Design Document

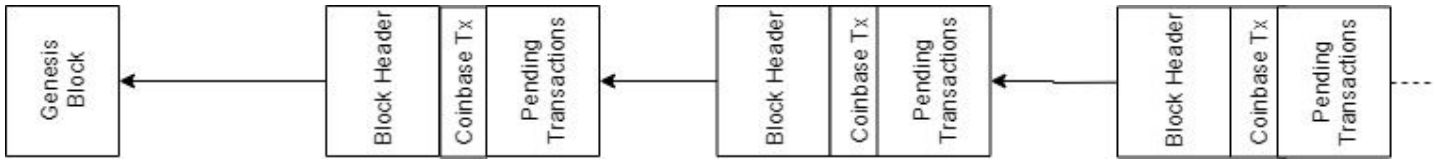
CSE 535: Asynchronous Systems

November 19th, 2018

Project Topic	**Implementation of Proof of Work Consensus in Dist Algo**
Technology	DistAlgo and Python
Team Members	Arun Swaminathan (SBU ID:112044697)
	Hardik Singh Negi (SBU ID:111886786)
	Shubham Jindal (SBU ID:112129688)

Introduction

This design document explains the architecture and class module structure of our project. It starts by explaining the design of the blockchain and its constituent blocks. Then it describes the implementation of the transaction ledger in form of Merkle Tree and finally concludes with the class layout of miners and nodes involved in the blockchain.



A simple layout of Blockchain(without forks)

Implementation Overview

In this section, we will layout the basic implementation details of our implementation system and it's components. These are as follows:

- **Block:** This is a constituent element of the blockchain we are implementing. Its implementation has properties like index, header, data etc. It is implemented in class Block(object) of our system. Each block within the blockchain is identified by a double hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block also references a previous block, known as the *parent* block, through the "previous block hash" field in the block header.

- **Blockchain:** The blockchain data structure is an ordered, back-linked list of blocks of transactions. The block object forms a constituent of a blockchain. Blockchain maintains the ledger of transaction blocks as a list and the first block is called Genesys Block. We implemented blockchain using class

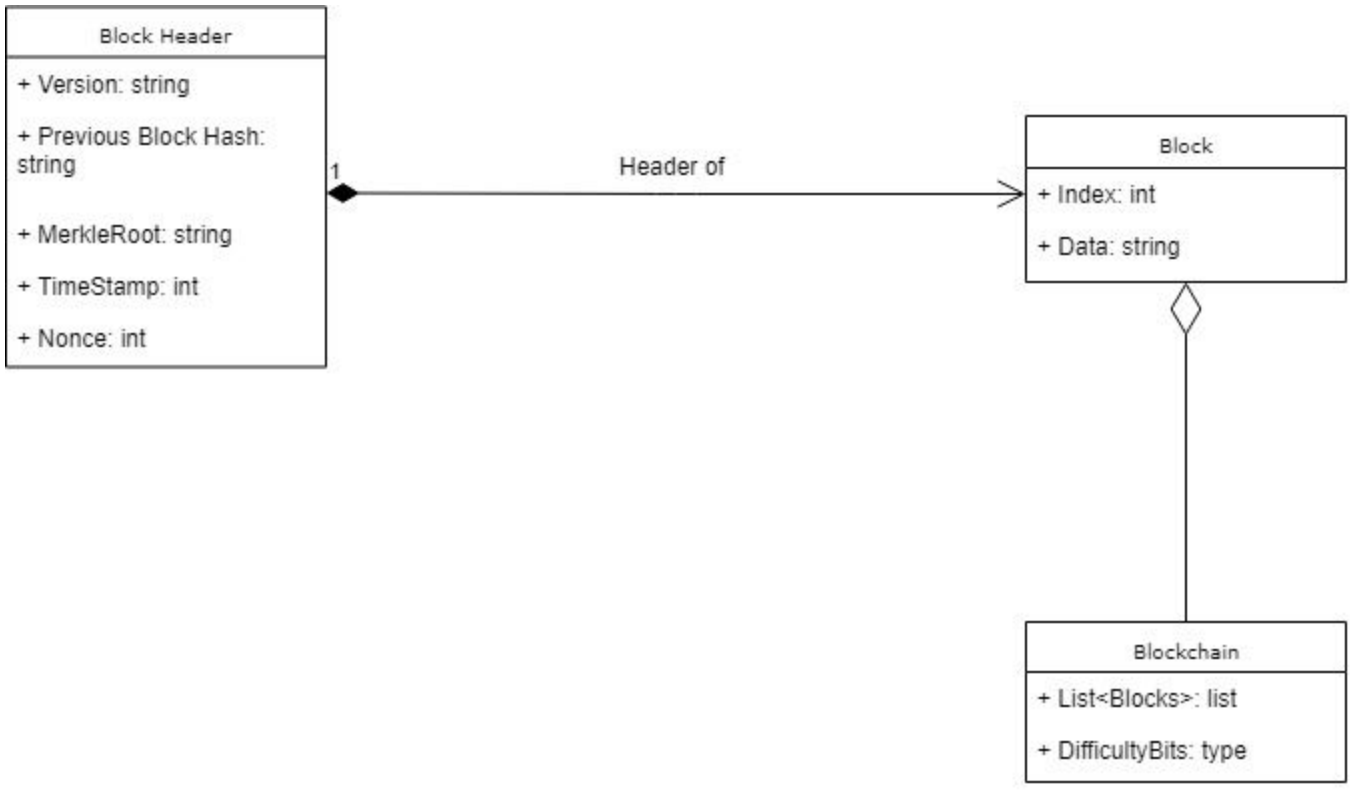
Blockchain(object), this class has methods to manage the blockchain, call block miners and most importantly apply proof of work consensus algorithm.

- **Transactions:** This is the implementation of the transaction ledger which contains transaction history of the blockchain. This is implemented in using class Transaction(object) in Merkle Tree form for ease of access.

- **Nodes and Miners:** Nodes are the sites participating in the blockchain and miners perform the transaction ledger validation. These are implemented in class Miner(process) and class Node(process).

In the upcoming section, we will describe the implementation structure and design of these classes along with a simple architectural layout.

Implementation Details



Architectural Diagram of Blockchain and Block Class

Block

This section describes the API layout of Block Class with important properties and parameters.

class Block(object)

Parameters	Use
index	Index of the block in the blockchain
data	Data related to the transactions
header	Block header has parameters like timestamp, previous_hash, nonce, merkle_root

Properties/Methods	Use
blockhash(self)	Other Block instance
__eq__(self, other_block)	Check if the other_block is equal to the given block or not

Following is the list of parameters in the **Block Header**

Parameters	Use
previous_hash	Hash of previous block
timestamp	Timestamp of block generation
merkel_root	Contains the root of the merkel tree based transaction ledger
nonce	Nonce generated from Proof Of Work

Blockchain

This section describes the API layout of Blockchain Class with important properties and parameters.

class Blockchain(object)

Parameters	Use
_chain	List of blocks initialized with Genesys block
difficulty_bits	Difficulty level for the nonce calculation using Proof Of Work Consensus algorithm

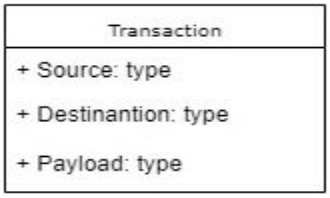
Properties/Methods	Use
chain(self)	Used to display the blockchain ledger in form of a dictionary
get_latest_block(self)	Return the latest block in the blockchain list
proof_of_work(self, candidate_block)	Implementation of proof of work consensus algorithm in the blockchain, which generates a nonce for the new block

Proof Of Work

We have implemented Proof Of Work in Blockchain class itself, however, in this section we will give some more details to proof of work implementation.

Proof of Work through mining secures the blockchain system and enables the emergence of network-wide consensus without a central authority. We have used SHA 256 as a hashing algorithm to generate a nonce for the new block. The difficulty_bits, control the difficulty of the hashing problem used to generate a nonce for a new block. Difficulty bits help in setting a target and for generating new block we need to loop over the nonce space of 4 billion to find a valid nonce which generates a valid nonce for the new block. Increasing the difficulty by 1 bit causes a doubling in the time it takes to find a solution.

Transactions



This class represents transactions taking place in the blockchain.

class Transaction(object)

Parameters	Use
source	Source of the transaction
destination	Recipient of the transactions
payload	Payload/Object involved in transactions

In our system, we have implemented the transaction ledger in form of a Merkle Tree.

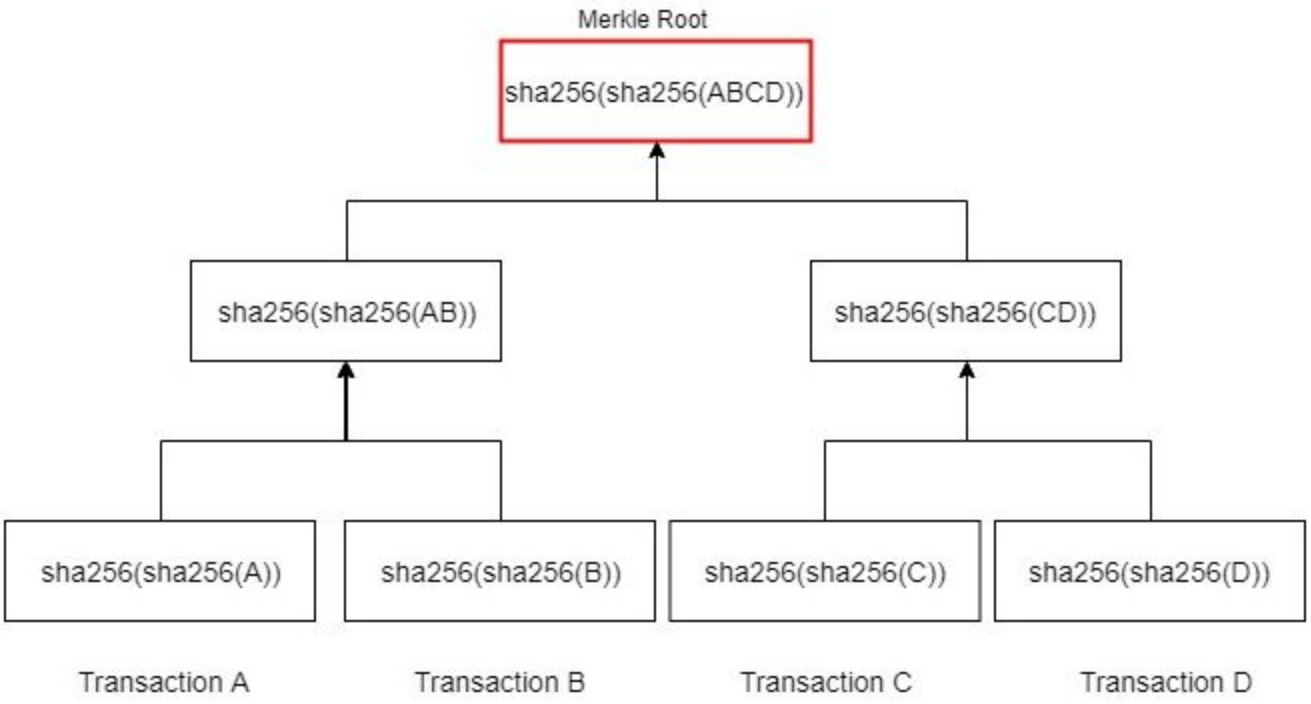
Merkle Tree

Each block in the bitcoin blockchain contains a summary of all the transactions in the block using a *merkle tree*, also known as a *binary hash tree*, a data structure used for efficiently summarizing and verifying the integrity of large sets of data.

Every node in the Merkle Tree contains the hash of one transaction. Because the merkle tree is a binary tree, it needs an even number of leaf nodes. If there is an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes, also known as a *balanced tree*.

The transaction hashes are then combined, in pairs by simply adding the hash and double hashing it again, creating each level of the tree, until all the transactions are summarized into one node at the "root" of the tree.

In a **merkle path** used to prove inclusion of a data element, a node can prove that a transaction K is included in the block by producing a merkle path that is only four 32-byte hashes long (128 bytes total). The path consists of the four hashes. With those four hashes provided as an authentication path, any node can prove that HK is included in the merkle root by computing four additional pair-wise hashes.



Merkle Tree for Transactions

Nodes

This class represent essential processes carried out by the node in a system.

Parameters	Use
neighbors	Set of all neighboring nodes
blockchain	Local copy of the blockchain

Properties/Methods	Use
broadcast(msg)	Broadcast a transaction to all neighboring nodes
receive_block(block)	Receive completed blocks from neighbors and add to its blockchain

Miners

Miners are specialized nodes which also do the proof of work and update the block in the blockchain. They have same properties and parameters as above but also contain the below:

Parameters	Use
pending_transactions	List of transactions received by the Miner to be added to the next block
orphan_block_pool	Blocks received by the miner whose parent blocks have not yet been received.

Properties/Methods	Use
validate_transaction(transaction)	Validate the transaction received from the neighbors and add to the pending transactions.
validate_block(block)	Validate the transaction received from the neighbors and add to the longest chain in its blockchain.
receive_transaction(transaction)	Receive transaction from its neighboring nodes.
mine()	Mine the next block to be added to the blockchain
broadcast()	broadcast a mined block to its neighbors

References

1. Mastering Bitcoin, https://github.com/bitcoinbook/bitcoinbook, November 2018