Warthog - An experimental fresh rewrite of a PoW cryptocurrency

Note: This white paper is a work in progress. ## Introduction and Project goals Warthog is a classical Proof-of-Work (PoW) based cryptocurrency which is meant to be a fun and experimental side project of its developers Pumbaa, Timon and Rafiki who work in blockchain industry.

We are working on Warthog in our free time and there is always the risk that we leave due to time constraints or personal reasons. In fact our colleague Timon has already left the team. Therefore we are trying to build up a strong community backing the project.

Mainly one can describe this project as an experiment to try out new things and learn how blockchain technology works in detail. But at the same time we want to be as transparent and fair as possible and avoid as much as possible fishy and questionable practice currently seen in most other new projects.

The code is freshly written in C++20 and is not a cheap fork of any other project. Therefore there is always the risk of serious or unfixable bugs. But at the same time there is real effort put into this project which sets it apart from most competitors.

The community is welcome to take actively part in the evolution of Warthog, the logo and the explorer are made by volunteers and also possible choices for a mining algorithm are proposed. The connection with the community shall be preserved and extended in the future. If you want to join, please do so!

There is no specific purpose or use case of Warthog. However we want to revive the days when crypto was a fun and an interesting experimental thing. One of our experiments with this project is to use SQLite to store blocks and state. Another is the completely novel idea of syncing nodes via *chain descriptors* instead of asking other nodes for blocks by their hashes.

One design principle of Warthog is fast sync speed and a low impact on system resources which is achieved by the use of fast hashing algorithms, appropriate data structures and our custom-built sync algorithm.

For now and in the near future the primary plan is to make the node implementation more robust and improve infrastructure like explorer and API for better interoperability.

Technical Details

Retarget Logic

Similarly to Bitcoin, the warthog blockchain will scale its difficulty periodically to adjust for changing hashrate. Changes in difficulty is partitioned into two phases: - In the initial phase the difficulty is adjusted every 720 blocks which

corresponds to approximately 4 hours. - In the second phase the difficulty is adjusted every 8640 blocks which corresponds to 2 days.

The reason for this two-phase approach is the high variability of hashrate in early stages of a project's life which initially requires a more frequent difficulty adjustment. On the other hand too short intervals also have disadvantages such as the tendency to oscillate and a possibly higher impact of faked timestamps. Therefore the second phase stretches the difficulty adjustment interval after the initial phase.

While in Bitcoin the difficulty change is capped by factor 4, we have implemented a factor 2 cap because our difficulty adjustment is more frequent than 2 weeks.

Emission Scheme

Warthog was started without any premined or reserved amount of coins on June 29, 2023. The project implements a classical halving-based emission scheme with halvings occurring every 3153600 blocks (every 2 years). The emission for the next 4 years is summarized in the following table:

Date	Lifetime	% of total supply in circulation
June 29 2023	0 years	0%
~June 29 2024	1 years	25%
~June 29 2025	2 years	50%
~June 29 2026	3 years	62.5%
~June 29 2027	4 years	75%

There is no tail emission which means there is a hard cap of the amount in circulation. The hard cap is 18921599.68464 WART (around 19 million coins).

Before halving occurs every block yields 3 WART as miner reward. Since the block time is 20 seconds, every day approximately $60/20 * 60 * 24 = 4320$ blocks and 12960 WART are mined before halving.

Coin Precision

The reference implementation uses the C++ data type `uint64_t` for storing amounts of WART. This is a 64 bit unsigned integer. To represent fractions of a coin these values are interpreted in fixed point arithmetic with 8 digits precision. This means that 1 WART is internally represented as `uint64_t` number with value 100000000. The smallest representable step is 0.00000001 WART and represented as `uint64_t` number with value 1.

For easier integration all API endpoints return both, the WART amount as a string (like `"amount": "12.0"`), and the internal integer representation indicated with label "`E8`" (like `"amountE8": 1200000000`).

One-of-a-kind chain descriptor based sync

This project is an experiment where the developers try out new things and push the boundary of what is possible in blockchain technology. We invented a completely unique and new way of syncing nodes which is not presently not known to the industry.

Traditionally during synchronization new nodes request block bodies identified by block hashes. The replying node has to look up the block body based on the hash and then sends it back.

In contrast we have invented a node communication protocol which works without block hashes for block body lookup. In our setup nodes keep track on fork heights with other nodes. A *chain descriptor* is used to identify a specific chain on the peer. When a node appends to its chain, the chain descriptor remains unchanged, however the current chain descriptor is increased when the consensus chain switches to a longer fork. Block bodies for previous chains are also kept for some time in case a peer requests them.

When syncing nodes request block bodies identified by a chain descriptor and a block range. This way we avoid overhead in communication and lookup.

SQLite backed block store

SQLite is a battle-proven and well-established embedded SQL database engine. Warthog nodes use SQLite as their main storage engine for both, blocks and state. Nodes also index transactions and can provide basic blockchain explorer functionality directly via API thanks to SQLite.

SQLite databases are also portable across 32-bit and 64-bit machines and between big-endian and little-endian architectures such that chain snapshots can easily shared. Furthermore SQLite supports transactions which are essential for data integrity even in case of a power outage or node crash.

The default SQLite database file name used for the chain is `chain.db3` and can be configured via the `--chain-db` command line option

Account based architecture

Warthog implements an account based architecture. This is similar to Ethereum and different from Bitcoin's UTXO model. Every account along with its balance is stored in the `State` table of the chain database. For efficiency reasons accounts are referred by their id: Every account is assigned a unique auto-incremented id value on first use. This makes blocks more space-efficient since a block id only requires 8 bytes of storage whereas an address would require 20 bytes.

Anatomy of a block

The binary content of a block is a concatenation of the following sections in their specified order:

1. Mining section
2. New address section
3. Reward section
4. Transfer section

Below we describe the above sections. All numbers and id values are in network byte order.

Mining section This section allows miners to put 4 bytes of arbitrary data to affect the merkle hash.

byte range	content
1-4	arbitrary data

New address section This section lists new addresses that receive payments in this block and therefore need to be added to the `state` table. This way they will be assigned a new id value which is referenced in the other sections to specify a particular account.

byte range	content
1-4	number `n` of new addresses
5-(4+n*20)	`n` addressess of 20 bytes each

Miners are responsible to ensure that the addresses appearing in the new address section are **not already present in the state table** and **are actually referenced in this block**. Otherwise the block is considered invalid.

Reward section Mining reward is distributed to at least one reward address.

byte range	content
1-2	number `r` of reward entry
3-(4+r*16)	`r` reward entries

Every reward entry consists of 16 bytes:

byte range	content
1-8	accountId
9-16	amount

The sum of the amounts received by the addresses listed in the mining reward section must not exceed the total mining reward (block reward + transaction fees), otherwise the block is considered invalid.

The total size of the mining section is 2+ `r`*16 bytes.

Transfer section The transfer section contains the transfers made in this block. Its binary outline is as follows:

byte range	content
1-4	number `t`of transfer entries
5-(4+t*99)	`t` transfer entries

Every transfer entry has the following structure:

byte range	content
1-8	fromAccountId
9-16	pinNonce
17-18	fee
19-26	toAccountId
27-34	amount
35-99	recoverable signature (65 bytes)

Each payment entry has length 99 bytes. Compare this to the average transaction size of around 200 bytes per Bitcoin transfer.

Fee specification

For efficiency and compactness transaction fees are encoded as 2-byte floating-point numbers (16 bits), where the first 6 bits encode the exponent and the remaining 10 bits encode a 11 bit mantissa starting with an implicit 1. This means that fee values cannot be 0 and are of lower precision than regular amount values which use 4 bytes. A fee of value of `0` specified on transaction generation will automatically transform into the minimal fee value of `0.0000001 WART`.

Project Links

- Bitcointalk
- GitHub
- Discord
- Website
- Explorer