The ΣRGO Yellowpaper

Alexander Chepurnoy Dmitry Meshkov

November 10, 2017

Contents

Introduction

Multiple Modes

Ergo (since the very first testing network Testnet0) is supporting multiple security models. In addition to fullnode mode, which is similar to Bitcoin fullnode, Ergo reference implementation will support Light-SPV, Light-Fullnode, Pruned-Fullnode modes.

Full-Node Mode

Like in Bitcoin, a full node is storing all the full blocks since genesis block. Full node checks proofs of work, linking structure correctness (parent block id, interlink elements), and all the transactions in all the blocks. A fullnode is storing all the full blocks forever. It is also holding full UTXO set to be able to validate an arbitrary transaction.

The only optimization a fullnode is doing is that is is skipping downloading and checking AD-transformation block part (see below in the "Light-Fullnode" section).

Pruned Full-Node Mode

This mode is similar to fast-sync in Geth or Grothendieck, warp-mode in Parity (all the three are Ethereum protocol clients), but makes more aggressive optimizations. In particular, a pruned-fullnode is not downloading and storing full blocks not residing in a target blockchain suffix, and also removing full blocks going out of the suffix.

In detail, a pruned client is downloading all the headers, then, by using them, it checks proofs-of-work and linking structure(or parent id only?). Then it downloads a UTXO snapshot for some height from its peers. Finally, full blocks after the snapshot are to be downloaded and applied to get a current UTXO set.

A pruned fullnode is also skipping AD-transformation block part, like a fullnode.

Additional setting: "suffix" - how much full blocks to store(w. some minimum set?)

Light Full-Node Mode

This mode is based on an idea to use a 2-party authenticated dynamic dictionary built on top of UTXO set. A light-fullnode holds only a root digest of a dictionary. It check all the full blocks, or some suffix of the full blockchain, depending on setting, thus starting from a trusted pre-genesis digest or some digest in the blockchain. A light-fullnode is using AD-transformations (authenticated dictionary transformations) block section containing batch-proof for UTXO transformations to get a new digest from an old one. It also checks all the transactions, but doesn't store anything but a single digest for that. Details can be found in the paper https://eprint.iacr.org/2016/994.

Additional settings : "depth" - from which block in the past to check transactions (if 0, then go from genesis)

"additional-checks" - light-fullnode trusts previous digest and checks current digest validity by using the previous one as well as AD-transformations.

"additional-depth" - depth to start additional checks from.

Light-SPV Mode

This mode is not about checking any full blocks. Like in Bitcoin, an SPV node is downloading block headers only, and so checks only proofs of work and links. Unlike Bitcoin's SPV, the Light-SPV is downloading and checking not all the headers but a sublinear(in blockchain length) number of them(in benchmarks, this is about just tens of kilobytes instead of tens or hundreds of megabytes for Bitcoin/Ethereum).

Light-SPV mode is intended to be useful for mobile phones and low-end hardware.

Mode-Related Settings

Ergo has the following settings determines a mode:

- ADState: Boolean - keeps state roothash only
- VerifyTransactions: Boolean - download block transactions and verify them (requires BlocksToKeep == 0 if disabled)
- PoPoWBootstrap: Boolean - download PoPoW proof only
- BlocksToKeep: Int - number of last blocks to keep with transactions, for all other blocks it keep header only. Keep all blocks from genesis if negative
- MinimalSuffix: Int - minimal suffix size for PoPoW proof (may be pre-defined constant)

‘if(VerifyTransactions == false) require(BlocksToKeep == 0)‘
Mode from **"multimode.md"** can be determined as follows:

Ergo Block Structure

ErgoMinimalHeader is a minimal data amount, required to calculate blockId:

payloadRootHash: Array[Byte] - root hash (or simple hash of all payload data) of block payload nonce: Int - field to iterate and generate valid PoW

ErgoHeader is a header to keep in History and transfer:

Field	Size	Decription
version	1	block version, to be increased on every soft- and hardfork
parentId	32	id of parent block
interlinksRoot	32	root hash of interlinks structure
ADProofsRoot	32	hash of ADProofs for transactions in a block
stateRoot	32	root hash (for an AVL+ tree) of a state after block application
transactionsRoot	32	root hash (for a Merkle tree) of transactions in a block
timestamp	8	block timestamp(in milliseconds since beginning of Unix Epoch)
nonce	8	Proof-of-Work nonce

Some of this fields may be calculated by node by itself: - parentId: $if(status == bootstrap \wedge PoPoWBootstrap == false)$ (kushti: ???) - interlinksRoot: $if(PoPoWBootstrap == false)$ - ADProofsRoot: $if(status == regular \wedge ADState == false \wedge BlocksToKeep > 0)$ - stateRoot: $if(status == regular \wedge ADState == false \wedge BlocksToKeep > 0)$

Ergo Modifiers Processing

This section describes processing algorithm for Ergo modifiers in all security modes.

Unlike most of blockchain systems, Ergo have the following types of **modifiers**:
In-memory:

- *Transaction* - in-memory modifier
- *TransactionIdsForHeader* - ids of transactions of a block
- *UTXOSnapshotManifest* - ids of UTXO chunks and

Persistent:

- *BlockTransactions* - Sequence of transactions, corresponding to 1 block.
- *ADProofs* - proof of transaction correctness relative to corresponding UTXO
- *Header* , that contains data required to verify PoW, link to previous block, state root hash and root hash to it's payload (BlockTransactions, ADProofs, Interlinks ...)
- *UTXOSnapshotChunk* - part of UTXO
- *PoPoWProof*

Modifiers processing

Bootstrap

2.Download initial State to start process transactions:
3.Update State to best headers height
4. GOTO regular mode

regular

Two infinite loops in different threads with the following functions inside:
1. $updateHeadersChainToBestInNetwork()$
2.Download and update full blocks when needed

Bibliography