SUTDcoin Project Report

This project implements a simple blockchain with Account/Balance Model.

1. Mining and coin creation

   The mining and coin creation are demonstrated by running Miner.py where Miners mine keeps mine new blocks and get 100 coins as a reward for each block. The time taken to mine a new block is between 2 to 5 seconds.

```
==================Server starts======================
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
mined_blk: {"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa
Miner receives 100 coins.
Miner 957bce balance: 100
==================broadcast block starts======================
{"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214ac9
Data sent
==================Block received====================
{"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214ac9
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
mined_blk: {"header": {"hash_of_previous_header": "000006f24fcb21c5a604b981927c99efdf4ff0e
Miner receives 100 coins.
Miner 957bce balance: 200
==================broadcast block starts======================
{"header": {"hash_of_previous_header": "000006f24fcb21c5a604b981927c99efdf4ff0e61279fc538a
Data sent
---------------there is a new block-------------
new block <bound method Block.to_json of <block.Block object at 0x000002E08E0BFF60>>

hash_header_validation True
previous_header_validation True
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
```

2. Fork resolution

   Fork resolution is implemented in Miner.py and Blockchain.py. Each Blockchain object has a block dictionary and a last block which is updated whenever a new

block is added. This dictionary is later used to find the longest chain in terms of blocks.

Update the last block in blockchain in def mine_block() in Miner.py

```
self.blockchain.add(blk)
if blk.header['hash_of_previous_header'] == self.blockchain.last_blk.generate_header_hash():
    self.blockchain.last_blk = blk
else:
    if len(self.blockchain.get_longest_chain(blk)) > len(self.blockchain.get_longest_chain(self.blockchain.last_blk)):
        self.blockchain.last_blk = blk
```

Get the longest chain in Blockchain.py

```
def get_longest_chain(self, block):
    chain = [block]
    temp_block = block
    previous_header = temp_block.header.get('hash_of_previous_header')
    while previous_header != '0'*64:
        chain.insert(0,self.block_dict.get(previous_header))
        temp_block = self.block_dict.get(previous_header)
        previous_header = temp_block.header.get('hash_of_previous_header')
    return chain
```

3. Transaction resending protection

   Due to the randomness of a 32-bit nonce, it is hard to produce two identical transactions.

4. Payments between miners and SPV clients
   a) transaction validation (for miners and SPV clients)

   In transaction.py, transactions are validated by checking the types and lengths of amount, comment, nonce and keys. In Miner.py, create_transaction() and add_transaction() validate and add the transaction to the pool of transactions before the transactions are put into blocks and broadcasted.

P2P network:

Broadcasting of newly created block and transactions are implemented using multiprocessing and Sockets.

Running Miner.py and then Miner2.py simulates each miner mining new blocks, getting rewards and broadcasting new blocks to the network.

```
Successfully added block
=================Coin request received================
92577a7778b6e1a65bdb33264692f46fa5d37e0d3bdc497fe51193588acd23846ca8fae208809ec0980ad6139f65b3df 1
mined_blk: {"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214
Miner receives 100 coins.
Miner 957bce balance: 100
==================broadcast block starts=======================
{"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214ac9c9155859
Data sent
Amount 10 Balance 100
Trans_json is successfully added to pool
trans_json {"sender": "957bce29fc16ded351536e9adb7cf424fb7488fac0400ef0647164d96b756267ba2cc0dea63
Pool is not empty!
temp_txn: ['{"sender": "957bce29fc16ded351536e9adb7cf424fb7488fac0400ef0647164d96b756267ba2cc0dea6
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
=================Transaction received==================
mined_blk: {"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214
Miner sold coins.
Miner 957bce balance: 90
==================broadcast block starts=======================
{"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214ac9c9155859
Data sent
Trans_json is successfully added to pool
Pool is not empty!
temp_txn: ['{"sender": "92577a7778b6e1a65bdb33264692f46fa5d37e0d3bdc497fe51193588acd23846ca8fae208
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
mined_blk: {"header": {"hash_of_previous_header": "ecaa2bf9602228d4427978fe933bfa9082764aa5d12e214
Miner receives coins from client.
Miner 957bce balance: 110
```

Running Miner.py and then SPVclient.py simulates handling of coin purchase request from SPV client to a miner and handling of broadcasted transactions.

Note: copy the public key of Miner printed to be the receiver of the traction

After SPV client buying 10 coins from the miner, miner's balance reduces by 10.

After SPV client sending 20 coins to the miner, miner's balance increases by 20.

```
Successfully added block
==================Coin request received================
c2fdd4516a10f523b997b19e186c7e09019dea7922d1dbb233a7d4e6342ddddc6db505426ba31b2ed7
mined_blk: {"header": {"hash_of_previous_header": "58ac9a7b6e2d45766463ce42ae9beb7
Miner receives 100 coins.
Miner 957bce balance: 100
==================broadcast block starts======================
Data sent
Amount 10 Balance 100
Trans_json is successfully added to pool
trans_json {"sender": "957bce29fc16ded351536e9adb7cf424fb7488fac0400ef0647164d96b7
Pool is not empty!
temp_txn: ['{"sender": "957bce29fc16ded351536e9adb7cf424fb7488fac0400ef0647164d96b
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
=================Transaction received==================
mined_blk: {"header": {"hash_of_previous_header": "58ac9a7b6e2d45766463ce42ae9beb7
Miner sold coins.
Miner 957bce balance: 90
==================broadcast block starts======================
Data sent
Trans_json is successfully added to pool
Pool is not empty!
temp_txn: ['{"sender": "c2fdd4516a10f523b997b19e186c7e09019dea7922d1dbb233a7d4e634
POW Done
hash_header_validation True
previous_header_validation True
Successfully added block
mined_blk: {"header": {"hash_of_previous_header": "58ac9a7b6e2d45766463ce42ae9beb7
Miner receives coins from client.
Miner 957bce balance: 110
```

I am sincerely sorry that due to time constraint and limited man power, some features and the two attacks are not implemented.

5. Differences between Bitcoin and my SUTDcoin

Bitcoin uses the UTXO Model but SUTDcoin uses the Account/Balance Model. Both models achieve the same goal of keeping track of account balances in a consensus system.

The benefits of the UTXO Model are:

- Scalability—Since it is possible to process multiple UTXOs at the same time, it enables parallel transactions and encourages scalability innovation.
- Privacy—Even Bitcoin is not a completely anonymous system, but UTXO provides a higher level of privacy, as long as the users use new addresses for each transaction. If there is a need for enhanced privacy, more complex schemes, such as ring signatures, can be considered.

The benefits of the Account/Balance Model are:

- Simplicity—Ethereum opted for a more intuitive model for the benefit of developers of complex smart contracts, especially those that require state information or involve multiple parties. An example is a smart contract that keeps track of states to perform different tasks based on them. UTXO's stateless model would force transactions to include state information, and this unnecessarily complicates the design of the contracts.
- Efficiency—In addition to simplicity, the Account/Balance Model is more efficient, as each transaction only needs to validate that the sending account has enough balance to pay for the transaction.

One drawback for the Account/Balance Model is the exposure to double spending attacks. An incrementing nonce can be implemented to counteract this type of attack. In Ethereum, every account has a public viewable nonce and every time a transaction

is made, the nonce is increased by one. This can prevent the same transaction being submitted more than once.