Due: 13 May 2022 (Friday Week 11) by 23:59

COMP3221 Assignment 2: Blockchain

The goal of this project is to implement a Peer-to-Peer (P2P) Blockchain system in Python.

1 Introduction

Blockchain is a distributed ledger technology (DLT) working based on a peer-to-peer (P2P) topology. The blockchain implements a distributed ledger as a chain of blocks, each block stores a series of transactions just like the page of a ledger. The goal of blockchain is to maintain integrity, confidentiality and authentication of digital assets. Recently, blockchain has been applied into a wide range of application only in digital currency but also for many other sectors such as storing business contracts, recording medical patients history, etc.

In this assignment, you are going to develop a blockchain system by yourself to understand more about this technology from a developer's perspective. Your blockchain can be seen as a distributed system built upon a peer-to-peer (P2P) network containing almost basic components and mechanisms of a real-world blockchain application.

1.1 Learning Objectives

- Enhance the understanding of essential blockchain terms, characteristics and concepts
- Understand how a P2P model works and how to design and implement a P2P system.
- Develop programming skill in the fields of Blockchain and P2P.

1.2 Submission Details

The final version of your assignment should be submitted electronically via CANVAS by 23:59 on the Friday of Week 11. **Students will work individually or in groups of two members**.

2 Assignment Specifications

Task 1: Defining Transaction, Block and Chain of Blocks

In this task, you will define Blockchain's core data structures, which are transactions, blocks and the chain of blocks.

Transaction

Each transaction will be defined as an object containing the sender of the message and the content of the message that is being transferred. A transaction is made by a client with format "tx|[sender]|[content]". You should perform some checks on "[sender]" and "[content]" of a transaction before considering a message as a valid transaction. Here are the rules for a valid transaction:

- The message sender and receiver must present and should match a unikey-like form (regex: [a-zA-Z]{4}[0-9]{4})
- The message content cannot have more than 70 English characters or contain a '|' character (| is used as delimiter)

If a transaction violates those rules, it should be considered as an invalid transaction. While a valid transaction is added to the pool of a blockchain node to wait for being processed, an invalid transaction is going to be rejected. For implementation details, you can reference Week 7 tutorial.

You are required to define a **`Transaction`** class that is used for creating and validating transactions in a **`Transaction.py`** file.

Block

A block is a fundamental element of a blockchain where information is stored and encrypted. Each block is composed by six components defined as follows.

- *Index of Block*: the unique ID of a block. The ID of the first block (genesis block) is 1.
- *Timestamp*: the time at which the block is created.
- *Transactions*: a list of valid transactions that belong to the block.
- *Proof*: The number presents proof of work.
- *Previous Hash*: The hash of the previous block.
- *Current Hash*: The current hash of the block.

In this assignment, we will use SHA-256 hashing algorithm for calculating hash.

An example of block:

```
{
    "index": 2,
    "timestamp": "2022-04-11 15:30:55.778685",
    "transaction": [
    "tx|Comp3221|100BTC",
    "tx|Comp3221|200BTC",
    "tx|Comp3221|300BTC"
    ],
    "proof": 5,
    "previousHash": "482a58cd41e9fcd5cf6000d727e62880ec9f751e2da2b2bf6d5619baa546f895",
    "currentHash": "1bac020dcab752b94ad432a71b6c42541bd9ec6541c3b885bbd963cca2f449e5"
}
```

Chain the blocks

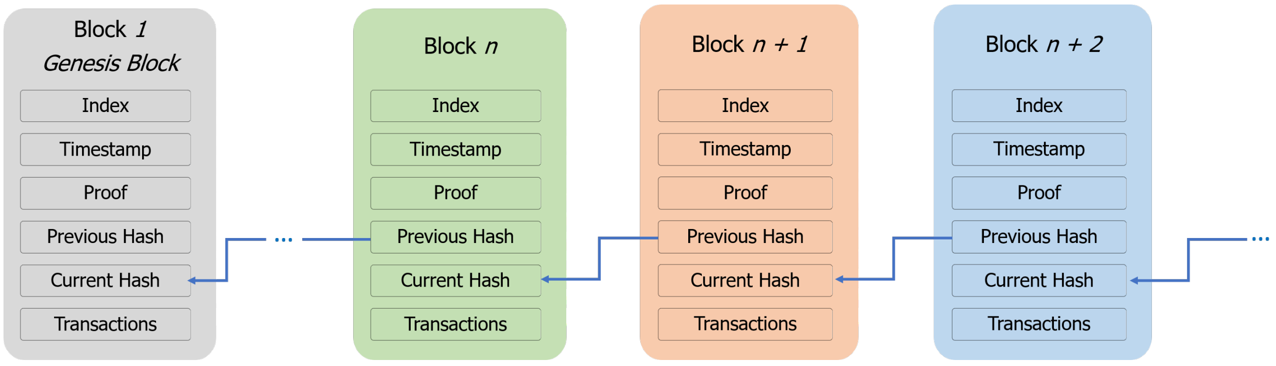

Figure 1: Chain of blocks

The whole concept of a blockchain is based on the fact that the blocks are "chained" to each other an ordered linked-list (Fig. 1). As the blocks are linked together one after the other, they have a parent-child relationship. Each block is the parent of the upcoming block. Each child block contains the hash of the previous block i.e., its parent block. For the first block - the genesis block - it is defined nearly the same with normal block except that there is no previous hash, index is 1 and a predefined content.

You are required to design and implement a **`Blockchain`** class in **`Blockchain.py`** file. Your class should have 3 methods.

- **`calculateHash(data)`**: calculate hash using SHA-256 for a given data.
- **`currentHash(data)`**: calculate hash for data where data = (previousHash + current transactions in pool + proof) ("+" means string concatenation).
- **`PreviousHash`**: the hash value of the previous block.

Task 2: Implementing Peer-to-Peer Communication for Blockchain

A peer-to-peer (P2P) network is a decentralized communication model where all nodes (peers) in the network generally have equal power and can execute the same tasks. The P2P protocol allows peers to communicate with each other without the need to go through any intermediaries.

In this task, you are required to design and implement **a Blockchain P2P network containing at least 3 peers** for exchanging information about transactions and new blocks. Each peer in the network will act as three roles including: Server, Client, and Miner. The detail specifications for each peer's role is described as follows.

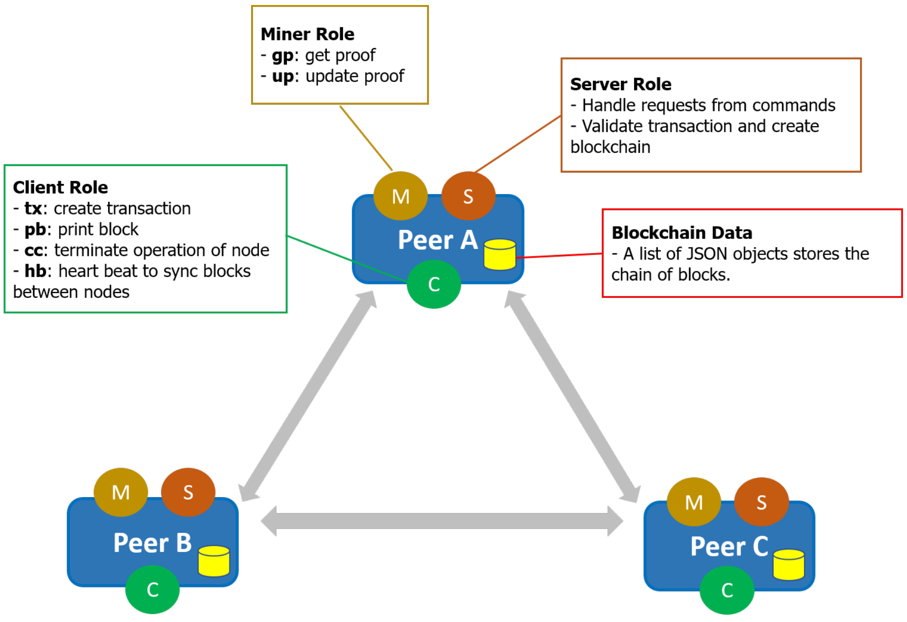
Miner Role
- gp: get proof
- up: update proof
Server Role
- Handle requests from commands
- Validate transaction and create blockchain
Client Role
- tx: create transaction
- pb: print block
- cc: terminate operation of node
- hb: heart beat to sync blocks between nodes
Blockchain Data
- A list of JSON objects stores the chain of blocks.
M
S
Peer A
C
M
S
Peer B
C
M
S
Peer C
C

Figure 2: Peer-to-Peer Blockchain Network

Server Role of Peer

As a server, each peer in the P2P blockchain system will keep listening to a specific port provided by the user. Once a client connects, the server should accept the connection and handle clients' requests. We assume there will be multiple clients accessing your server at the same time, hence, a multi-threaded server is required.

While handling requests, server's threads might modify the same blockchain. Hence, you are also required to improve the blockchain system with proper synchronisation support. You should make sure all your clients can see the same result, and no race condition occurs.

The Blockchain's servers need to handle the following types of requests form clients.

- Transaction request with format: **`"tx|<sender>|<content>"`**. Server validates received transactions. The valid transactions are going to be added to the pool for processing and server sends an "Accepted" message to client, while invalid transactions will be ignored by server and server sends "Rejected" message to client.
- Print block request with format: **`"pb"`**. Server returns the current blockchain with type of a list of json.
- Close connection request. **`"cc"`**. Server closes connection with users and the peer will be terminated.
- Get current proof request: **`"gp"`**. Server sends current proof (or nonce) of the lastest block to client.
- Update proof request: **`"up"`**. Server checks correctness of proof from a sender. If the proof is valid, server will send message "Reward" to the sender. Otherwise, server will sends "No reward" to the sender. In case of valid proof, server will check pool for creating a new block and notify for other peers about the new block.

- HeartBeat message: **`hb`**. Server returns the current blockchain with type of a list of json.

You are required to design and implement a **`BlockchainServer`**class in a **`BlockchainServer.py`** file.

Client Role of Peer

As a client, each peer uses a scanner to listen to inputs from a terminal, and create message requests to the server based on received inputs. There are four commands that a client can receive from the user defined as follows.

- Transaction command: **`tx [sender] [content]`**. A client sends a transaction to blockchain server. This command is received from the terminal and makes a corresponding request to server which resides in the same peer node with client.
- Print Blockchain command: **`pb`**. A client requests to receive blockchain in json format. This command is received from the terminal and makes a corresponding request to server which resides in the same peer node with client.
- Close connection command: **`cc`**. A client terminates server and closes connection with server. This command is received from the terminal and makes a corresponding request to server which resides in the same peer node with client.

Upon receiving user inputs, clients will create requests to the server. For example:

- Transaction request has format: **`tx|[sender]|[content]`**.
- Print Blockchain request has format: **`pb`**.
- Close connection request has format: **`cc`**.

Requests should be string and encoded UTF-8 before sending to server. Your code should show a helper if users enter unknown requests. If the request is **`"cc"`**, the client should terminate. Otherwise, it keeps waiting for new input commands. All responses from server should be correctly printed on the terminal.

You are required to design and implement a **`BlockchainClient`** class in a **`BlockchainClient.py`** file.

Miner Role of Peer

Mining is a process of finding hash for the block that will be considered valid by the system. The underlying algorithm that sets the difficulty and rules for the work miners do is called Proof-of-Work.

Proof of Work Algorithm

Proof of Work (PoW) is one of the basic consensus algorithms used in the mining processes of a blockchain system. It requires network peers to expend effort solving an arbitrary mathematical puzzle to prevent "double spending" or frivolous or malicious activities such as spam and denial-of-service attacks.

In blockchain, if attackers change the previous block, they can re-compute the hashes of all the following blocks quite easily and create a different valid blockchain. To prevent this, we can exploit the asymmetry in efforts of hash functions to make the task of calculating the hash difficult and random.

In particular, instead of accepting any hash for the block, we will add a constraint that our hash should start with "n leading zeroes" where n can be any positive integer. In other words, we hash the contents of a block and adding a nonce (or a proof in our defined block structure), until the returned hash starts with a defined number of zeros. The number of zeroes specified in the constraint decides the "difficulty" of the PoW algorithm. Finding the right proof to compute a hash with the right amount of leading zeros is computationally expensive (more the number of zeroes, harder it is to figure out the proof).

Below is an PoW algorithm which achieves the functionality described above (*** is the exponentiation operator*). In this assignment, we will use hash start with "2 leading zeroes" to find the proof.

Algorithm 1: Proof of Work Algorithm

```
Input  : previousProof
Output: newProof
newProof ← 0;
repeat
    newProof++;
until calculateHash(newProof**2 – previousProof**2)[:2] == '00';
```

As a miner, when new transactions are sent to a peer in the network, the peer will collect all of them to put every 5 transactions into a single block. They will validate the transactions, and then try to solve the PoW problem by finding a proof to generate a hash string that begins with at least two 0. If a peer successfully found it, it will create a new block, append it to the longest chain he has and send this new mined block to all over the network. The new mined block will be confirmed by checking the information of the previous block and itself. Other peers in the network will check the proof and add the new block to their chain.

A miner polls the blockchain server every 1 seconds to check the proof. If the proof is changed (a new block is added to the blockchain), miner will try to find a new proof and submit it to server for creating a new block as soon as possible. To poll and update proof to the server, a miner will use two requests:

- Get current proof with format: `gp`.
- Update the proof for server with format: `up|[newProof]`.

Multiple miners can interact with a server at the same time using multi-threading techniques. There are no commands required for miners as miners automatically request to the servers and find the proof.

You are required to design and implement a **`BlockchainMiner`** class in a **`BlockchainMiner.py`** file.

HeartBeat Message

A peer node automatically generates HeartBeat messages (**`hb`**) and periodically broadcasted to all other connected peers every 5 seconds. Upon receiving the blockchains from connected peers, the peer node will compare with its current blockchain and update if there is any difference.

Blockchain P2P Communication Processes

To deploy and simulate the activities of the P2P blockchain network, each peer should be run and functions as all the defined roles (i.e,server, client and miner) in one peer and provide an interface (e.g., a commandline terminal) for entering the request inputs. The communication processes of peers in the network is described as follows.

1. When a new peer joins the network, the client sends a heartbeat to sync the blockchain data and check all alive peers in the network. Assuming that the total number of peers in the network is fixed and all peers know the port number of other peers in the network.
2. If clients request transactions. New transactions are broadcast to all alive peers in the network.
3. Each peer collects new transactions into a block (max 5 transactions for 1 block).
4. Each peer works on finding a difficult proof-of-work for its block (mining).
5. When a peer finds a proof-of-work, it broadcasts the new block to all peers.
6. Peers accept the new mined block only if all transactions in it are valid and not already spent.
7. Peers express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash and deleting processed transactions.

When a peer in the network starts, it will create at least three different threads to run client, server, and miner. For deploying a peer, you are required to design and implement a **`BlockchainPeer`** class using multithread in a **`BlockchainPeer.py`** file accepting the following command line arguments:

```
python BlockchainPeer.py <Peer-id> <Port-no> <Peer-config-file>
```

For example:

```
python BlockchainPeer.py 1 6000 config_A.txt
```

- **Peer-id**: the ID of a peer in the network. In this assignment, Peer-id is indexed following the English alphabet, for example, A, B, C, etc..
- **Port-no**: the port number of a peer listening to the requests. The port number is integer indexed starting from 6000 and is increased by one for each peer node.
- **Peer-config-file**: **config_A.txt**, for example, is the configuration file for Peer A that has the following details:

```
2
B 6001
C 6002
```

  The first line of this file indicates the number of peers neighbors to Peer A. The next two lines are to determine the connection of Peer A to its neighbors. Those lines start with the neighbor ID, followed by the port number that this neighbor opens for listening requests.

  For example, the second line in the **config_A.txt** above indicates that Peer A has connection to Peer B and Peer B is using port number 6001 for listening the requests from other peers.

3 Program structure

The following python files must be submitted.

- Transaction.py
- Blockchain.py
- BlockchainClient.py
- BlockchainMiner.py
- BlockchainServer.py
- BlockchainPeer.py

You can use the skeleton code provided on Canvas as the base. All files should be located in the folder named "SID1_SID2_A2" with no subfolders. All python files should be correct and do not forget to remove any dummy files that do not count as source files. Please zip the SID1_SID2_A2 folder and submit the resulting archive SID1_SID2_A2.zip to Canvas by the deadline given above. The program should be compiled with Python 3.x version.

3.1 Submission and Report

You are required to submit your source code and a short report to Canvas.

- Code (zip file includes all implementation, readme) `SID1_SID2_A2.zip`.

- Code (including all implementation in one file exported in a txt file for Plagiarism checking)
  `SID1_SID2_A2_Code.txt`.
- Readme: Clearly state how to run your program.
- Report (pdf)
  `SID1_SID2_A2_Report.pdf`.

Please note that you must upload your submission BEFORE the deadline. The CANVAS would continue accepting submissions after the due date. Late submissions would carry penalty per day with maximum of 5 days late submission allowed.

The size of your report MUST be under 2 pages. Your report should briefly document your P2P Blockchain system, techniques and methodology used for implementation and the simulation results of each requirement. It should act a reference for your markers to quickly figure out what you have and haven't completed, how you did it, and it should mention anything you think that is special about your system.

4 Academic Honesty / Plagiarism

By uploading your submission to CANVAS you implicitly agree to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not your work you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

In assessing a piece of submitted work, the School of Computer Science may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

5 Marking

This assignment is worth 20% of your final grade for this unit of study.

- Code: 80%
- Report: 20%

Please refer to the *rubric* in Canvas (Canvas -> Assignment 2 -> Rubric) for detailed marking scheme. The report and the code are to be submitted in Canvas by the due date.

6 Feedback

After Assignment 2 marks come out, please submit your inquiries about marking within the 1st week. All inquiries after that will NOT be responded.