NTUA, “Distributed Systems” Term Project, 2023-2024

Introduction – Purpose

Blockchain is the technology behind many modern decentralized applications and is essentially a distributed ledger that allows its users to securely engage in transactions with each other without the need for a central authority.

The purpose of this project is to create `BlockChat`, a platform for exchanging messages and recording transactions that will be supported by a simple blockchain, providing users with a secure and reliable means of sending coins or recording messages exchanged among participants. For consensus, a Proof-of-Stake algorithm will be implemented.

The system should implement the following functionalities:

1. Each `BlockChat` user (each process) has a `BlockChat` wallet with coins (`BlockChat` Coins - BCC) necessary for conducting transactions, whether sending coins or messages. Each wallet consists of (a) a private key known only to the user-owner, allowing them to send coins/messages to other users (other wallets) using BCC, and (b) the corresponding public key required for another user (and wallet owner) to receive coins/messages. The public key also serves as the user's wallet address.
2. Each wallet owner can make transactions (coins/messages), with the types of transactions being:
    a. **Coin Transfer**: If Alice wants to send 9 BCC to Bob, she creates a transaction containing the amount she wants to send and Bob's wallet address (Bob's public key). Alice signs this transaction using her private key, and she incurs an additional 3% fee.
    b. **Message Sending**: If Alice wants to send a message "Hello Bob" to Bob, she creates a transaction containing (a) the string of the message she wants to send and (b) Bob's wallet address (Bob's public key). Alice signs this transaction using her private key. The charge for each transaction is proportional to the message size: 1 BCC per character. In this case, Alice incurs a charge of 9 BCC.
3. Every transaction created is broadcast to the entire `BlockChat` network.
4. Nodes participating in the block validation process verify the new transaction upon receiving it. In our example, they first check that the transaction originated from Alice using her public key, verify that Alice has enough funds in her wallet to execute the transaction (9 BCC in our case), and if all checks succeed, add the transaction to the current block. For this exercise, assume that all users participate in the validation process.
5. When the current block is full, nodes start the validation process following the Proof-of-Stake algorithm described below. The node that eventually validates the block broadcasts it to the entire network, making it known to all participating nodes.
6. The validator of the block is the one whose wallet is credited with the charges from the transactions included in the block.

Network

We assume that our network has a size of n, meaning n nodes participate in the network with IDs ranging from 0 to n-1. Each node in our network should be able to conduct transactions with all other nodes in the system, knowing their IP address and the port they are listening on. For simplicity, we assume that each node maintains a list of all other nodes currently participating in the `BlockChat` system. Communication can be implemented either with sockets (TCP or UDP) or using REST API, utilizing any library of your choice.

The first node of the network is called the bootstrap node (ID 0), responsible for creating the genesis block, the first block of our blockchain. In the genesis block, we set `previous_hash=1`, `validator=0`, and the list of transactions includes only one transaction, giving the bootstrap node 1000*n BCC coins from wallet address 0. This is the only block that is not verified. Gradually, the remaining n-1 nodes of the network are added.

To introduce a new node into the system, it first communicates with the bootstrap node, whose communication details (IP address/port) are known to all, and sends its wallet's public key. From the bootstrap node, it receives its unique ID (the first node gets ID 1, the second gets ID 2, and so on). When all nodes are introduced, the bootstrap node broadcasts to all the pairs of IP address/port, as well as the public keys of the wallets of all nodes participating in the system. Assume that from this point onward, there are no node introductions or departures from the system. Additionally, each new node receives the blockchain from the bootstrap node as it has been configured up to that moment, validates it, and can then perform transactions.

The bootstrap node, as mentioned earlier, has 1000*n BCC coins from the genesis block. After connecting each new node to the network, the bootstrap node executes a transaction transferring 1000 BCC to that node. Thus, after the introduction of all nodes, each of them has 1000 BCC in their wallet.

`ChatBlock` Backend

In the `ChatBlock` backend, the entire logic of the blockchain is implemented. The fundamental components of the blockchain are as follows:

Block

Each block contains the following information:

- `index`: the incrementing number of the block
- `timestamp`: the timestamp of the block's creation
- `transactions`: the list of transactions contained in the block
- `validator`: the public key of the node that validated the block
- `current_hash`: the hash of the block
- `previous_hash`: the hash of the previous block in the blockchain

It is assumed that each block has a specific capacity for the number of transactions, and this capacity is determined by the constant `capacity`.

Blockchain

A list of blocks that have been verified.

Wallet

A wallet is associated with a pair of public/private keys. The public key serves as the wallet's address, which can be shared with anyone to receive messages. The private key is used to sign transactions, ensuring that only the wallet owner (the holder of the corresponding private key) can spend money from the wallet by sending a message. This is done with the function `sign_transaction()`. Anyone who knows the public key of a wallet can verify that a transaction sent from that wallet was created by its owner using the `verify_transaction()` function.

Transaction

Each transaction contains information about the transfer of coins/messages from one wallet to another. The information it includes is as follows:

- `sender_address`: the public key of the wallet from which the message originates
- `receiver_address`: the public key of the wallet recipient of the message
- `type_of_transaction`: determines the type of transaction (coins or message)
- `amount`: the number of coins to be sent
- `message`: the string of the message being sent
- `nonce`: a counter maintained per sender, incremented by 1 after each transaction
- `transaction_id`: the hash of the transaction
- `signature`: a signature proving that the wallet owner created this transaction

A transaction can be created by the wallet owner, who will be charged either the amount they want to send or the cost of the message. Each transaction is broadcast to all members of the blockchain. Upon receiving any transaction, the `validate_transaction()` function is called to check its correctness. Each transaction in the blockchain, for the scope of the exercise, should belong to one of the two use cases (either sending coins or a message). However, should you wish to implement a transaction that can send both a message and coins, you are free to do so. Additionally, if it is more convenient, you can define two different types of transactions in your code (instead of one, as shown above).

Proof-of-Stake (PoS)

Proof-of-Stake is a consensus mechanism that provides an alternative approach compared to the more well-known Proof-of-Work (PoW). In PoS, the selection of the next block validator depends not on computational power but on the amount of cryptocurrency (staking) held by a node. In practice, each node that wants to participate in the validation process commits an amount from its wallet using the `stake(amount)` function. This function translates into a transaction where the `receiver_address` is 0, and the amount is the one the node wants to commit. Upon receiving such transactions, it should be verified, among other things, that the staking amount is available in the sender's wallet.

The probability of being the next validator is proportional to the amount staked in relation to the total staking amount from all possible validators. If a node has, for example, 5% of the total staking amount, it has a 5% chance of being selected as the validator for the next block. The process is as follows: Once the current block reaches its capacity with transactions, the node calls a pseudo-random number generator that will indicate who the validator of the block will be. To make the result of the generator deterministic for any node, all nodes must use the same seed, which is determined by the hash of the previous block in the chain. The node recognizing itself as the validator validates the block and broadcasts it to all.

To find the validator from the pseudo-random generator based on each node's probabilities of being selected, a “lottery” logic can be followed: Each node puts as many lottery tickets in the bucket as the number of BCC it has committed, giving them consecutive numbers. The generator draws a lottery ticket, i.e., a number from 1 to the total number of committed BCC, determining the next validator.

Data Model:

The account model will be followed to maintain the state data of the blockchain. Thus, the state that each node in the network must maintain is the list of existing accounts and their balances. Additionally, information about the staking of each node must be kept (which should be deducted from the available balance of the account). With each transaction, nodes receiving it must reduce the balance of the sender's account according to the number of coins they want to send or according to the message charging rule (1 BCC for each character). After receiving a validated block, each node must increase the balance of the validator in the following ways:

1. The total fees charged to the senders of transactions related to money transfers and included in the block.
2. The total BCC charged to the senders of all messages included in the block.

To prevent replay attacks, where a malicious node could resend a transaction it received to charge the sender again, each transaction in the account model contains a nonce field. The nonce is a counter maintained by each account in `BlockChat`, incremented by one with each outgoing transaction. This prevents the submission of the same transaction more than once to the network.

Basic Functions

The basic functions of the system include:

- `generate_wallet()`
  Creates a new wallet, i.e., a pair of public/private keys using the RSA cryptographic algorithm.

- `create_transaction()`
  Creates a new transaction containing all the required information.

- `sign_transaction()`
  Signs a transaction with the wallet's private key.

- `broadcast_transaction()`
  Sends a transaction with broadcast to all nodes.

- `verify_signature()`
  Verifies the signature of a transaction upon receiving it.

- `validate_transaction()`
  Validates the correctness of a received transaction. Verification includes (a) signature verification (`verify_signature()`) and (b) checking the balance of the sender's account to ensure it has the required amount to perform the transaction (either the coins specified in the amount field or the coins corresponding to the characters of the message to be sent). The staking amount must also be considered (should not be spent).

- `mint_block()`
  This function is called once `capacity` transactions have been received and verified by a node. It implements the proof of stake by calling the pseudo-random number generator with the appropriate seed. If the calling node is the validator, it fills all the fields of the block with the appropriate information.

- `broadcast_block()`
  Once a block is created and validated by the validator, it is broadcast to all other nodes.

- `validate_block()`
  Called by nodes (except for the genesis block) upon receiving a new block. Verifies that (a) the validator is indeed the correct one (as indicated by the pseudo-random number generator) and (b) the `previous_hash` field is indeed equal to the hash of the previous block.

- `validate_chain()`
  Called by incoming nodes, which verify the correctness of the blockchain received from the bootstrap node. In reality, it calls `validate_block()` for all blocks except the genesis block.

- `stake(amount)`
  Called by nodes to determine the amount they commit as a stake for the proof-of-stake process. This amount determines the probability of the node being selected as a validator. Each validator should be able to update the committed amount.

`BlockChat` **Client**

You need to implement a client (a simple CLI is sufficient, but if you want to be fancier, it will be appreciated) that allows the user to execute the following commands:

- `t <recipient_address> <amount>`
  New transaction: Send the specified amount in BCC coins from the sender's wallet to the recipient's wallet specified by `recipient_address`. Calls the `create_transaction()` function of the backend that implements the above functionality.

- `m <recipient_address> <message>`
  New message: Send the specified message to the recipient's wallet, specified by `recipient_address`, appropriately charging the sender's wallet. Calls the `create_transaction()` function of the backend that implements the above functionality.

- `stake <amount>`
  Set the node stake: Commit the specified amount for staking of the current node. Calls the `stake(amount)` function as defined above.

- `view`
  View the last block: Print the transactions contained in the last verified block of the BlockChat blockchain, along with the ID of the validator of that block. Calls the 'view_block()' function of the backend that implements the above functionality.

- `balance`
  Show balance: Print the balance of the wallet.

- `help`
  Explanation of the above commands.

Experiments

You will develop `BlockChat` in any programming language you prefer and deploy it on the `~okeanos` infrastructure (detailed instructions will be provided). For the report, you will perform the following experiments:

1. **System Performance**
   Set up `BlockChat` with 5 clients. Each client will read the `transX.txt` file, where X is the node ID of the node. Each file contains messages to other nodes in the format:
   `<recipient_node_id> <message string>`
   Each node will create and send a transaction per line to the network. All clients will send their transactions concurrently. With a fixed staking of 10 BCC for each node and capacities of 5, 10, and 20, record the following:
   - **Throughput**: the number of transactions served per unit of time.
   - **Block time**: the average time it takes to add a new block to the blockchain.

2. **System Scalability**
   Repeat the experiment for 10 clients and compare the results with the previous ones. Plot the results. Present metrics from the previous experiment (y-axis) in relation to the number of clients (x-axis).

3. **Fairness**
   Repeat the experiment with 5 nodes and a capacity of 5. However, this time, one of all nodes has a staking of 100 BCC. What do you observe regarding the fairness of the system?

Work in groups of 2-3 individuals. The deliverables for the assignment include the source code (tarball with all relevant files) and a digital document (`pdf`, `docx`, or `odt`) presenting the design of your system and the results of the experiments. Both the code and the report should be submitted online on the course website. A demonstration of the assignment will be scheduled with the instructors after the deadline for submission.