Blockchain implementation go

David Devoogdt

August 2019

1 Introduction

This is a homemade implementation of a blockchain cryptocoin from scratch in go. No technical specification was followed. Some parts are briefly discussed here. The aim of this hobby project is twofold: to learn about concurrency patterns and to learn about proof of work cryptocurrencies.

2 Overview (POW) cryptocurrency

The blocks are the fundamental entity in the blockchain. Each block is cryptographically linked to the previous block. This induces an ordering of the blocks in the chain. Each block is accompanied by transaction data. The transaction need to reference an previously unspent transaction as input and can make outputs maximally adding up to the referenced input amount. Transaction request are broadcast over a network, where miners pick them up and include them in the next block. To link 2 blocks together, some computer has to brute force an input number of the block to create a block with hash with enough leading zeros. This is called mining. As reward for this, the miner receives a reward in the form of a transaction output of a fixed amount with no corresponding input. As only the longest chain in the transaction tree is considered valid, it is in every miner's interest to switch to this new block to mine upon. Once a block is in the chain, it is nearly impossible for an attacker to create another, longer chain because the chances of mining 2 consecutive blocks are very slim.

3 Overview implementation

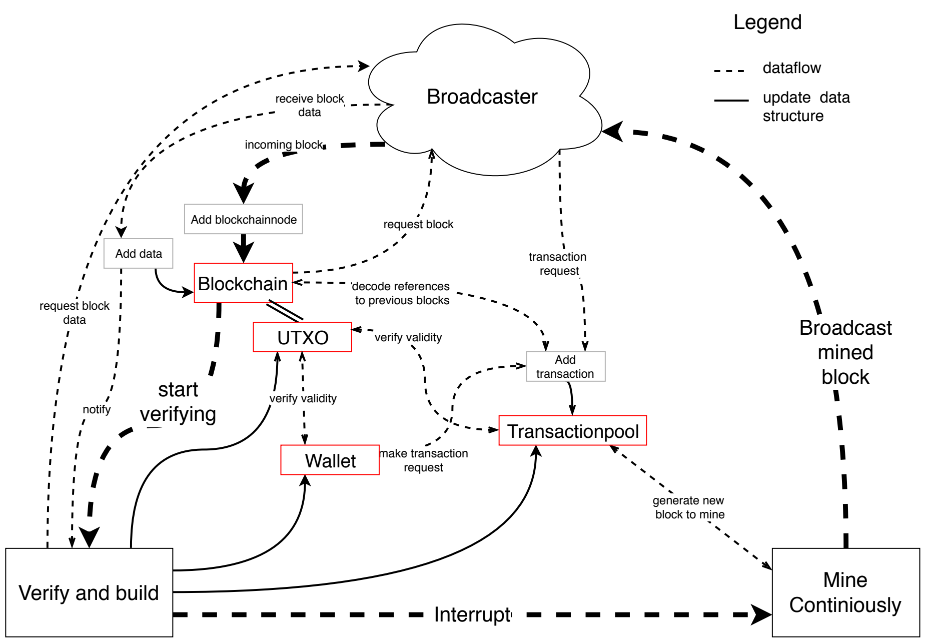

Figure 1: Dataflow through the program.

The program and dataflow are shown in figure 2. A block, mined by someone else on the network, is received by broadcaster. This block gets added to the internal representation of the blockchain. It is checked whether the previous block in the chain is already known. If not the missing block is requested to the network. If a block is added with a block number which exceeds the current head of the chain, verify and build is invoked. This function checks whether the whole chain leading up to this block is valid, possibly requesting the data corresponding to the block from the network. If this new block is valid, the miner is instructed to stop and fetch a new block from the transaction pool. The cycle start again when someone mines a new block.

Most of these sections run concurrently. The blockchain keeps track of all the incoming blocks, thus also the side chains. When a block with an unknown predecessor arrives, a separate blockchain is made until it can be joined to the main chain. Verify and build checks the chain from bottom up. At each step the new block data is checked against the unspent transaction output (utxo) manager. When all the transaction inputs of the new block reference an unspent transaction output, and all the signatures are correct, the utxo manager gets updated to the next block. A copy of the utxo is made each 5 blocks. This is necessary because a side chain which becomes the main chain does not need to go back all the way to the genesis block to verify the new block.

Each time the utxo manager gets updated, the wallet manager and the transaction pool are informed of this new block. The wallet keeps track of the transaction of the miner itself. The transactionpool gets updated because an incoming transaction request possibly referenced a block which was not yet in the main chain. If a new head for the main chain is made, the miner is interrupted. A new block to mine is generated in order to keep mining on the head.

Some relations are ommited in this diagram, such as the procedures to add a mined block internally to save time.

4 Selected parts of the program

4.1 Broadcaster

Everything sent over the network is encoded and later decoded into a []byte. There are several levels of encapsulation of the data. First the data gets serialised to a byte array by a function specific to the type of data. This gets encapsulated together by the type number, the lenght of the data and other information such as the inteded receiver and the sender. This gets in its own turn gets encapsulated by network type: send, request, confirmationrequest and confirmation accept.

Before the actual data is sent, the receiver is requested whether the data will be accepted. The sender keeps an separate routine open to receive the acceptation and then continues sending the data. This is useful because request to the network is received by many different receivers. Once the requester receives the data, it tells the others not to send the data. The data gets filtered based on the hash of the incoming data.

Each miner keeps a separate thread listening to the incoming network messages. All the incoming data gets processed in a new thread. Also the sending and requesting of data is done asynchronously.

4.2 Transactionblockgroups

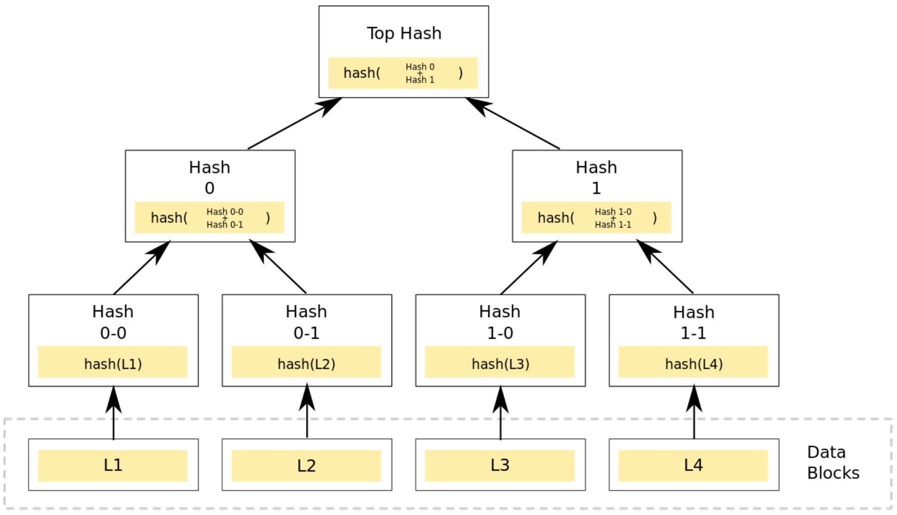

Figure 2: Illustration of a hash tree, taken from [1]

Each block is accompanied by a transactionblockgroup. This is a number of transactions grouped together in a Merkle tree. A Merkle tree is a binary hash tree. The advantage of using such a tree is that each transaction can be sent over a network, together with a number of hashes. In the tree, the next hash in the tree is the hash of the previous 2. Only the root hash, saved in a block, needs to be known to the receiver to verify whether the data sent over the network is really the data in the blockchain. The data and the hash next to it are sent over the network. The receiver hashes these 2 things together and gets a new hash. This gets again hashed together with the sent hash next to it. This pattern repeats until one arives at the root hash. This repository has an own implementation of a merkle root under pkg/merkleroot

5 Interacting with the underlying blockchain

A shell interface was added to experiment with the underlying blockchain. The commands are self explanatory. One could for instance add a new miner, make a transaction, let every miner print its internal representation of the blockchain,...

References

[1] Wikipedia contributors, “Merkle tree — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 2-September-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Merkle_tree&oldid=911089346