THE UNIVERSITY OF SYDNEY

Due: Tuesday Week 12 by 11:59PM

COMP3221 Assignment 3: P2P Blockchain

The goal of this project is to implement a consistent Peer-to-Peer (P2P) Blockchain application in Java that tolerates the dynamism of the system and the unreliability of the network.

1 Submission Details

The assignment comprises two tasks and each task can be submitted separately. The final version of your assignment should be submitted electronically via PASTA by 11:59PM on the Tuesday of Week 12 (Hard Deadline). The project is an individual project, and each student has to submit his/her own version.

1.1 Program structure

For task 1, seven java files must be submitted. It is recommended to submit by 11:59 AM on the Sunday of Week 10 (Soft Deadline).

- A **`Transaction.java`** file.
- A **`Block.java`** file.
- A **`Blockchain.java`** file.
- A **`BlockchainServer.java`** file.
- A **`BlockchainServerRunnable.java`** file.
- A **`PeriodicCommitRunnable.java`** file.
- A **`ServerInfo.java`** file.

For task 2, seven java files must be submitted. It is recommended to submit by 11:59 AM on the Sunday of Week 11 (Soft Deadline).

- A **`Transaction.java`** file.
- A **`Block.java`** file.
- A **`Blockchain.java`** file.
- A **`BlockchainServer.java`** file.

- A **`BlockchainServerRunnable.java`** file.
- A **`PeriodicCommitRunnable.java`** file.
- A **`ServerInfo.java`** file.

Please use the skeleton code which is provided on Canvas and ed as the base. All classes will be stored in the same default package (no package header in java files), and all files should be located in the same **`src`** folder with no subfolders. All present **`.java`** files should be correct and do not forget to remove any dummy files that do not count as source files (e.g., junit test cases, class files). Please zip the **`src`** folder and submit the resulting archive **`src.zip`** by the deadline given above. The program should compile with Java 8. No optional packages that are not part of the default Java 8 JDK can be used.

1.2 Submission system

PASTA will be used to assess that your program correctly implements the P2P Blockchain protocol. The archive **`src.zip`** should be submitted at `https://pastpd01191.srv.sydney.edu.au/PASTA/home/`. To access this website, you will have to be connected to the network of the University of Sydney (physically on campus or through VPN).

PASTA stands for "Programming Assessment Submission and Testing Application" and is a web-based application that automates the compilation, execution and testing of your program. When you submit your **`src.zip`** archive in PASTA, the system enqueues your submission in the shared queue of all assignments to be tested. It may take more time close to the deadline as many students will try to submit at the same time.

1.3 Academic Honesty / Plagiarism

By uploading your submission to PASTA 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 IT 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 IT for the purpose of future plagiarism checking.

2 Marking

This assignment is worth 15% of your final grade for this unit of study.

The first category of tasks, called *Blockchain: HeartBeat-Based Dynamic Neighbour Management*, assesses the behaviour of the server with respect to some tests. If your code passes all the tests, three marks are given.

- Maximum 1 marks for server sending HeartBeat message with correct sequence number periodically.
- Maximum 1 marks for server receiving HeartBeat message and forwarding ServerInfo to its neighbour correctly.
- Maximum 1 marks for server receiving ServerInfo message and relying ServerInfo to its neighbour correctly.

The second category, called *Blockchain: Catch Up Protocol and Blockchain Consensus*, assesses the behaviour of the server with respect to different tests. If your code could pass all the corresponding tests, ten marks are given.

- Maximum 1 marks for server sending LatestBlock message with correct blockchain length and hash value.
- Maximum 1 marks for server sending correct CatchUp message.
- Maximum 2 marks for server catching up single block properly.
- Maximum 2 marks for server catching up multiple blocks properly.
- Maximum 2 marks for server resolving confilicts properly.
- Maximum 2 marks for server efficiency (manual marking).

  Note: Server efficiency will be assessed manually. You will be required to explain your consensus algorithm and show your implementation to your tutor during demo.

The third category, called *Blockchain: Demo*, assess the ability for you to deploy and demonstrate your P2P blockchain application. Maximum two marks are given.

Please make sure previous tasks are implemented correctly before moving to next task. You may face cascading failures due to incomplete implementation of previous tasks.

3 Functionalities of P2P Blockchain

The goal of this project is to implement a basic Peer-to-Peer(P2P) Blockchain application. In Assignment 2, we have built a client which was able to talk to multiple servers and a server able to accept multiple connections. However, the servers could not communicate with each other, which led servers to have different versions of the blockchain. At this stage, we are going to convert your Blockchain server into a peer node within a P2P network. It will communicate with other peers and reach consensus on their Blockchain. The following tasks provide you with instructions to accomplish this.

Task 1 Communication: HeartBeat-Based Dynamic Neighbour Management

For this task, three functional requirements must be fulfilled:

- HeartBeat sending.
- HeartBeat receiving and ServerInfo sending.
- ServerInfo receiving and ServerInfo relaying.

HeartBeat sending HeartBeat message is defined as **hb|<Port Number>|<Sequence Number>**. The sequence number starts from zero and monotonically increases by one in each period. This message periodically gets broadcast to all neighbour peers every two seconds. If the peer is unable to set up a TCP connection with another peer within two seconds in order to send the heartbeat message, it has to retry in the next period.

HeartBeat receiving and ServerInfo sending The server should handle HeartBeat messages in a similar way as the previous **tx** and **pb** requests. When a peer receives a HeartBeat message, it must log the corresponding ServerInfo into the ServerStatus with the current time. If a peer has not heard from a Server for four seconds, it must remove that ServerInfo from the ServerStatus. If this is the first time the peer Q hears about another peer P joining in, Q should also broadcast this information with **si|<Q's Port>|<P's IP>|<P's Port>** message to all neighbours different from peer P. Each peer thread should not try to connect with another peer for more than 2 seconds (to avoid blocking).

ServerInfo receiving and relaying Once a ServerInfo message is received. If the server S already knows P from before, it must not relay. Otherwise, the server relays the ServerInfo message to all its neighbours except to the originator of the ServerInfo message itself and the server P **si|<S's Port>|<P's IP>|<P's Port>**.

Task 2 Blockchain: Catch Up Protocol and Blockchain Consensus

So far you have encountered most basic building blocks of a distributed system, and now it is time for you to design and implement the last part on your own. However, there are some constraints you must follow. Therefore, two peers must communicate via the same protocol. The primary workflow for the last part is designed as follows.

- Periodically the server sends its latest block's hash to five randomly selected peers. (This is used to indicate "Hey, this is my latest block, Do you have it?")
- If a server needs to catch up, then it must set up a new connection and do the catch up. (This is used to indicate "Gotcha, let me catch up with you!")
- If a server does not needs to catch up, simply close the connection. (This is used to indicate "I am all good, see you soon!")

Communication For the LatestBlock message, the signature is defined as **lb|<Port Number>|<Blockchain's Size>|<Last Block's hash>**, the hash should be encoded in base64 string before concatenation with the request message. There is no reply needed for the LatestBlock message.

For CatchUp message, there are two types of signatures. The first type is defined as **cu**, the reply for this is the latest block (head block) from the blockchain. The second type is defined as **cu|<Block's hash>**. The hash should be encoded into base64 string before concatenation with the request message. The program should find the block with the given hash, transfer the block over sockets, and close the connection. (Hint: To transfer blocks over sockets, you need to use `ObjectInputStream` and `ObjectOutputStream`.)

When to Catch Up Using the two messages designed above, design an algorithm to synchronise different blockchains on different peers.

There are the three cases where you must issue a catch up request to your peer:

- At the very beginning, your program should catch up with the peer specified by program arguments.
- After receiving a lb request, if the length of your neighbour blockchain is longer than yours, a catch up is required.
- After receiving a lb request, if the length of your neighbour blockchain is the same to yours, but the hash is smaller than yours.

Hash comparison of type `Byte[]` is defined as assuming two bytes bi and bj from two hashes i and j at the same index, if bi is smaller than bj, hash i is smaller than hash j, starting from index 0.

How to Catch Up Catching up is about finding missing blocks and recovering from forks. Design your own algorithm and discuss with your tutor about the algorithm. (Hint: A naive solution is to catch up all blocks from head block to genesis block one by one from your neighbour, reconstruct all blocks into a new blockchain and replace your old blockchain with the new blockchain. This actually requires significant amount of network traffic compared to a more sophisticated algorithm, but it will pass all visible test cases.)

Even though no code should be shared, you are allowed (and encouraged) to test with others at the last stage of development.