals specified by the token creator, which can range from `0` to `8`. For example, 10,000 TEST tokens with 2 decimals would be represented as 1,000,000 satoshis. The easiest and most obvious data type to store satoshi values in Message Pack is binary, which represents a byte array[23] since we can specify how much space we would need for our value field. We will convert integer values into a big-endian 10-byte array, which should be enough to cover most of the Token Layer requirements.

The token ticker has a set of limitations imposed on it, such as that it can only be uppercase, have Latin letters, numbers and `.` and `_` symbols. The length of the token ticker should be in the range of 3-32 characters. It also solves a couple different issues that are present in other similar solutions. For example, it can be used to represent different token types: root, sub, unique and owner.

Root token is base token type which can be used without any limitations. When root token created with `reissuable` field set to true Token Layer automatically creates owner token which is denoted by `!` symbol at the end of ticker: `TEST` (root) and `TEST!` (owner).

Owner token is used to represent ownership over root token as well as authorizing such actions as issuance of additional supply and creation of sub/unique tokens on top of root name. When issuing sub token same rules are applied.

Sub token can only be issued on top of an existing root token by its owner and is denoted by `/` symbol. As an illustrative use case, someone can issue `COMPANY` token and issue `COMPANY/STOCK` which can prove the authenticity of that token.

A unique token is used to represent non-fungible things and can only be created in 1 unit with 0 decimals, denoted by `#` symbol. As well as sub tokens, it requires a root token to be issued on top of them. An example of such an approach would be someone issuing `COLLECTION` root token and `COLLECTION#ART` on top of it.

Lastly, in order to manage such a complex system, the Token Layer has a governance system in place. It has an admin address that can issue special types of transactions that can ban, unban, change token creation or issuance costs, and update the fee address that will receive funds from token issuance. The admin address is set on network launch and can be updated only through hardfork. This system would ensure that such events as hacks and changes in underlying currency price fluctuations could be mitigated swiftly.

8.3 Protocol

Each protocol message has two permanent fields: version and category. The version field can be used in the future if modifications to the token layer protocol are needed. The category field defines what each given message does, for example, token issuance, transfer, burning, etc. Depending on the category, the message payload can contain additional fields like token metadata or the transfer amount. Unless the data structure matches all requirements for given category, the message will be discarded.

Create token message

This message is responsible for creating new tokens. It requires additional output to the governance address in order to pay the creation fee, this is done to prevent spam and token ticker squatting. Each token ticker is unique and can be used only once. Tokens ha`reissuable` field that, if set to true, creates an additional owner token, for example, if the user creates token `TEST` he would also receive owner token `!TEST`.

Category fields:

- `reissuable` : bool - defines whether owner can increase supply of this token
- `value` : bytes - token supply value encoded in bytes
- `decimals` : int - token decimal places
- `ticker` : str - token ticker

Example raw data:

```
{
    "v": b"\x00\x00\x00\x00\x00\x00\x00\x0fB@", # Value - 1000000 satoshis
    "r": False, # Reissuable - false
    "t": "TEST", # Ticker - TEST
    "d": 2, # Decimals - 2
    "c": 1, # Category - create
    "m": 1, # Version - 1
}
```

Example encoded message:
`86a176c40a000000000000000f4240a172c2a174a454455354a16402a16301a16a01`

Issue token message

This message is responsible for increasing token supply. It can be use only if token `reissuable` is set to true and user has owner token. This category required additional output to the governance address in order to pay the creation fee.

Category fields:

- `value` : bytes - token additional supply value encoded in bytes
- `ticker` : str - token ticker

Example raw data:

```
{
    "v": b"\x00\x00\x00\x00\x00\x00\x00\x0fB@", # Value - 1000000 satoshis
    "t": "TEST", # Ticker - TEST
    "c": 2, # Category - issue
    "m": 1, # Version - 1
}
```

Example encoded message:
`84a176c40a000000000000000f4240a174a454455354a16302a16a01`

Transfer token message

This category is responsible for transferring tokens between address balances. It requires additional output to the receiver address with a small marker amount, which would help Token Layer clients identify the transfer receiver. If `lock` field is set to `int` value, the receiver won't be able to spend the tokens he received until the specified height.

Category fields:

- `lock` : int or null - optional block height until which transfer will be locked and unspendable
- `value` : bytes - token transfer value encoded in bytes
- `ticker` : str - token ticker

Example raw data:

```
{
    "v": b"\x00\x00\x00\x00\x00\x00\x00\x0fB@", # Value - 1000000 satoshis
    "t": "TEST", # Ticker - TEST
    "l": 100, # Lock - block #100
    "c": 3, # Category - transfer
    "m": 1, # Version - 1
}
```

Example encoded message:
`85a176c40a000000000000000f4240a174a454455354a16c64a16303a16a01`

Burn message

This message is responsible for burning tokens on address balance. It requires no additional outputs and can be performed by any token holder.

Category fields:

- `value` : bytes - token burn value encoded in bytes
- `ticker` : str - token ticker

Example raw data:

```
{
    "v": b"\x00\x00\x00\x00\x00\x00\x00\x0fB@", # Value - 1000000 satoshis
    "t": "TEST", # Ticker - TEST
    "c": 6, # Category - burn
    "m": 1, # Version - 1
}
```

Example encoded message:
`84a176c40a000000000000000f4240a174a454455354a16306a16d01`

Cost message

This admin message is responsible for changing MBC cost of token creation/issuance. Can be used only by Token Layer governance addresses.

Category fields:

- `value` : bytes - MBC cost value encoded in bytes
- `category` : str - cost category (create/issue)
- `type` : str - token type (root/sub/unique)

Example raw data:

```
{
    "v": b"\x00\x00\x00\x00\x00\x00\x00\x0fB@", # Value - 1000000 MBC satoshis
    "a": "create", # Action - create
    "t": "root", # Type - root
    "c": 8, # Category - cost
    "m": 1, # Version - 1
}
```

Example encoded message:
`85a176c40a000000000000000f4240a161a66372656174 65a174a4726f6f74a16308a16d01`

Ban message

This admin message is responsible for banning token balances at the specified address. It requires additional output to the ban address with a small marker amount, which would help Token Layer clients identify address to be banned. No additional fields are required.

Example raw data:

```
{
    "c": 4, # Category - ban
    "m": 1, # Version - 1
}
```

Example encoded message:
`82a16304a16d01`

Unban message

This admin message is responsible for unbanning token balances at the specified address. It requires additional output to the unban address with a small marker amount, which would help Token Layer clients identify address to be unbanned. No additional fields are required.

Example raw data:

```
{
    "c": 5, # Category - unban
    "m": 1, # Version - 1
}
```

Example encoded message:
`82a16305a16d01`

Fee address message

This admin message is responsible for updating fee address which would receive MBC payments for token creation and issuance. It requires additional output to the new fee address with a small marker amount, which would help Token Layer clients identify new fee address. No additional fields are required.

Example raw data:

```
{
    "c": 7, # Category - fee address
    "m": 1, # Version - 1
}
```

Example encoded message:
`82a16307a16d01`

References

[1] https://en.bitcoin.it/wiki/ASIC
[2] https://www.groestl.info
[3] https://bitcointalk.org/index.php?topic=5057818.0
[4] https://www.slideshare.net/bschn2/the-rainforest-algorithm
[5] https://www.investopedia.com/terms/u/utxo.asp
[6] https://en.bitcoin.it/wiki/Genesis_block
[7] https://en.bitcoin.it/wiki/Controlled_supply
[8] https://github.com/bitcoin/bitcoin/compare/v0.17.1...luke-jr:example_300k-0.17
[9] https://bitcoin.org/bitcoin.pdf
[10] https://www.openwall.com/yespower/
[11] https://github.com/MicroBitcoinOrg/Power2B
[12] https://blake2.net/
[13] https://github.com/zawy12/difficulty-algorithms/issues/3

[14] https://en.bitcoin.it/wiki/Satoshi_(unit)
[15] https://bitcoincore.org/
[16] https://github.com/btcsuite/btcd
[17] https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki
[18] https://petertodd.org/2016/forced-soft-forks
[19] https://lightning.network/
[20] https://www.omnilayer.org/
[21] https://en.bitcoin.it/wiki/Script
[22] https://msgpack.org/index.html
[23] https://github.com/msgpack/msgpack/blob/master/spec.md#type-system

Links

Official Website: https://microbitcoin.org
GitHub: https://github.com/MicroBitcoinOrg/
Explorer: https://microbitcoinorg.github.io/explorer/#/
Web Wallet: https://microbitcoinorg.github.io/wallet/#/
API: https://api.mbc.wiki/
Discord: https://discord.gg/8zg2nTV
Telegram: https://t.me/microbitcoinorg
Twitter: https://twitter.com/MicroBitcoinOrg
Forum: https://mbc.wiki
BitcoinTalk: https://bitcointalk.org/index.php?topic=3982489.msg37769108
Reddit: https://www.reddit.com/r/MicroBitcoinOrg/
Token Layer: https://github.com/MicroBitcoinOrg/Tokens