Published in Towards Data Science

James Hope Follow

Aug 5, 2019 · 3 min read · · Listen

Build a blockchain protocol for a distributed ledger

Demystifying blockchain through a simple coding examples

mechanisms around blockchain and how they might be implemented in Python.

An quick overview of blockchain

The basic idea of a blockchain protocol is to preserve a database called a *ledger* across a distributed network of machines or *nodes.* Ledgers can be used for anything, to hold transactions or digital currency as is the case for BitCoin or Ethereum. To maintain the integrity of the ledger which is held in different places across the network, the blockchain protocol rewards participants on the network when they solve a computational intensive problem — a process referred to as *mining.* It is this process into which participants on the network enter into, that ensures the integrity of the distributed ledger on public networks. A blockchain is essentially a proof of the solutions to the problems that participants on the network solve, which are linked together in a chain, and with each new solution found, a new block is created, and into which the next set of records are copied.

Building a Blockchain Class

First, we'll build the Blockchain class constructor method that creates an initial empty list (to store our blockchain) and another to store the ledger —a record of transactions. The Blockchain class object will be responsible for managing the entire blockchain. We'll create the following methods to help us manage the blockchain:

[1] *register_node()* — this method will register a new node on the network

[2] *new_block()* — this method will create a new block in the blockchain, copy recent transactions into this new block and clear the set of transactions.

[3] valid_proof() — this method will check that a block being submitted to add to the blockchain solves the problem.

[4] proof_of_work() — this method will check that mining has yielded the correct proof to create the next block in the blockchain. We return the proof if the proof is correct, otherwise it will keep calling *valid_proof()* until we find a valid proof.


[7] *last_block()* — this method will return the last block in the chain.

Here is how the Blockchain class looks:

```
        self.current_transactions = []
        self.chain = []
        self.nodes = set()

        # Create the genesis block
        self.new_block(previous_hash='1', proof=100)

    def register_node(self, address):

        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def valid_chain(self, chain):

        last_block = chain[0]
        current_index = 1

        while current_index < len(chain):
            block = chain[current_index]
            print(f'{last_block}')
            print(f'{block}')
            print("\n-----------\n")
            # Check that the hash of the block is correct
            if block['previous_hash'] != self.hash(last_block):
                return False

            # Check that the Proof of Work is correct
            if not self.valid_proof(last_block['proof'], block['proof']):
                return False

            last_block = block
            current_index += 1

        return True


    def new_block(self, proof, previous_hash):

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
```

```
        # Reset the current list of transactions
        self.current_transactions = []

        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):

        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1

    @property
    def last_block(self):
        return self.chain[-1]

    @staticmethod
    def hash(block):

        # We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()

    def proof_of_work(self, last_proof):
        '''
        Simple Proof of Work Algorithm:
         - Find a number p' such that hash(pp') contains leading 4 zeroes, where p is the previ
         - p is the previous proof, and p' is the new proof
        '''

        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1

        return proof

```

```
94          return guess_hash[:4] == "0000"
```

Resolving conflicts on the network

Since the Blockchain protocol is run over a distributed network we'll need a method on the Blockchain class object to resolve conflicts on the network. We'll simply check if there are any other blockchains on the network that are longer than ours and if so, make that our own chain, otherwise we can continue to mine in the knowledge that we are mining for the next block and our efforts are worth it.


```
        neighbours = self.nodes
        new_chain = None

        # We're only looking for chains longer than ours
        max_length = len(self.chain)

        # Grab and verify the chains from all the nodes in our network
        for node in neighbours:
            response = requests.get(f'http://{node}/chain')

            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                # Check if the length is longer and the chain is valid
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # Replace our chain if we discovered a new, valid chain longer than ours
        if new_chain:
            self.chain = new_chain
            return True

        return False
```

blockchain.py hosted with ❤ by GitHub view raw

Interacting with the Blockchain

Finally, to bring our blockchain to life and to interact with it, we can expose a number of methods via a web app interface using flask.

Registering a node on the network

We'll want to allow users to register a node on the network so they can participate in the *proof of work* process and be rewarded for their efforts. We call the register_node() method on the Blockchain class object to register a node.

```
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

blockchain.py hosted with ❤ by **GitHub** **view raw**

We'll also want to allow these participants to check if their Blockchain is valid and fetch the Blockchain from the network if it isn't.

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

blockchain.py hosted with ❤ by **GitHub** **view raw**


added.

```python
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount']
    #if not all(k in values for k in required):
    #    return 'Missing values', 400

    # Create a new Transaction
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201
```

blockchain.py hosted with ❤ by **GitHub** **view raw**

Mining

One last important thing! We need to create the interaction to allow participants on the network to mine their blockchain.

```
        # We run the proof of work algorithm to get the next proof...
        last_block = blockchain.last_block
        last_proof = last_block['proof']
        proof = blockchain.proof_of_work(last_proof)

        # We must receive a reward for finding the proof.
        # The sender is "0" to signify that this node has mined a new coin.
        blockchain.new_transaction(
            sender="0",
            recipient=node_identifier,
            amount=1,
        )

        # Forge the new Block by adding it to the chain
        previous_hash = blockchain.hash(last_block)
        block = blockchain.new_block(proof, previous_hash)

        response = {
            'message': "New Block Forged",
            'index': block['index'],
            'transactions': block['transactions'],
            'proof': block['proof'],
            'previous_hash': block['previous_hash'],
        }
        return jsonify(response), 200
```

blockchain.py hosted with ❤ by **GitHub** **view raw**

Now we can instantiate the flask app, create the Blockchain class object and run the flask application and interact with our blockchain.


```python
# Generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', '')

# Instantiate the Blockchain
blockchain = Blockchain()

if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=5000, type=int, help='port to listen on')
    args = parser.parse_args()
    port = args.port

    app.run(host='0.0.0.0', port=port)
```

blockchain.py hosted with ❤ by GitHub view raw

To download the full code file see: https://github.com/jamesdhope/blockchain

References

[1] https://www.wikihow.com/Build-a-Blockchain-App

[2] https://www.codecademy.com/learn/introduction-to-blockchain/modules/blockchain-in-python

[3] Daniel van Flymen, https://hackernoon.com/learn-blockchains-by-building-one-117428612f46.

[4] https://towardsdatascience.com/building-a-minimal-blockchain-in-python-4f2e9934101d

[5] Omer Goldberg, https://hackernoon.com/building-a-blockchain-the-grey-paper-5be456018040