

blockchain_database_implementation

December 3, 2017

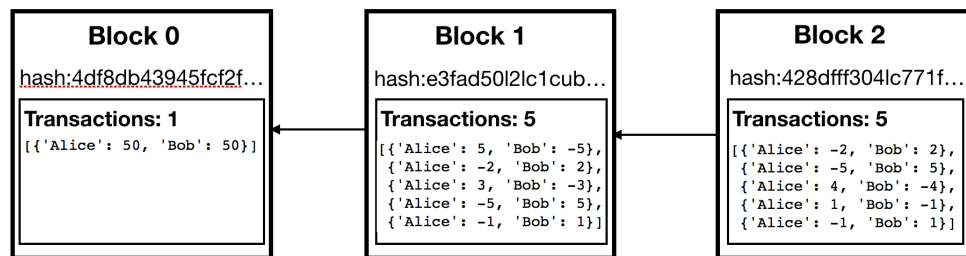
```
In [1]: import hashlib
import random
import json
```

1 Transactions, Validation, and updating system state

Blockchain is a distributed database with a set of rules for verifying new additions. We'll start off by tracking the accounts of two imaginary people: Alice and Bob, who will trade virtual money with each other.

To simulate a blockchain implementation we need: - initial state showing the balance of each account - incoming transactions - transaction validation functionality - transaction blocking functionality - block validation functionality

Blockchain



blockchain

2 Transactions

The function below generates random transactions with rules: - withdrawal have negative numbers, and deposits positive - deposits have same magnitude as the withdrawal

```
In [2]: def make_random_transaction():
amount = random.randint(1, 5)
```

```

        alice = random.choice([-amount, amount])
        bob = -alice
        return {'Alice':alice, 'Bob':bob}

[make_random_transaction() for _ in range(10)]

Out[2]: [{'Alice': -5, 'Bob': 5},
{'Alice': -5, 'Bob': 5},
{'Alice': 3, 'Bob': -3},
{'Alice': 1, 'Bob': -1},
{'Alice': -5, 'Bob': 5},
{'Alice': -5, 'Bob': 5},
{'Alice': -2, 'Bob': 2},
{'Alice': 4, 'Bob': -4},
{'Alice': 3, 'Bob': -3},
{'Alice': -4, 'Bob': 4}]

```

3 State

```

In [3]: def update_state(transaction, state):
        state = state.copy()
        for key in transaction:
            state[key] = state.get(key, 0) + transaction[key]
        return state

state = {'Alice':5, 'Bob':5}
transaction = {'Alice': -3, 'Bob': 3}
state = update_state(transaction, state)
state

Out[3]: {'Alice': 2, 'Bob': 8}

```

4 Transaction Validation

```

In [4]: def validate_transaction(transaction, state):
        if sum(transaction.values()):
            return False

        for key in transaction.keys():
            if state.get(key, 0) + transaction[key] < 0:
                return False

        return True

state = {'Alice':5, 'Bob':5}

assert validate_transaction({'Alice': -3, 'Bob': 3}, state)

```

```

assert not validate_transaction({'Alice': -4, 'Bob': 3}, state)
assert not validate_transaction({'Alice': -6, 'Bob': 6}, state)
assert validate_transaction({'Alice': -4, 'Bob': 2, 'Lisa': 2}, state)
assert not validate_transaction({'Alice': -4, 'Bob': 3, 'Lisa': 2}, state)

```

5 Blocks

Each block contains: - batch of N transactions (N - block size) - a reference to the hash of the previous block (unless block 0) - block number - a header with hash of its contents

6 Hash function

```

In [5]: def hash_function(msg=''):
        if type(msg) != str:
            msg = json.dumps(msg, sort_keys=True)
        msg = str(msg).encode('utf-8')
        return hashlib.sha256(msg).hexdigest()

        hash_function({'foo': 'bar'})

Out[5]: '426fc04f04bf8fdb5831dc37bbb6dcf70f63a37e05a68c6ea5f63e85ae579376'

```

7 Initial State

```

In [6]: state = {'Alice': 50, 'Bob': 50}
        blockchain = []

```

8 Block 0 is special

The blockchain is empty and we start by defining the block 0. - not linked to any prior block - instead of transactions it holds the initial state

```

In [7]: block0_contents = {
        'number': 0,
        'parent_hash': None,
        'transactions_count': 1,
        'transactions': [state]
        }

        block0 = {
            'hash': hash_function(block0_contents),
            'contents': block0_contents
        }

        blockchain.append(block0)
        blockchain

```

```
Out[7]: [{'contents': {'number': 0,
    'parent_hash': None,
    'transactions': [{'Alice': 50, 'Bob': 50}],
    'transactions_count': 1},
    'hash': '4df8dbf145fcf2f5ad3f122fa8f33fad4a2498a5325149c3515053fb6b844afd'}]
```

9 All other blocks are created as so:

```
In [8]: def make_block(transactions, chain):
    parent = chain[-1]

    contents = {
        'number': parent['contents']['number'] + 1,
        'parent_hash': parent['hash'],
        'transactions_count': len(transactions),
        'transactions': transactions
    }
    return {
        'hash': hash_function(contents),
        'contents': contents
    }
```

10 Random transactions for our blockchain

```
In [9]: transactions = [make_random_transaction() for _ in range(15)]
transactions
```

```
Out[9]: [{'Alice': -5, 'Bob': 5},
    {'Alice': 1, 'Bob': -1},
    {'Alice': 4, 'Bob': -4},
    {'Alice': 5, 'Bob': -5},
    {'Alice': -1, 'Bob': 1},
    {'Alice': 4, 'Bob': -4},
    {'Alice': 5, 'Bob': -5},
    {'Alice': -2, 'Bob': 2},
    {'Alice': 3, 'Bob': -3},
    {'Alice': 5, 'Bob': -5},
    {'Alice': -3, 'Bob': 3},
    {'Alice': 4, 'Bob': -4},
    {'Alice': 2, 'Bob': -2},
    {'Alice': 3, 'Bob': -3},
    {'Alice': -2, 'Bob': 2}]
```

11 Create blocks from transactions

```
In [10]: block_size = 5
```

```

transactions_buffer = []
for transaction in transactions:
    if not validate_transaction(transaction, state):
        continue

    state = update_state(transaction, state)
    transactions_buffer.append(transaction)

    if len(transactions_buffer) == block_size:
        block = make_block(transactions_buffer, blockchain)
        blockchain.append(block)
        transactions_buffer = []

state

```

Out[10]: {'Alice': 73, 'Bob': 27}

12 Blockchain output

In [11]: blockchain

```

Out[11]: [{'contents': {'number': 0,
    'parent_hash': None,
    'transactions': [{'Alice': 50, 'Bob': 50}],
    'transactions_count': 1},
    'hash': '4df8dbf145fcf2f5ad3f122fa8f33fad4a2498a5325149c3515053fb6b844afd'},
    {'contents': {'number': 1,
    'parent_hash': '4df8dbf145fcf2f5ad3f122fa8f33fad4a2498a5325149c3515053fb6b844afd',
    'transactions': [{'Alice': -5, 'Bob': 5},
        {'Alice': 1, 'Bob': -1},
        {'Alice': 4, 'Bob': -4},
        {'Alice': 5, 'Bob': -5},
        {'Alice': -1, 'Bob': 1}],
    'transactions_count': 5},
    'hash': 'db852f505b97b25ba79831ac6d733ffed9051f2af8ef79ed5b8eee5cc2ebb056'},
    {'contents': {'number': 2,
    'parent_hash': 'db852f505b97b25ba79831ac6d733ffed9051f2af8ef79ed5b8eee5cc2ebb056',
    'transactions': [{'Alice': 4, 'Bob': -4},
        {'Alice': 5, 'Bob': -5},
        {'Alice': -2, 'Bob': 2},
        {'Alice': 3, 'Bob': -3},
        {'Alice': 5, 'Bob': -5}],
    'transactions_count': 5},
    'hash': '6295074487edad26942c90ee1a4cf1a79eb71369f7570daafb9528142a402458'},
    {'contents': {'number': 3,
    'parent_hash': '6295074487edad26942c90ee1a4cf1a79eb71369f7570daafb9528142a402458',
    'transactions': [{'Alice': -3, 'Bob': 3},
        {'Alice': 4, 'Bob': -4},
        {'Alice': 2, 'Bob': -2},

```

```

    {'Alice': 3, 'Bob': -3},
    {'Alice': -2, 'Bob': 2}],
    'transactions_count': 5},
    'hash': '776c583ab2d4e04e2bafaa27deb12654f681fa65a5ad8c341f47a241bf091871'}]]

```

13 Checking Chain Validity

When we initially set up our blockchain, we have to download the full blockchain history. After downloading the chain, we would need to run through the blockchain to compute the state. To protect against somebody inserting invalid transactions in the initial chain, we need to validate of the entire chain in this initial download. Once our blockchain is synced with the network it will need to validate new blocks as they come.

In order to validate blocks we need the following checks: - block hash matches the contents - block attributes are valid - all transactions in block are valid

```

In [12]: def validate_block(block, parent, state):
    error_msg = 'Error in %d' % block['contents']['number']

    # check block hash
    assert block['hash'] == hash_function(block['contents']), error_msg

    # check block numbers
    assert block['contents']['number'] == parent['contents']['number'] + 1, error_msg

    # check parent hash
    assert block['contents']['parent_hash'] == parent['hash'], error_msg

    # check transaction count
    assert len(block['contents']['transactions']) == block['contents']['transactions_c

    # validate all transactions
    for transaction in block['contents']['transactions']:
        assert validate_transaction(transaction, state), error_msg
        state = update_state(transaction, state)

    return state

```

14 Blockchain initialization

```

In [13]: def check_chain(blockchain):
    state = {}

    for transaction in blockchain[0]['contents']['transactions']:
        state = update_state(transaction, state)

    parent = blockchain[0]

```

```
for block in blockchain[1:]:
    state = validate_block(block, parent, state)
    parent = block

return state
```

```
check_chain(blockchain)
```

```
Out[13]: {'Alice': 73, 'Bob': 27}
```

15 Conclusions

We've created all the basic architecture for a blockchain, from a set of state transition rules to a method for creating blocks, to mechanisms for checking the validity of transactions, blocks, and the full chain. We can derive the system state from a downloaded copy of the blockchain, validate new blocks that we receive from the network, and create our own blocks.

16 Extensions

We haven't explored the network architecture, the proof-of-work or proof-of-state validation step, and the consensus mechanism which provides blockchains with security against attack. We also haven't discussed public key cryptography, privacy, and verification steps.

17 References

- https://github.com/post2web/basic_blockchain
- <http://ecomunsing.com/build-your-own-blockchain>
- <http://www.androidauthority.com/what-is-a-blockchain-gary-explains-801514/>