Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

Lab4 Project Report

Distributed Blockchain System

Introduction

A simplified bitcoin blockchain project. Implemented several core functions of blockchain. The overall structure, mine a new block, consensus, bitcoin transaction(simplified), cryptography, verification and signature based on security library, decentralized p2p network, p2p websocket communication.

Design RoadMap

1. Blockchain design
    a. Mining algorithm
    b. Transaction
    c. Wallet
2. LabCoin design
    a. Consensus
    b. Blockchain cryptography
3. Transport design
    a. HTTP server with api
    b. P2P communication

1. Blockchain design

Blockchain design

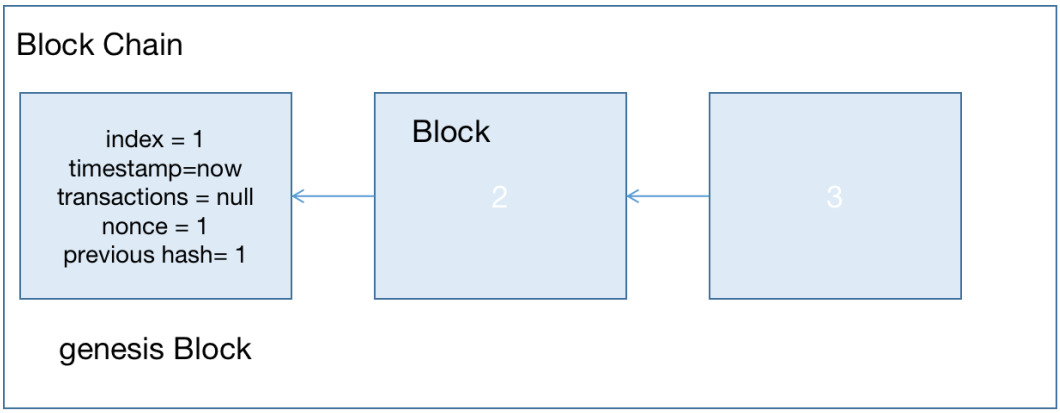


Fig 1. The blockchain diagram

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

1.1 Block Structure

```
/**
*      //*****************************************************
*      // ***************_________________________*****************
*      // ***************|         index         |*****************
*      // ***************|         hash          |*****************
*      // ***************|     previousHash      |*****************
*      // ***************|        timestamp      |*****************
*      // ***************|         nonce         |*****************
*      // ***************|_______________________|*****************
*      // ***************|____transactions_____  |*****************
*      // ***************|    Transaction1->     |*****************
*      // ***************|     ->Transaction2->  |*****************
*      // ***************|     ->Transaction3    |*****************
*      // ***************|_______________________|*****************
*/
public class Block {
   private final static int DIFFICULTY = 3;
   private int index;
   private String hash;
   private String previousHash;
   private long timestamp;
   private List<Transaction> transactions;
   private int nonce;
```

1.2 The genesis block

index starts from 1, timestamp is now(when it is created), transactions are null, nouce is 1, hash and previous hash are both 1.

```
Block genesisBlock = new Block(
     1, System.currentTimeMillis(), new ArrayList<Transaction>(),
     1, "1", "1");
```

1.3 Proof of work and the calculation of Block hash

Use SHA256 Hash in `java.security` packet, content to hash is:

```
Hash = SHA256(lastblockhash, transactions, nouce)
```

Calculate hash until the hash meets the DIFFICULTY requirements(starts with DIFFICULTY numbers of '0');

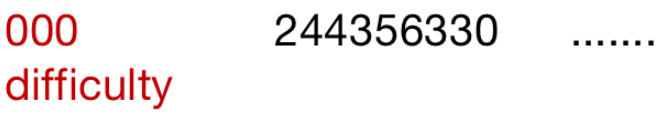


Fig 2 Difficulty illustration

Default difficulty is set to 3, can be edited in block-*DIFFICULTY*
In this design, the block hash is calculated by

```
SHA256(index + transaction + previous hash + nonce)
```

Because a transaction is a list, we should parse it to a string. This project adopts json parse.

1.4 Transactions(Transaction, transactionInput, transactionOutput)

Transactions that occur on a blockchain must be validated by a number of the network's participants.

A transaction is only valid once the participants verify the transaction and a consensus is reached about its validity.

The transaction details are then recorded on the block and distributed to the network's participants.

The 'number' is encrypted by the owner's hash. New UTXO is created by mining or transaction. UTXO was stored in the entire blockchain. The sender will send from his own UTXO and encrypt it with the recipient's public key.

1.4.1 transactionInput and transactionOutput

When a block is mined, it will contain a base transaction(system reward), the base transaction contains only output and requires no signature.

When a peer wants to make a transaction, the transaction will find previous hash,

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

The sender can only send assets belonging to themselves(public key hash should match the sender's public key).

Transaction:

```
public class Transaction {
  /**
   * the globally globally unique ID, AKA UUID
   */
  private String id;
  /**
   * transaction input, indicate the input source of this transaction
   */
  private TransactionInput txIn;
  /**
   * transaction output, indicate the UTXO of this transaction
   */
  private TransactionOutput txOut;
}
```

TransactionInput & TransactionOutput

```
public class TransactionInput {
  /**
   * the previous transaction's ID
   */
  private String txId;
  /**
   * the value of the input
   */
  private int value;
  /**
   * the signature of the sender
   */
  private String signature;
  /**
   * the public key of the receiver
   */
  private String publicKey;
}
```

```
public class TransactionOutput {
  /**
   * the value of this UTXO
   */
  private int value;

  /**
   * the scripted public key of this
receiver
   */
  private String publicKeyHash;
}
```

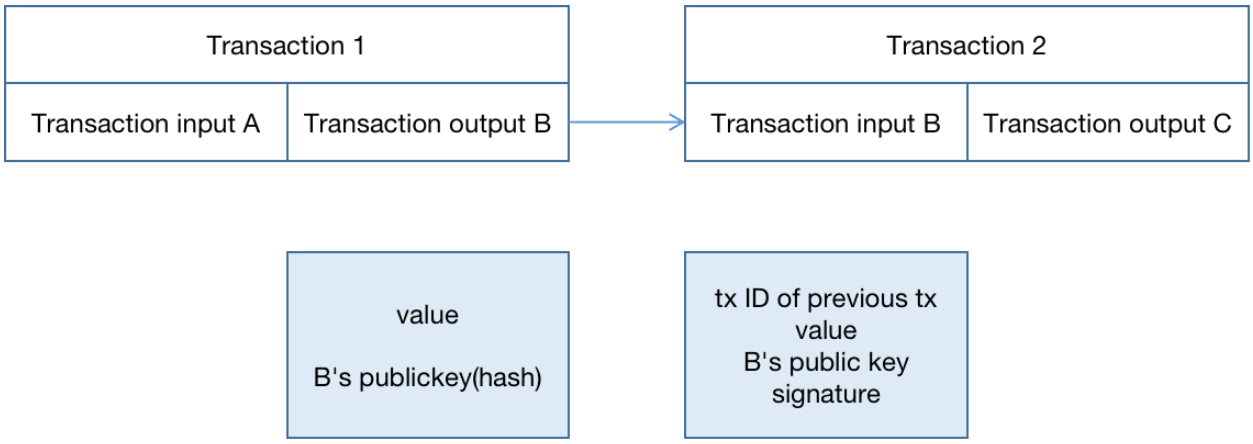


Fig 3. The structure of Transactions in a block

When a transaction is generated, it will be recorded locally as UTXO(unpacket transactions) until the next block is generated. During this process, the sender's UTXO will be consumed but the recipient hasn't received the amount of virtual asset.

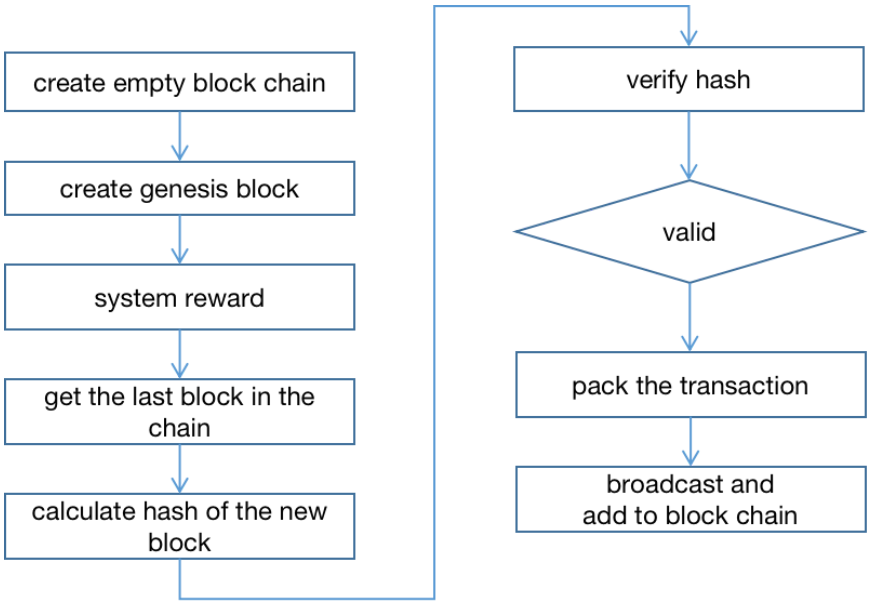


Fig 4. the work process

1.4.2 Labcoin balance

The balance is calculated by UTXO.

When a user A wants to query for his balance, the system will iterate the UTXO list and find out all the outputs that *belongs to A and calculates the sum of the virtual asset.

A UTXO belongs to A if the UTXO's public key(but in this design, publickeyhash) matches A's.

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

1.6 Labcoin wallet

Structure

```
public class Wallet {
   private String publicKey;
   private String privateKey;
}
```

Wallet is a pair of 'shorter name' for public key.

Peers will have access to all the information of its own wallets and the public key information of others' wallets.

The Wallet is anonymous, a user can see the details of wallets created by himself(public key, private key), but can only see the address and public key hash for others' wallets.

When making a transaction, a user can use the address of the wallets as sender and recipient to process the transaction.

my wallet map

Address	**Addr1**	**Addr2**	**Addr3**	**Addr4**	**...**
Wallet	Wallet1 publickey1 privatekey1	Wallet2 publickey1 privatekey1	Wallet3 publickey1 privatekey1	Wallet4 publickey1 privatekey1	...

other's wallet map

Address	**Addr1**	**Addr2**	**Addr3**	**Addr4**	**...**
public key (hash)	publickeyhash1	publickeyhash1	publickeyhash1	publickeyhash1	...

2.Labcoin design

2.1 Consensus

If two peers mine a block successfully, keep the two blockchains locally until one is longer than the other one. The longer one will overwrite the other's local records until all achieve the same record.

When a peer wants to participate in the system, it will need to download all the information. When a peer has been disconnected for some time, it will need to update it's local cache.

Compare the global blockchain and the local blockchain.

If the last block of the received blockchain has a higher index than the local blockchain:

- If the last one is exactly what the peer misses, directed add the block to local blockchain
- If the peer has missed more than one block, it should broadcast and ask all peers for the longest chain. Then pull down the longest chain.

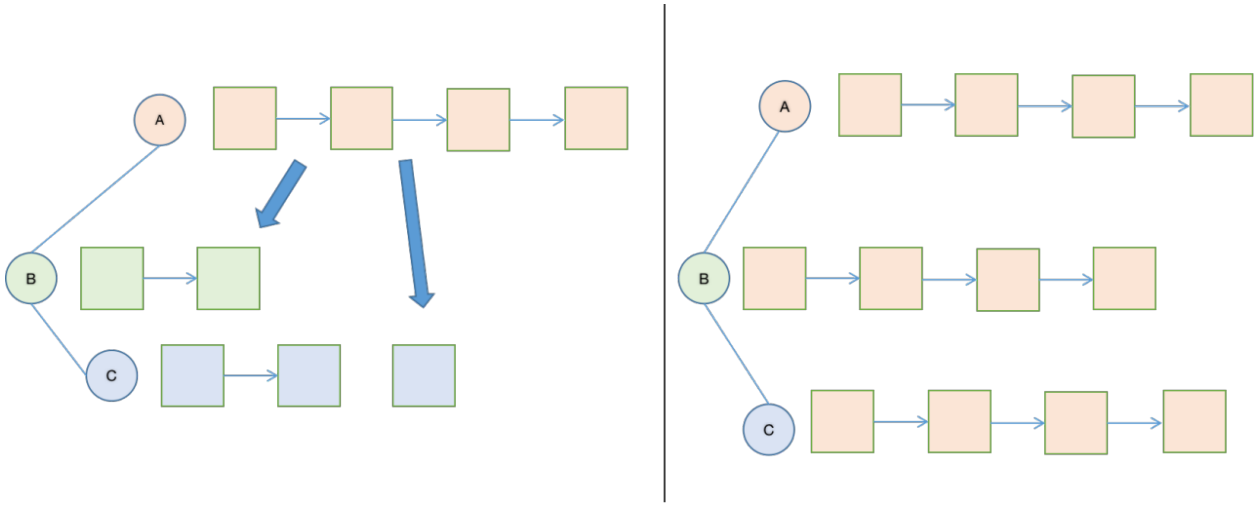


Fig 5. Consensus of the chain

2.2 Cryptography

Asymmetric cryptography
Use RSA algorithm to generate key pair
Use MD5 to generate signature
Both in `java.security.*`

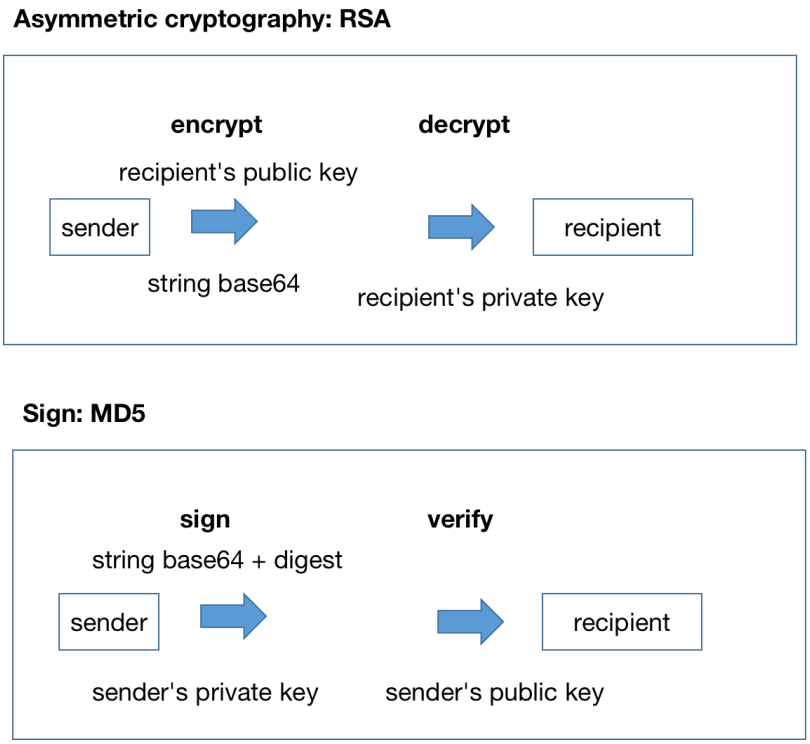


Fig 5. The encryption and sign

3. Transportation Layer:

3.1 HTTP Server by jetty

1. client->server, request for data
2. Server receives request, call server service, return data response to client

Client receives the response, calls client service and deals with data.

HTTPSever api:

api	method	function	param	Broadcast or not
`/chain`	GET	get blockchain	/	no
`/wallet/create`	POST	create wallet	/	yes
`/wallet/get`	GET	Get Wallet get others wallet	address=<wallet address>	no

`/wallet/get/other`	GET	Get others' wallets	/	np
`/wallet/get/balance`	GET	Get balance of a specific wallet	address=<wallet address>	no
`/mine`	POST	Mine a block	address=<wallet address>	no
`/transactions/new`	POST	Propose a new transaction from sender to receiver	sender=<sender's wallet address> recipient=<recipient's wallet address> amount=<value>	Yes But needs to wait for another mine and get consensus of all others.
`/transactions/get/packed`	GET	Get packed transactions in this node	/	no
`/transactions/get/unpacked`	GET	Get unpacked transactions in this node	/	no
`/transactions/get/all`	GET	Get all transactions in this node	/	no
`/peers`	GET	Get peers connect to this node	/	no

3.2 P2P : P2P decentralized network

P2P means each node can be a client or server at the same time(but at different ports). In the project, the P2P design is adapted from easy chatroom with websocket.

P2P client will be able to call client services. P2P server will be a basic server handling messages from clients by calling functions in labcoin implementation.

There are some operations that will require broadcast.

1. When A wants to start a transaction, it should tell all the peers or others who will not be able to know about the transaction.
2. When A mined out a new block(faster than others), it should broadcast the block to all others. Other peers will decide whether to upload their own chain.
3. When A generates a new wallet, it should broadcast it's address and public key to all peers.

3.

service	**message**	**function**
Get chain/ Blockchain broadcast	The json string of peer's chains	If the received block chain is newer, addBlock() if received a new last block. This is the normal add function or replaceChain() if more than 1 is added or the previous block does not match. This will happen when a peer is disconnected for a while and reconnects to the p2p network. Broadcast and ask all nodes for the chain if the received blockchain is genesis block. This will happen when a new peer joins the p2p network.
Deal with wallet request	The json string of others newly create wallet address and public key	Add to others' wallet map pool
Get all local wallets	The json string of get wallet query	Iterate my wallets and others' wallets, return a message containing all wallet lists
Generate all required queries as client	A json string carrying message type and message	Generate different service queries for other peers as a p2p client.
Peer communication	A json string carrying message type and message	websocket.send()

4. Project build:

4.1 how to build the project

```
mvn clean install
```

4.2 how to start a node

The peer node can be started with 1 or 2 or 3 arguments. The first argument is the port to start `HTTP service`, the second argument is the port to start `p2p service`, the third argument is the peer it wants to connect to.

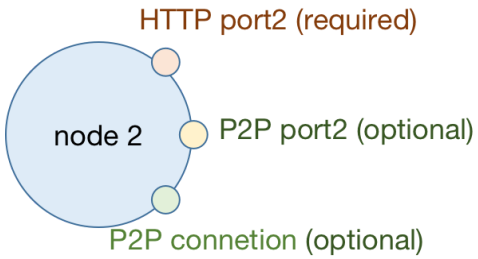


Fig 6. The there arguments of a peer node

The port can be freely chosen as long as is it unique. The peer websocket address is `ws://localhost:p2p_port_to_connect`

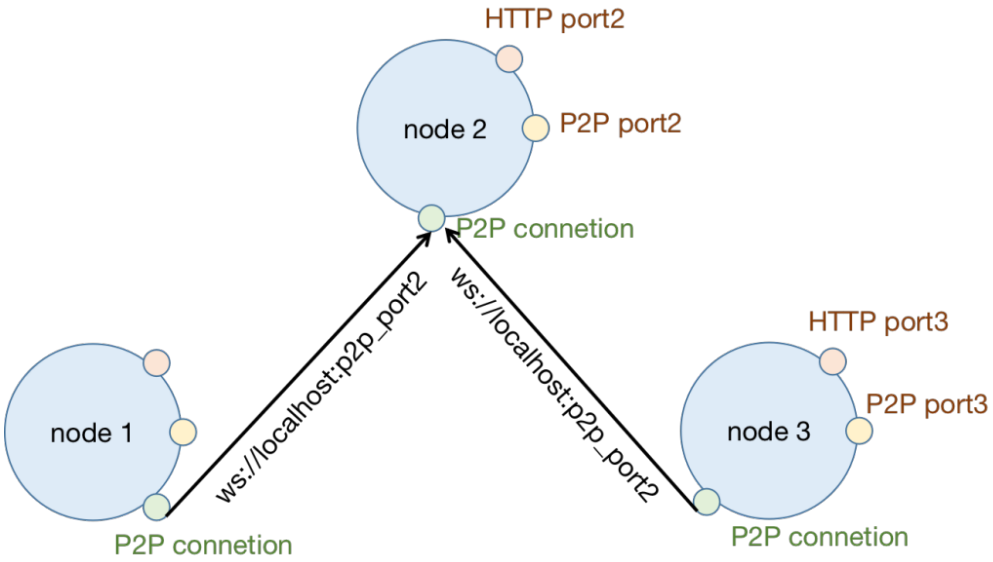


Fig 7. The illustration of connecting to p2p network

```
mvn clean install
cd target
```

Node A:

```
java -jar labBlockChain-1.0-SNAPSHOT.jar 1234 5678
```

Node B(connect to A):

```
java -jar labBlockChain-1.0-SNAPSHOT.jar 2345 6789
ws://localhost:5678
```

Node C(connect to A):

```
java -jar labBlockChain-1.0-SNAPSHOT.jar 3456 7890
ws://localhost:5678
```

4.3 project api in terminal

4.3.1 Current BlockChain(GET)

```
curl http://localhost:1234/chain
```

4.3.2 Create Wallet(POST)

```
curl -i -X POST http://localhost:1234/wallet/create
```

4.3.3 Mine(POST)

```
curl --data "address=78415FA4D1BCFFF8AFBE5DD5C565C144"
http://localhost:1234/mine
```

4.3.4 Transaction(POST)

```
curl -H "Content-type:application/json" --data '{"sender":
"9555D32727BD391C0A1B3FBE6C53FDBE","recipient":
"F784B5402BBCE003875E21C77550590A","amount": 1}'
http://localhost:1234/transactions/new
```

4.3.5 Query for UXTOs(GET)

```
curl http://localhost:1234/transactions/get/unpacked
```

4.3.6 Query for all Transactions(GET)

```
curl http://localhost:1234/transactions/get/all
```

4.3.7 Query for packed Transactions(GET)

```
curl http://localhost:1234/transactions/get/packed
```

4.3.8 All wallets in this node(GET)

```
curl http://localhost:1234/wallet/get
```

4.3.9 Other's Wallet Balance(GET)

```
curl "http://localhost:1234/wallet/get/other"
```

4.3.10 Wallet Balance(GET)

```
curl
"http://localhost:1234/wallet/get/balance?address=9555D32727BD39
1C0A1B3FBE6C53FDBE"
```

4.3.11 see socket peers connected(GET)

```
curl http://localhost:1234/peers
```

5. Test

5.1 Local Maven Test:

We implemented three local test cases which test the blockchain mining and transaction functionalities on a single machine. They are:

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

5.1.1 local Block OperationTest:

this testcase tests for mining a local block. We checked if the number of zeroes of the block hash matches the difficulty level. We also checked if the previous hash is indeed the previous block's hash.

5.1.2 stressBlockOperationTest:

this testcase is an extension of the previous one. It mines 100 blocks consecutively, and checks the validity of the appended block in every single iteration.

5.1.3 transaction API Test:

this testcase tests the functionality of making a transaction between wallets.

In our case, we define a dummy p2p network with only one client and one server. The client has two wallets, sender and receiver.

We will first mine 1 block in the sender wallet, and then send 1 LabCoin from sender to the receiver. When we get back the HTTP response, we will check if the transaction succeeds by asserting that the transaction is not a base transaction and the transaction amount is as expected.

5. 2 Demo Test:

1. Start A, A will create a genesis block
2. A create wallet A1, A2, both of these should have balance 0, should raise broadcast.
3. A1 mines a block, and should get a reward from the system, should raise broadcast.
4. Now start another peer B, B will create a genesis block
5. Create wallet B1, the balance should be 0, should raise broadcast.
6. A intend to make a transaction from A1 to B1, should raise broadcast

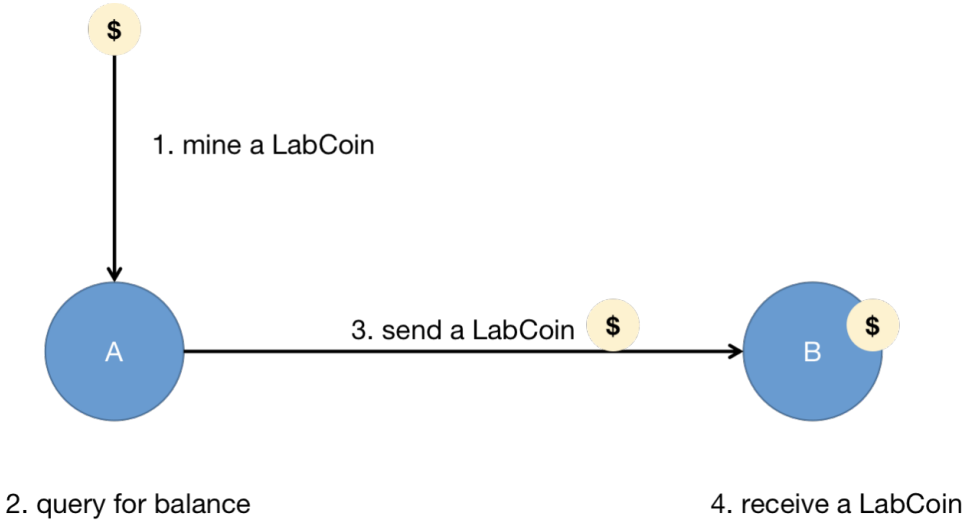


Fig 8. The brief process of demo

6. Points

6.1 Points Breakdown on Coding Tasks:

Local block mining/POW functionalities	20	finished
Broadcast, read, write, validation on blockchain	40	finished
Schema that handles concurrency issues on operations and helps reach the consensus on decisions	40	Finished and can reach consensus. Limited concurrency situation with `synchronization`.
Transportation Layer	20	Finished by HTTP server
Transaction verification and real p2p communication scheme	Bonus: 20 (each 10, if bonus is allowed)	Finished simplified transaction(send 1 coin each time). Finished by implementing

		real p2p

6.2 Points Breakdown on Test cases:

Local test cases in maven Usage: mvn test (The test cases are developed when we are implementing blockchain operations on the local machine)	30 LocalOperationTest - 10 StressOperationTest - 10 TransactionAPITest - 10	Passed when run `mvn clean install`
Remote APIs Usage: demo on postman (All APIs except the consistency check)	50 (5 for each, 10 in total)	Need to be demoed
Consistency check Usage: demo on postman (after mining and transactions, do `curl http://localhost:port/chain` for every single client)	10	

7. Conclusion

In this project, we designed and implemented a distributed blockchain system with a decentralized p2p network. Every user in our system, as a p2p client, is able to create a wallet, mine a block, make a transaction to another wallet, and connect to peers.

Our development process follows the test-driven development style. We realized that distributed systems are particularly hard to test on maven, so we first generate test cases that test local functionalities and modules. We also added a stress test on blockchain validity to make the local module robust enough. Then we begin developing the distributed part using Jetty and Java

Spring 2021, Lab 4 proposal

14736 Di Lu(dilu), Yuan Gu(guy)

Servlet to develop APIs for distributed operations. We first wrote shell scripts to test distributed functionalities, but for the demo purposes, we choose postman due to its intuitive UI.

We have finished some challenging tasks like transactions. Our transaction is simplified and can only support 1 coin transaction each time. We can implement more complex transactions by modifying the structure of the transaction class. Change inputs and outputs to lists, change the current signature method and we can implement coin aggregate and exchange. Further, the list of transactions can be manipulated and structured by merkle tree, but we did not have time to try it.

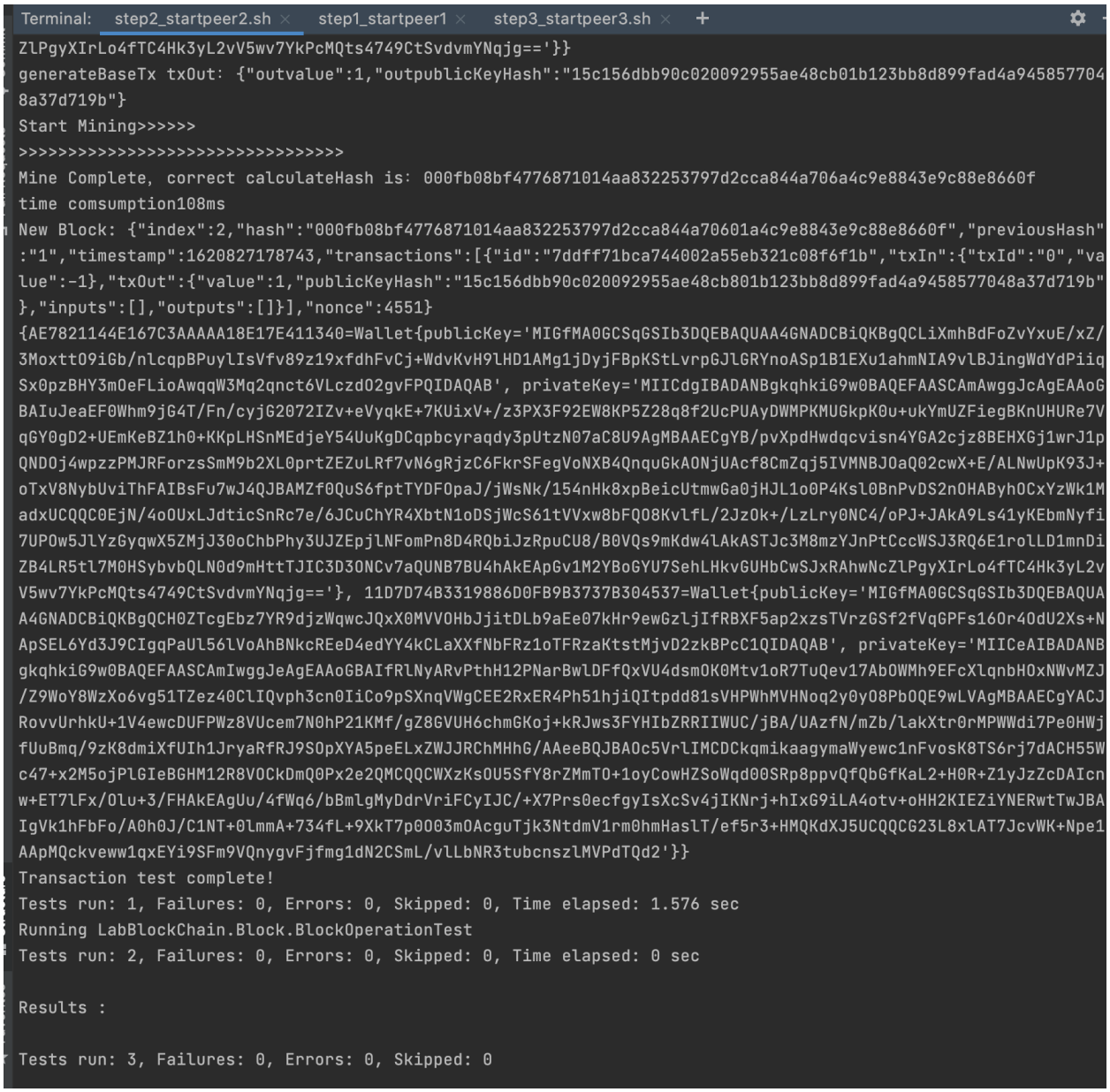
The cryptography we use is RSA and MD5, the real cryptography is ECDSA. But as asymmetric cryptography, RSA is enough for our lab.

We simplified the consensus of the distributed peers in order to spare time for p2p design. We only considered the synchronized keyword. The p2p design is based on chatroom, with server and client dealing with structured messages.

8. APPENDIX: Screenshots of test and demo

8.1 Local test:

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.576 sec
Running LabBlockChain.Block.BlockOperationTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

8.2 remote test

8.2.1 Start node A, B

Start node A:

```
java -jar labBlockChain-1.0-SNAPSHOT.jar 1234 5678
```

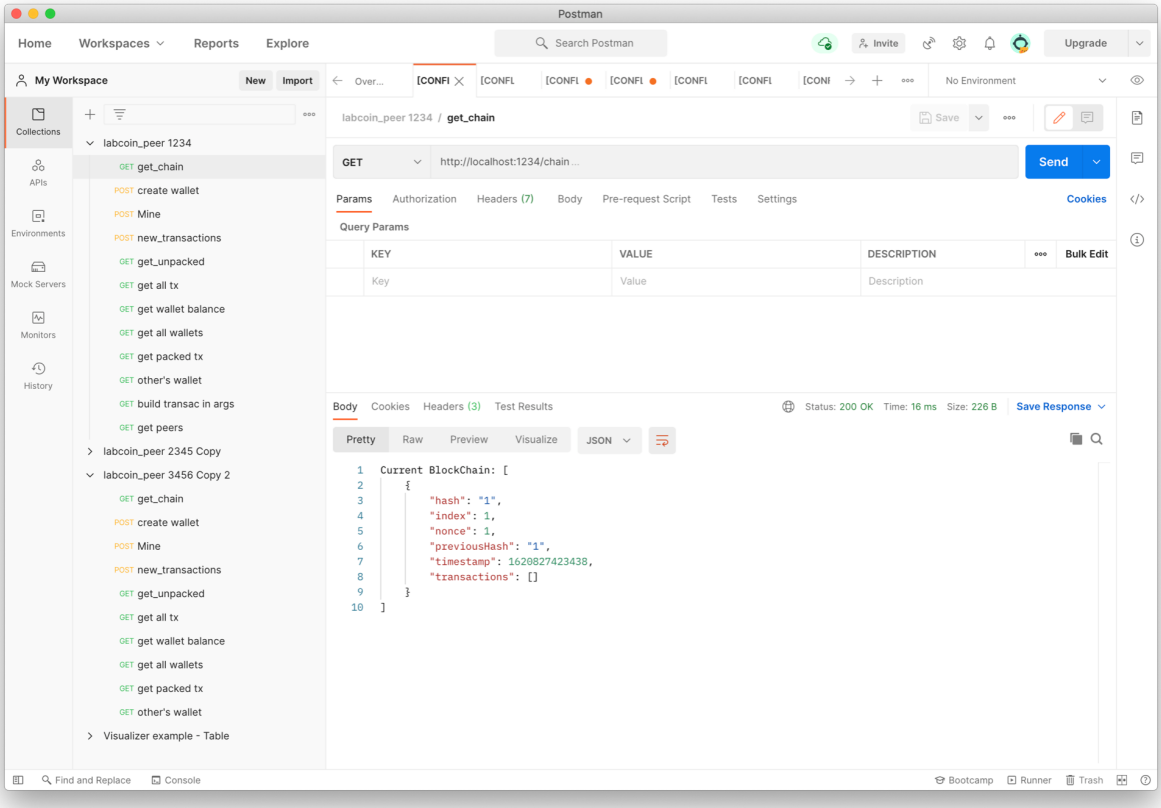
Start node B: connect to A

```
java -jar labBlockChain-1.0-SNAPSHOT.jar 2345 6789 ws://localhost:5678
```

Let A get the current block chain: GET http://localhost:1234/chain

8.2.2 check genesis block

When A is created, the genesis block is created.

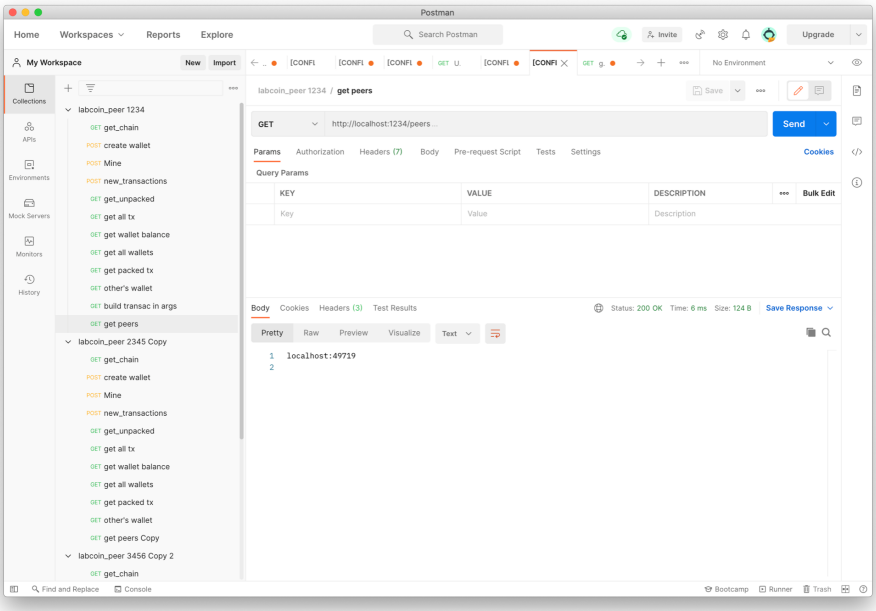


Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

8.2.3 check /peers

Let A GET peers: Get http://localhost:1234/peers

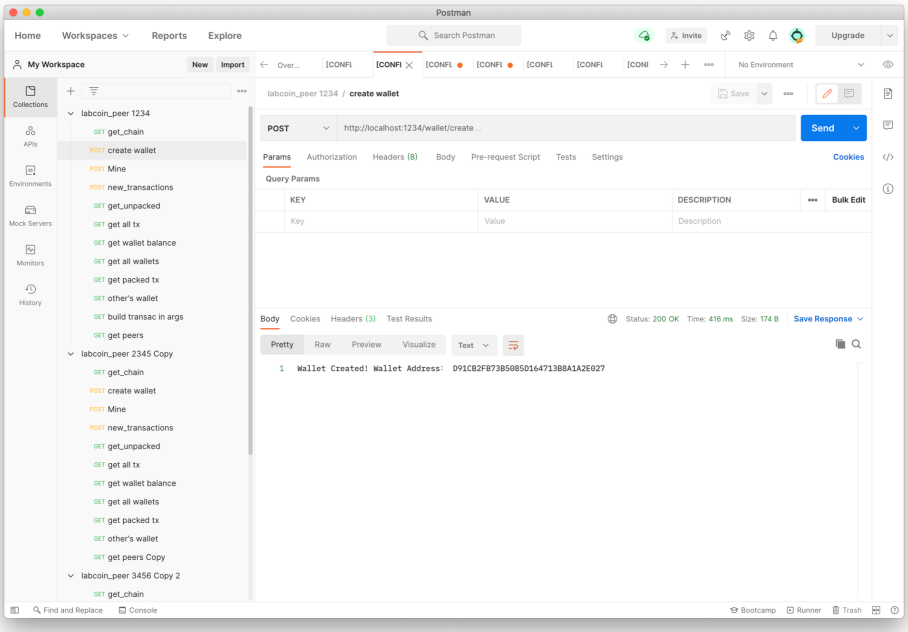
A is connected with B. The port is B's p2p working por t(not the one we start p2p service, the listen port).



8.2.4 test wallet operation

Now let A create wallet A1
POST http://localhost:1234/wallet/create

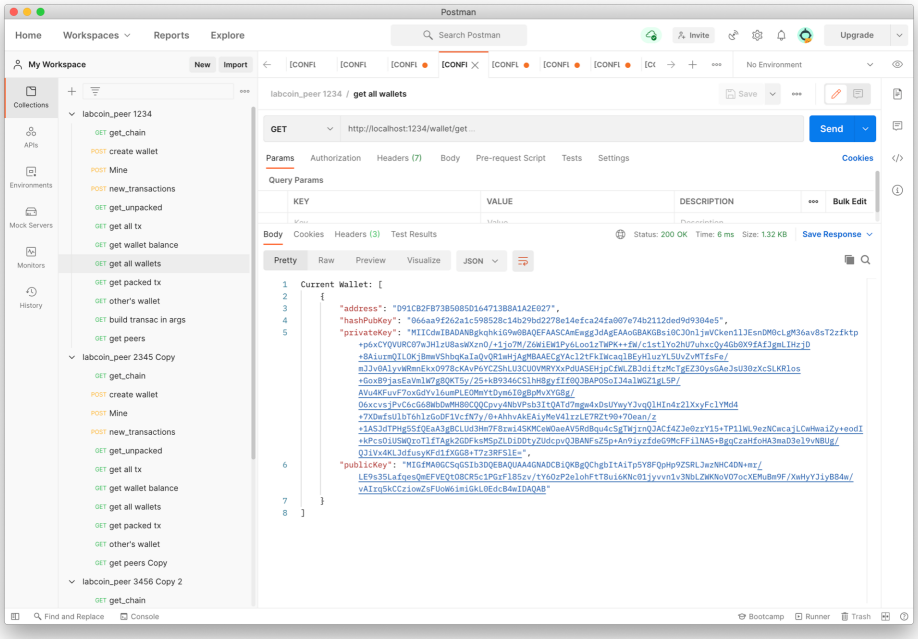
Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



A wallet is created, address is D91CB2FB73B5085D164713B8A1A2E027

Now Get all wallets in this node:
GET http://localhost:1234/wallet/get

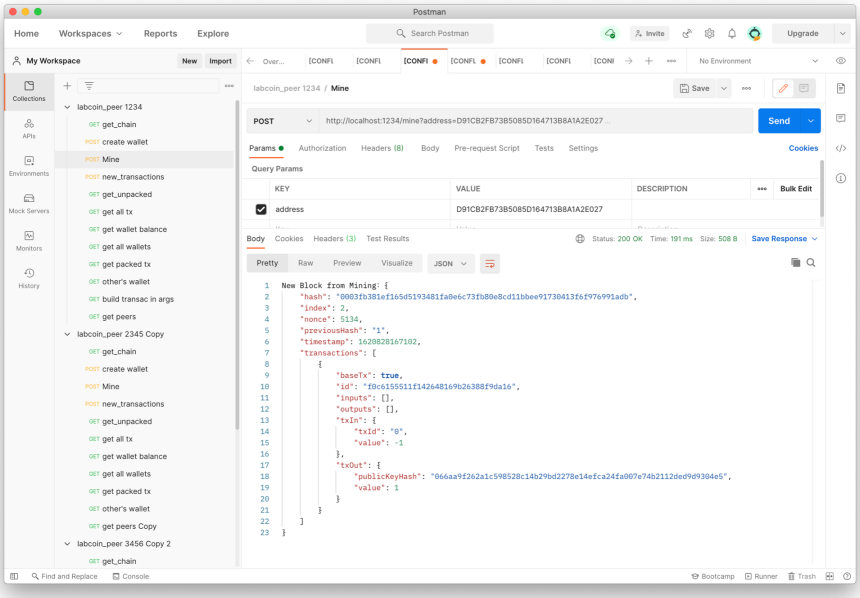
A can see the public key and private key of his wallet.



Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

8.2.5 test mining and consistency

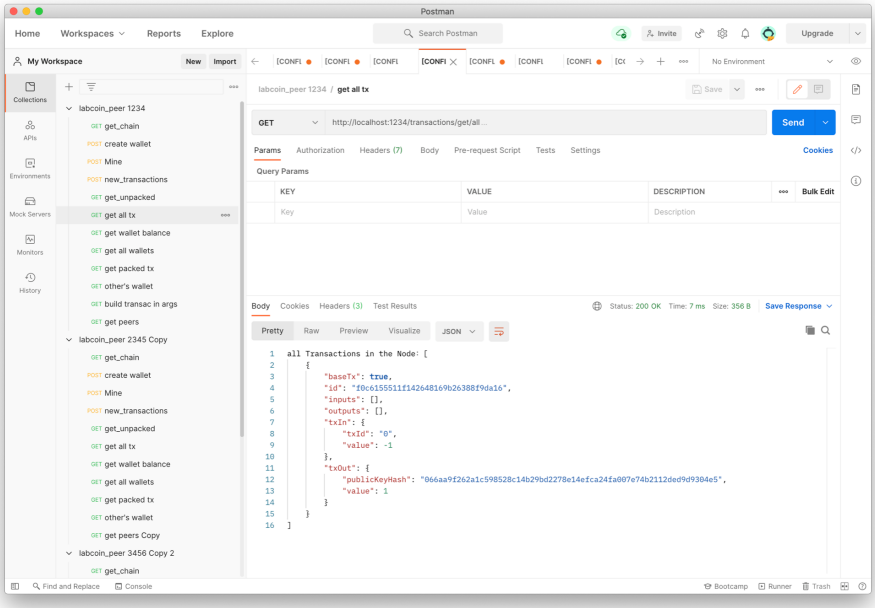
Let A Mine a block to the A1 wallet:
POST http://localhost:1234/mine?address=D91CB2FB73B5085D164713B8A1A2E027



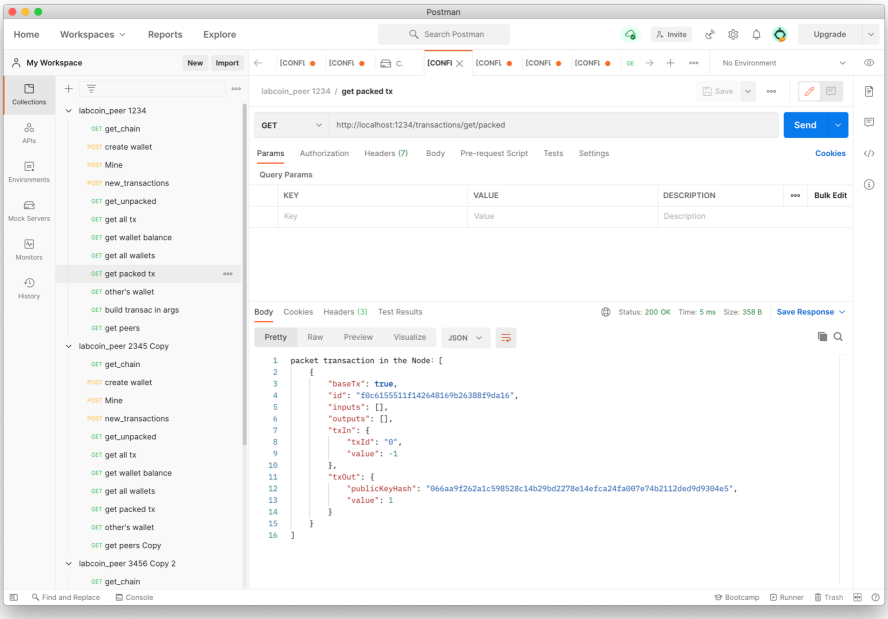
A block with base transaction (system reward) will be created.

Let A get the current all transactions:
GET http://localhost:1234/transactions/get/all

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



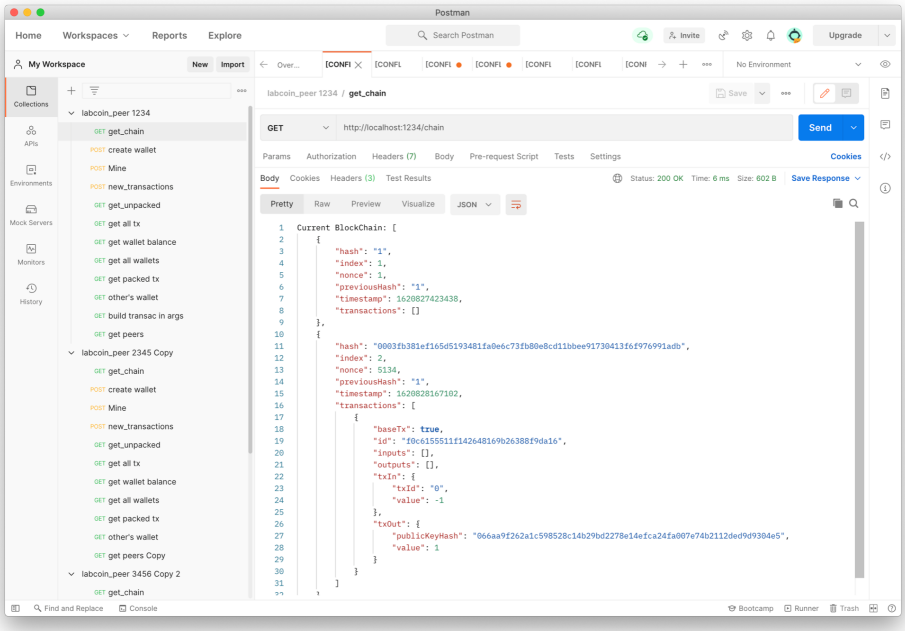
Let A get the current packed transactions:
GET http://localhost:1234/transactions/get/packed



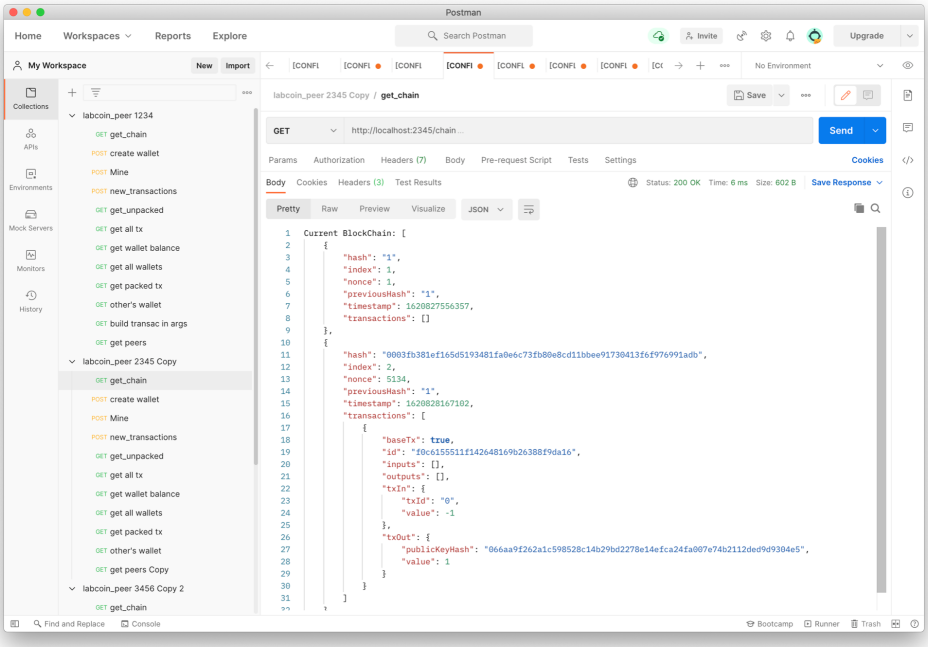
They are the same, all transactions are packed. A have no UTXO now.

Current blockchain in A:
GET http://localhost:1234/chain

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



The blockchain is updated, the newly mined blockchain is successfully added.



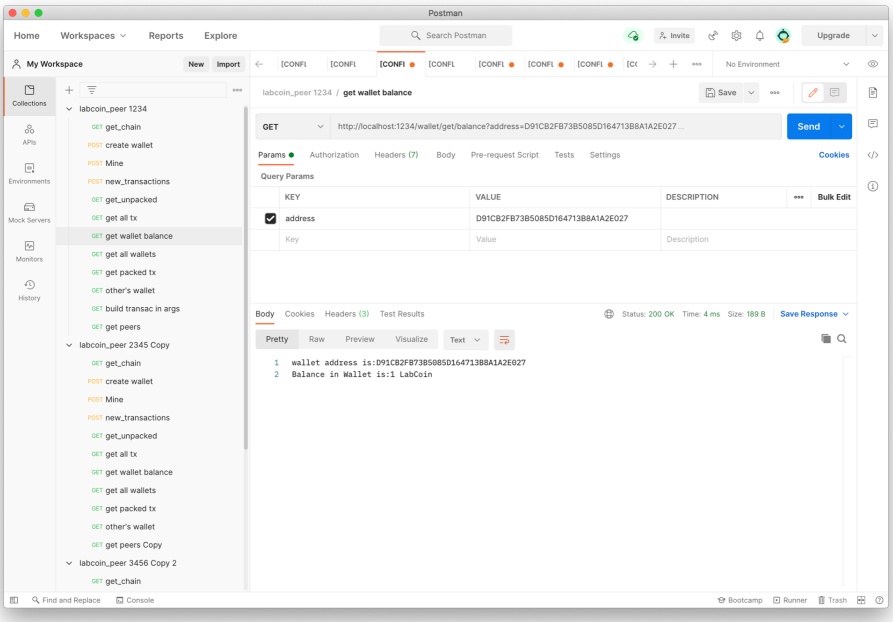
The previous hash matches the hash of the previous block(the genesis block). The new transaction ID is f0c6155511f142648169b26388f9da16

See A's wallet A1 balance now:

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

GET
http://localhost:1234/wallet/get/balance?address=D91CB2FB73B5085D164713B8A1A2E027

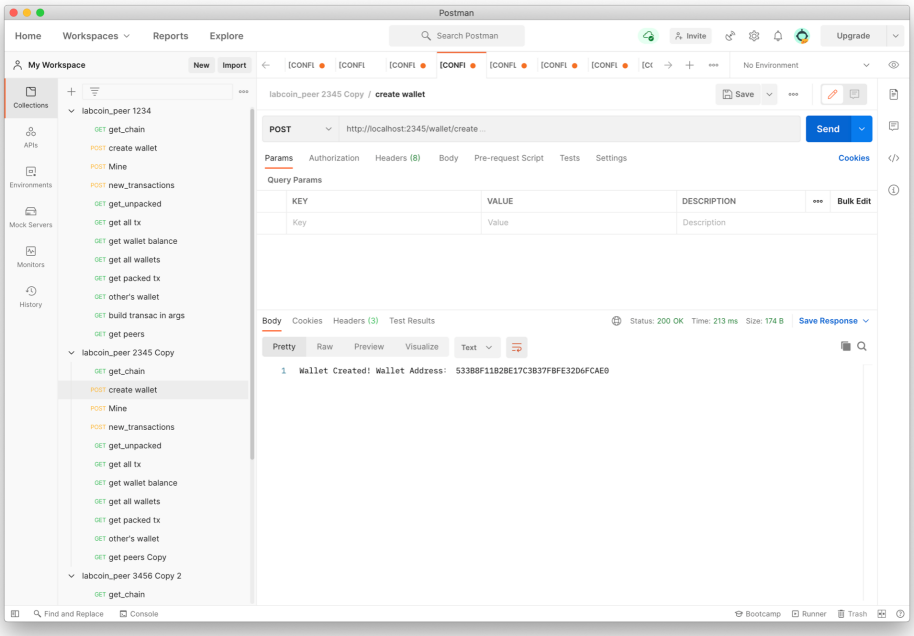
Is 1 labcoin (from system reward).



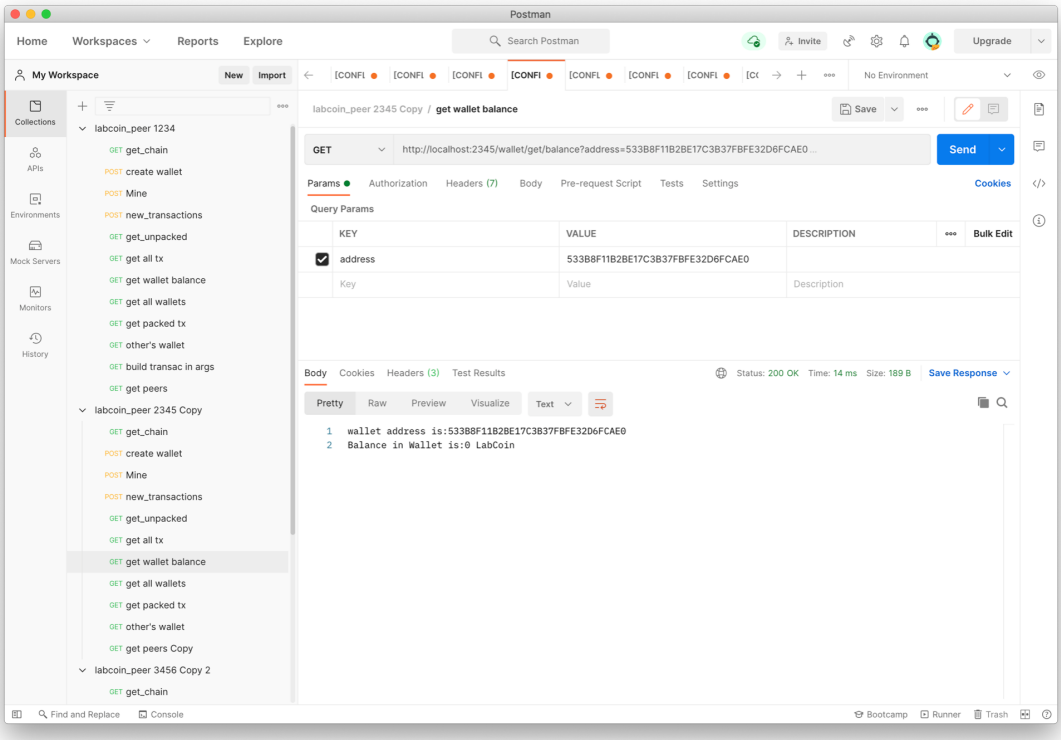
8.2.6 test transactions and consistency

Now let B create a wallet B1

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



B's wallet balance now:
GET
http://localhost:2345/wallet/get/balance?address=533B8F11B2BE17C3B37FBFE32D6FCAE0

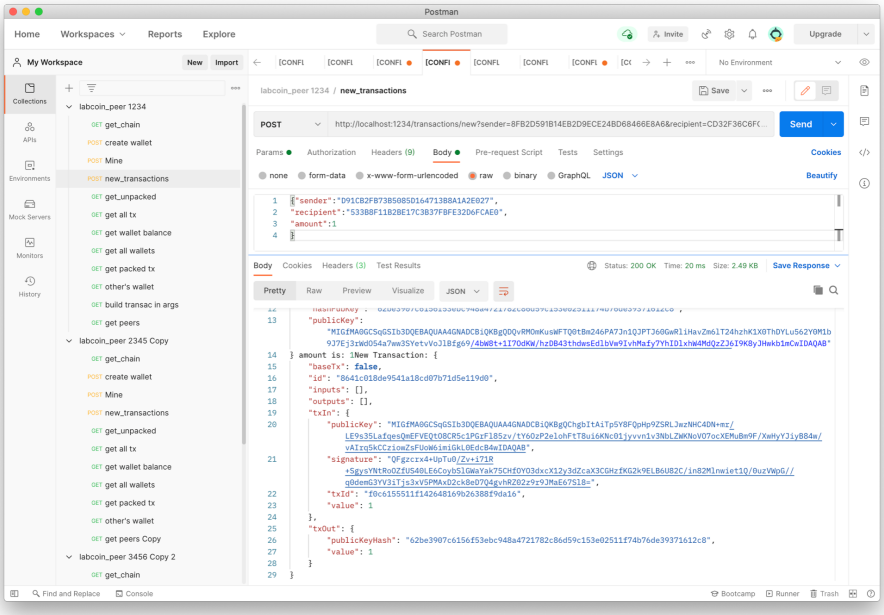


Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

Now A's wallet1 proposed a transaction to B:
POST http://localhost:1234/transactions/new

arguments:

{"sender":"D91CB2FB73B5085D164713B8A1A2E027",
"recipient":"533B8F11B2BE17C3B37FBFE32D6FCAE0",
"amount":1
}



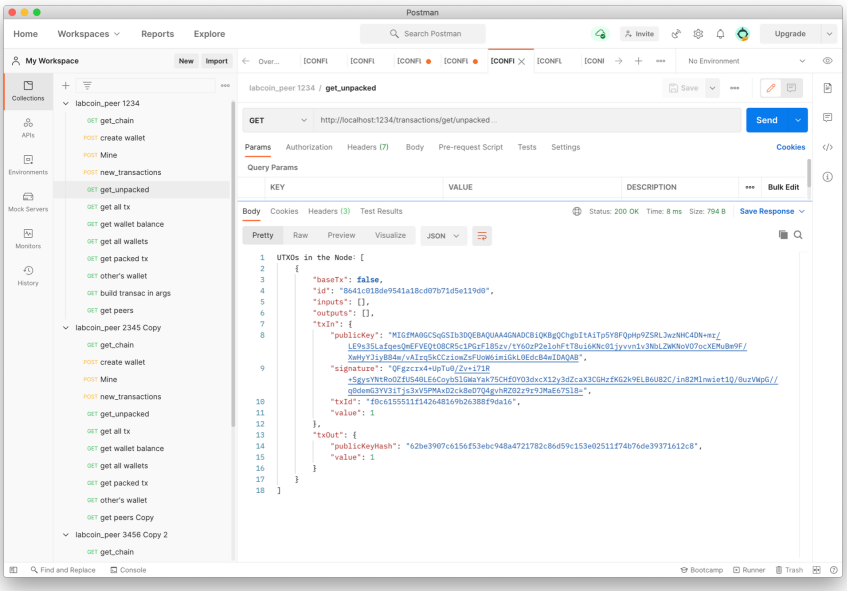
A new transaction is created locally.

The input transaction id is "txId": "f0c6155511f142648169b26388f9da16". Equals to the previous transaction ID. The value is 1, the output public key hash is B's public key hash.

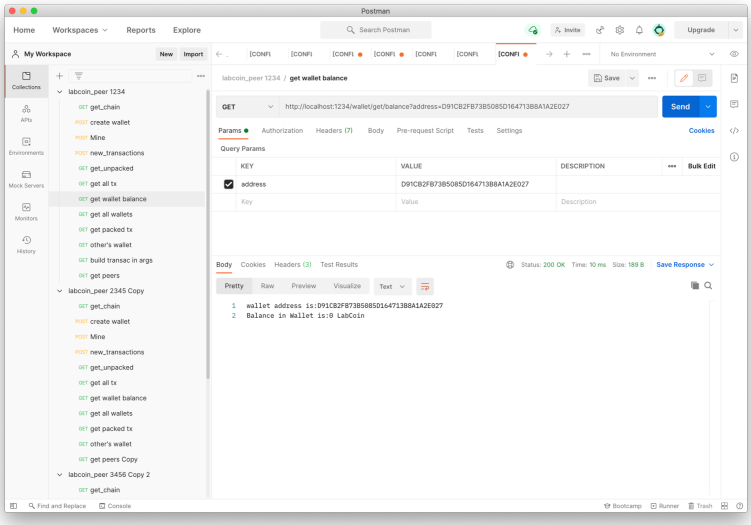
A's current unpacked transaction:
GET http://localhost:1234/transactions/get/unpacked

This transaction is not yet approved by all the nodes so is local.

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



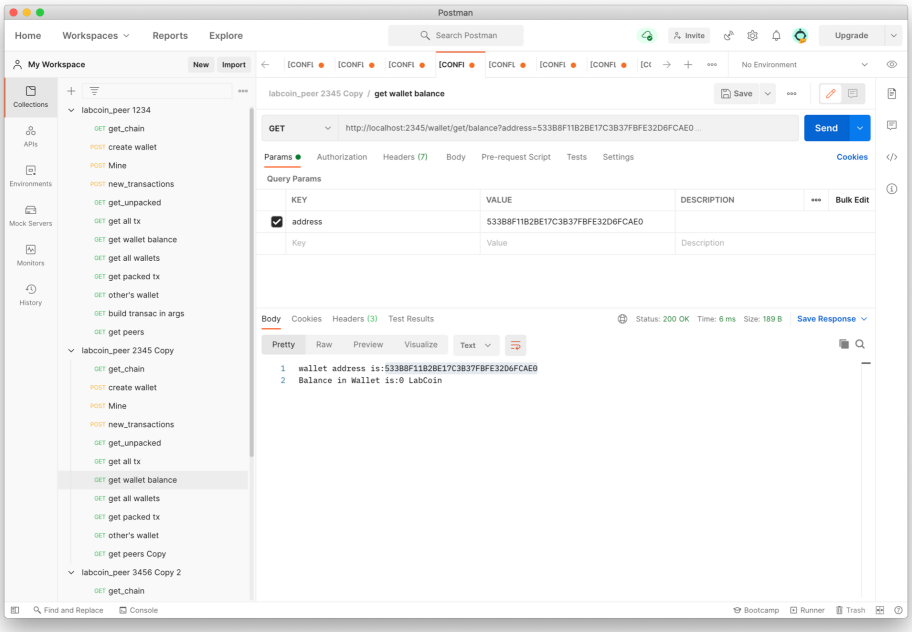
A's current balance is reduced anyway to prevent repeatedly using the same coin.
GET
http://localhost:1234/wallet/get/balance?address=D91CB2FB73B5085D164713B8A1A2E027



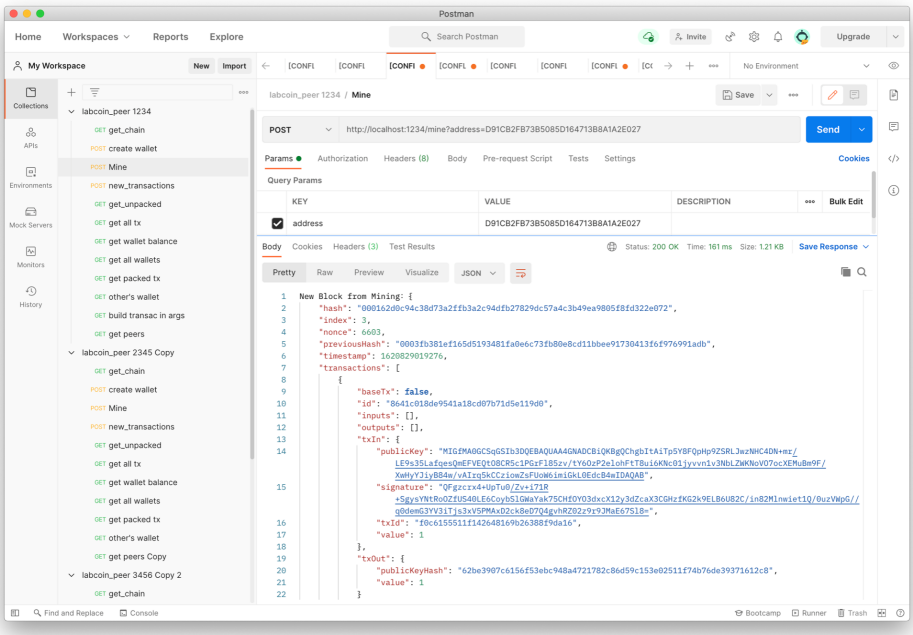
B's current balance:
GET
http://localhost:2345/wallet/get/balance?address=533B8F11B2BE17C3B37FBFE32D6FCAE0

Still 0, again, the transaction is not agreed by now.

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

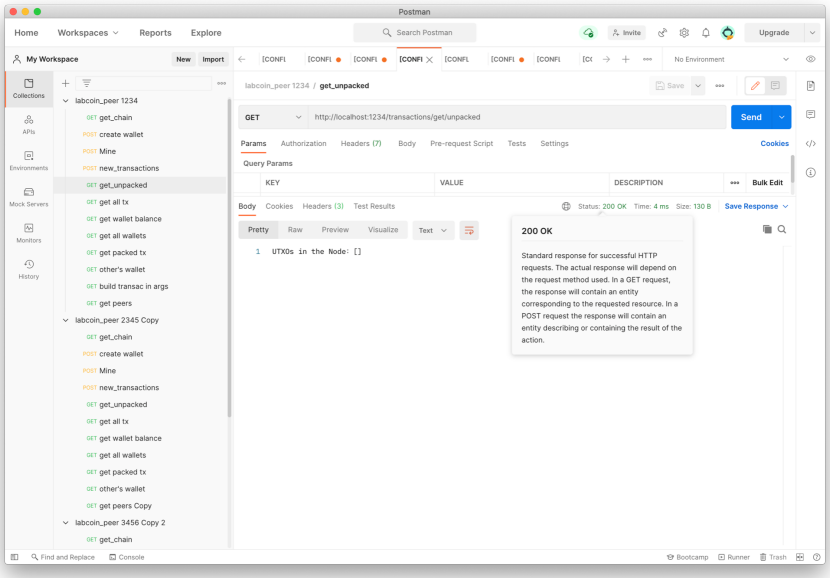


Now if A mines a new block:
POST http://localhost:1234/mine?address=D91CB2FB73B5085D164713B8A1A2E027

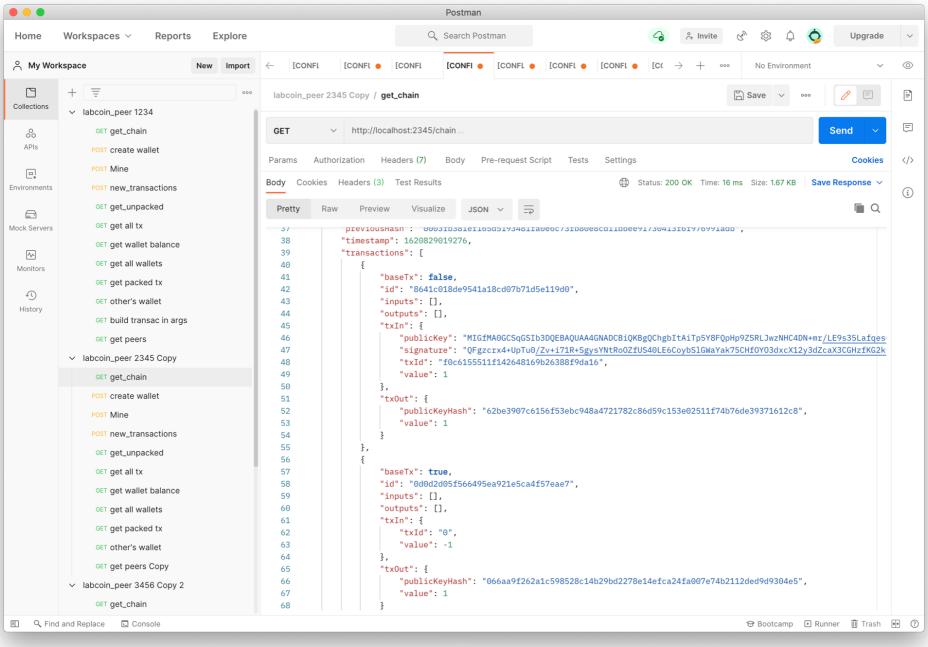


The transaction will be packed by the new block, A now have no unpacked transaction.
GET http://localhost:1234/transactions/get/unpacked

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

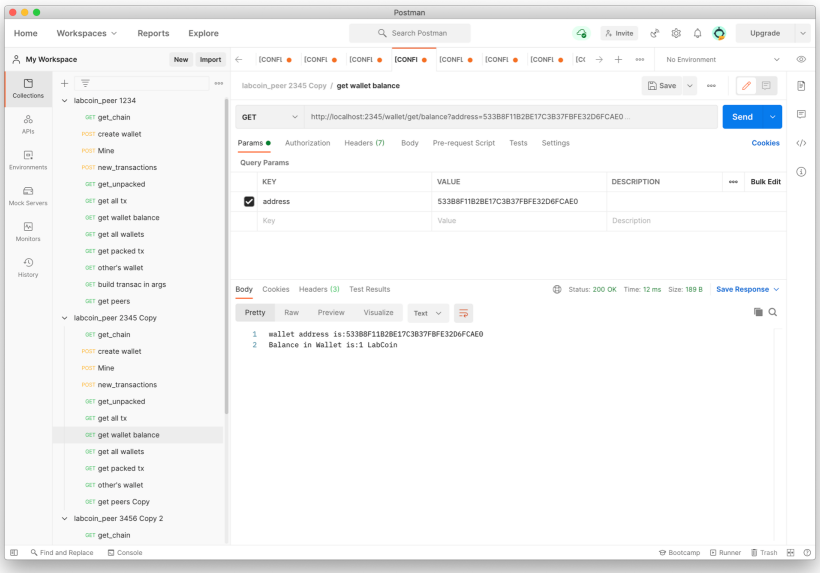


Now B's chain will be updated because A's chain is longer(2 blocks vs 1 block)
GET http://localhost:2345/chain

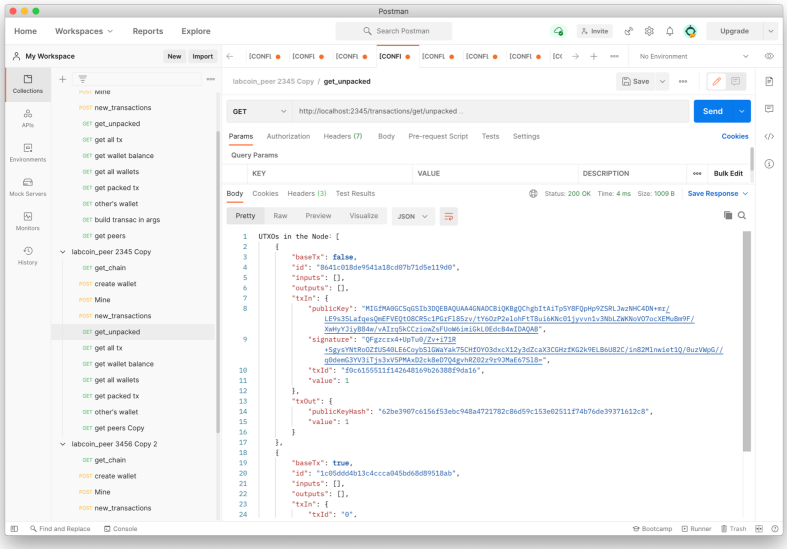


B's wallet B1 balance will be 1 now(the 1 from A), even B did not do any mining.
GET
http://localhost:2345/wallet/get/balance?address=533B8F11B2BE17C3B37FBFE32D6FCAE0

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



B's UTXO is updated, he will check whether this UTXO is his when making a transaction.
GET http://localhost:2345/transactions/get/unpacked



8.2.7 test new added peers and consensus

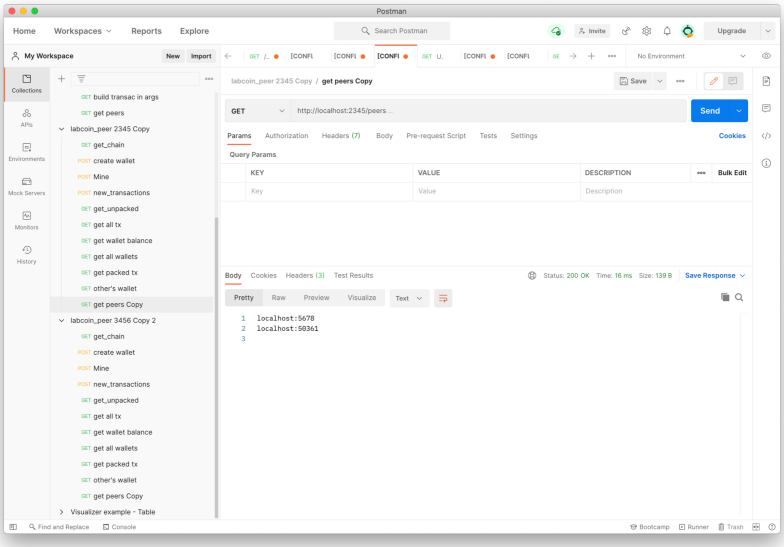
NOW Start a new node C, connect to B

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

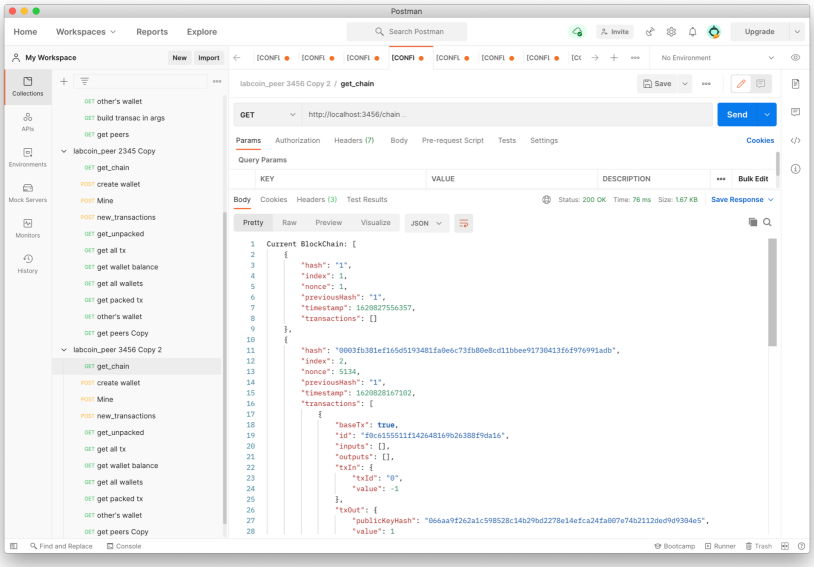
```
java -jar labBlockChain-1.0-SNAPSHOT.jar 3456 7890 ws://localhost:6789
```

B now now two peer connections:
GET http://localhost:2345/peers

One is A's p2p listen port, One is C's p2p working port.

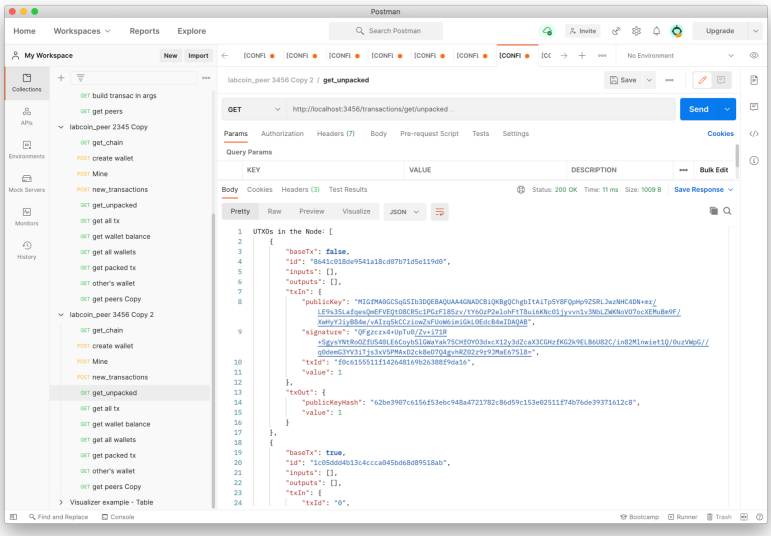


C is newly created, but it will receive the longest blockchain because it's length is 1(the genesis block)
GET http://localhost:3456/chain.



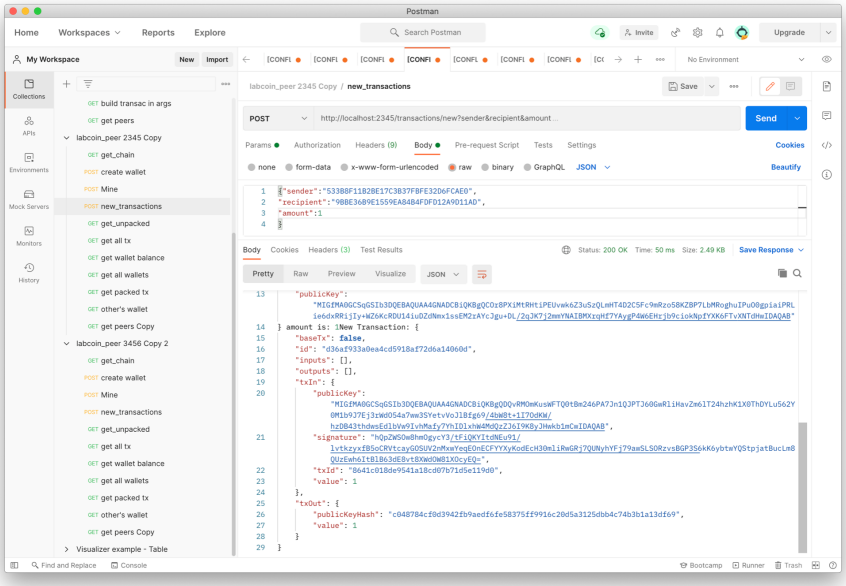
Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

C has the same UTXO as B from broadcast too. C will check whether this belongs to him if he wants to make a transaction.
GET http://localhost:3456/transactions/get/unpacked



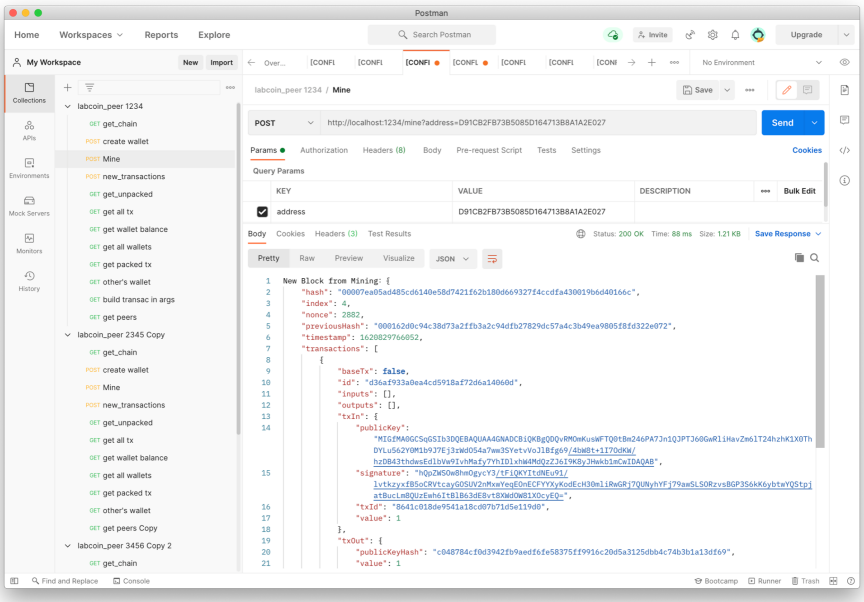
Let B create a transaction to C:
POST http://localhost:2345/transactions/new

{"sender":"533B8F11B2BE17C3B37FBFE32D6FCAE0",
"recipient":"9BBE36B9E1559EA84B4FDFD12A9D11AD",
"amount":1
}

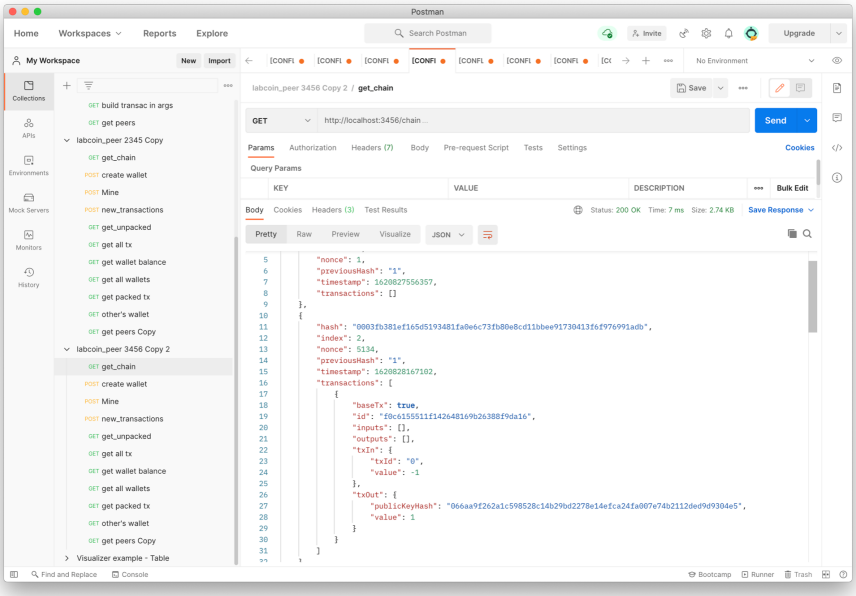


Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)

Had someone mine a block:
POST http://localhost:1234/mine?address=D91CB2FB73B5085D164713B8A1A2E027

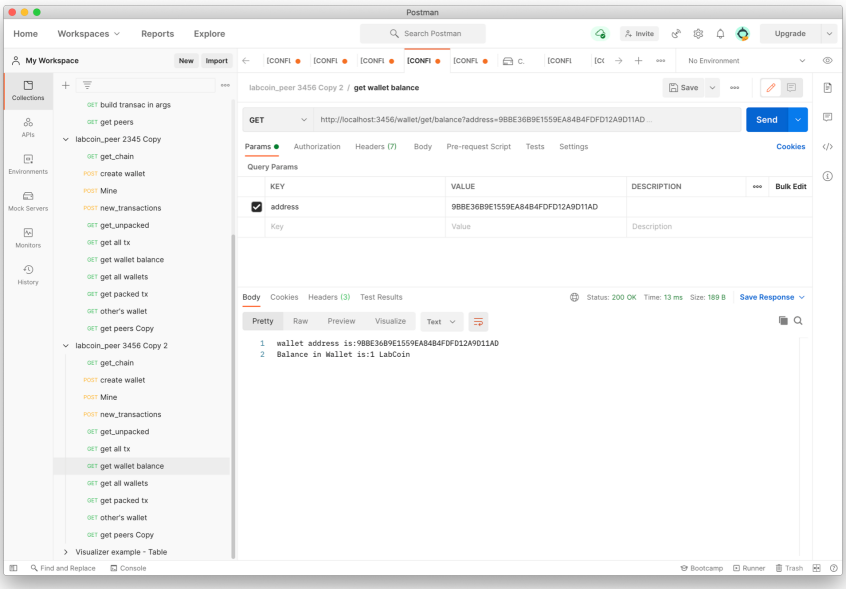


C will have the B->C transaction now
Get http://localhost:3456/chain

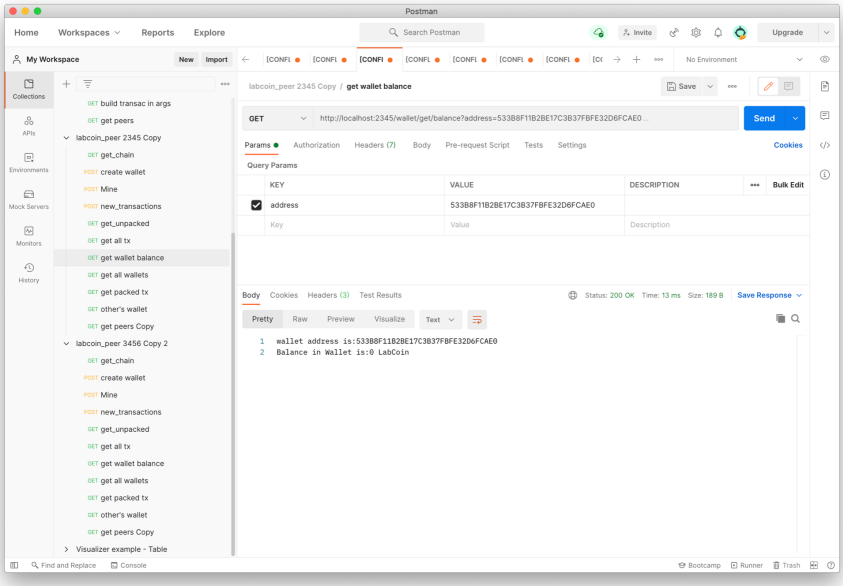


Get http://localhost:3456/transactions/get/unpacked

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)



B has 0 coin now:
GET
http://localhost:2345/wallet/get/balance?address=533B8F11B2BE17C3B37FBFE32D6FCAE0



If B now check other's wallet map, there will be 2 other wallet in script public key now.
GET http://localhost:2345/wallet/get/other

Spring 2021, Lab 4 proposal
14736 Di Lu(dilu), Yuan Gu(guy)