期中作业-潘昊璇

潘昊璇 121032910038

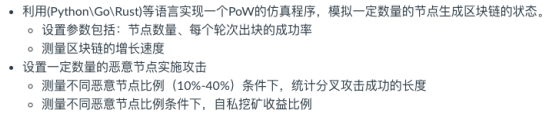
期中作业：

写一个模拟程序，模拟POW挖矿的过程，看看诚实节点、恶意节点的行为会带来什么变化。不用写成真正区块链共识的形式，一个进程模拟行为即可。

参考：

https://github.com/corgi-kx/blockchain_consensus_algorithm/tree/master/pow

https://medium.com/@vanflymen/learn-blockchains-by-building-one-117428612f46



1. 模拟程序

实际程序远远比此处所写复杂，可见本人github：https://github.com/merak0514/NIS8020_Blockchain

1.1 区块链部分

区块链结构：

```
block = {
  'index': len(self.chain) + 1,
  'timestamp': time(),
  'transactions': self.current_transactions,
  'proof': proof,
  'previous_hash': previous_hash or self.hash(self.chain[-1]),
}
```

每个节点会维护一个 `Blockchain` 类，其中包括难度系数、交易信息等等。

包含一个POW函数，具体如下：

```
def proof_of_work(self, last_proof):
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1
    print(proof)
    return proof
def valid_proof(self, prev_hash, proof):
    guess = f'{prev_hash}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()  # 十六进制的哈希。
    return guess_hash[:self.difficulty] == "0" * self.difficulty
```

其中 `prev_hash` 是上一个块的哈希。大体思路就是将上一个块的哈希（G()函数）和proof拼接起来，计算哈希2（H()函数），直到哈希2小于给定的difficulty阈值。

1.2 网络部分

采用Flask编写接口。在同一台电脑上运行，用不同的端口号来模拟节点/用户。

挖矿接口

```
@app.route('/mine', methods=['GET'])
def mine():
    q = 16  # 每次轮可以做的RO查询数

    type_ = request.args.get('type')

    last_block = blockchain.last_block
    prev_hash = blockchain.hash(last_block)
    # proof = blockchain.proof_of_work(prev_hash)
    for _ in range(q):
        # 挖一下
        success, proof = blockchain.fake_pow(prev_hash)
        if success:
            blockchain.new_transaction(
                sender="0",
                recipient=port,
                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, creator=type_)

            response = {
                'found': True,
                'message': "New Block Forged",
```

```
                'index': block['index'],
                'transactions': block['transactions'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash'],
            }
            return jsonify(response), 200
    response = {
        'found': False
    }
    return jsonify(response), 200
```

首先计算上一个区块的hash，随后开始计算nounce（`proof`）。在完成计算之后，需要给当前挖矿的矿工一个奖励。

添加neibour

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    # add neighbor nodes to current chain
    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.neighbour),
    }
    return jsonify(response), 201
```

在 `Blockchain` 类内部维护一个`neighbour`数组。

共识/同步

每个节点可以通过访问函数来和自己的邻居同步：同步方式：如果只保留合法的最长链条。

```
def resolve_conflicts(self):
    neighbours = self.neighbour
    new_chain = None
    max_length = len(self.chain)

    # Grab and verify the chains from all the nodes in our network
    for node in neighbours:
        length, chain = self.query_node(node)
        if length == -1:
            print(f"Invalidate node {node}")
            continue
        # 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

    if new_chain:
        self.chain = new_chain
        return True
    return False

@staticmethod
def query_node(node):
    response = requests.get(f'http://{node}/chain')
    if response.status_code == 200:
        js = response.json()
        return js['length'], js['chain']
  return -1, {}

def valid_chain(self, chain):
    chain_length = len(chain)
    if chain_length == 1:
        return True
    counter = 1
    p1 = chain[counter - 1]  # pointer 1
    p2 = chain[counter]
    while True:
        p1_hash = self.hash(p1)
        if p1_hash != p2['previous_hash']:  # 检查哈希
            return False
        if not self.valid_proof(p1_hash, p2['proof']):
            return False
        counter += 1
        if counter >= chain_length:
            break
        p1 = p2
        p2 = chain[counter]
    return True
```

2. 实验：基础挖矿

当然，在做实验的时候，为了让我的电脑能多撑几年，没有真的让他计算哈希，而是用了一个sleep函数+随机数替代。

2.1 出块时间测试

1. 出块时间：按照挖100块的平均时间计算。

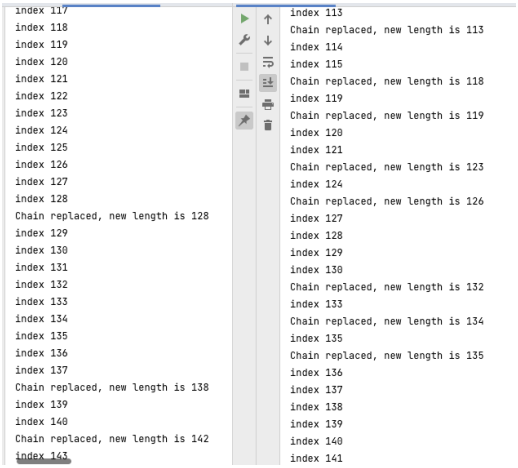
难度系数	出块平均时间/s
1	0.0022
2	0.0023
3	0.0078
4	0.1056
5	1.4350
6	25.3310

难度系数控制的是转化为16进制的哈希之后需要几个0开头。所以难度系数每增加1，理论上难度会增加16倍。观察出块所需的时间，除了在难度为 1 和 2 的时候几乎不耗时完成（可以看成所有耗时几乎由网络通讯产生），其他基本符合预期。

2.2 一起挖矿

共识同步：

采用的 tie break 策略是保留先来的。



3. 攻击

3.1 分叉攻击

假设在正常情况下，善意节点挖出的矿会给自己写入1单位的奖励；恶意节点挖出的区块会给自己写入100单位的奖励。恶意与善意的区块会进行标注。恶意节点的策略是并不拉取善意节点的list。当产生连续三个恶意区块的时候，认为分叉攻击成功。

当然，在恶意节点产生的块数超过50的时候，我们可以认为他没有机会再追上了。这时候就会重置一下（与诚实节点同步）。

在我的setting里面，因为善意节点每一轮都需要拉取所有节点上区块的信息并验证，而恶意节点不需要，所以恶意节点其实在计算上面是占优的。

1/3恶意节点实验数据：

组别	1	2	3	4	5	6	7	8	9
经过区块数	4	4	30	27	9	6	18	12	18

最后统计，平均经过14.1个区块，分叉攻击就会成功。

1/5恶意节点实验数据：

组别	1	2	3	4	5	6	7	8	9
经过区块数	129	20	49	44	43	60	116	13	26

最后统计，平均经过52.3个区块，分叉攻击就会成功。

1/10恶意节点实验数据：

组别	1	2	3	4	5	6	7	8	9
经过区块数	47	61	390	23	71	76	52	19	35

最后统计，平均经过86.8个区块，分叉攻击就会成功。

自私挖矿攻击

- 恶意矿工拥有一定比例的算力，当其成功挖出一个区块后，不公布该区块，而是试图在其后继续挖块
  - 诚实节点生成一个区块，则恶意矿工尝试用先挖优势与诚实区块竞争
  - 假设恶意节点所占算力为a，诚实节点所占算力为1-a

竞争情况	描述	收益结果	发生概率
1	恶意矿工在诚实节点出块前生成一个新块	2	a
2	恶意节点在诚实新块之前没有生成一个区块但是竞争成功	1	(1-a)/2
3	恶意节点在诚实新块之前没有生成一个区块但是竞争失败	0	(1-a)/2

- 收益的数学期望

$$2a + \frac{1-a}{2} > 1$$
$$a > 1/3$$

设定：当一个恶意节点挖出一个区块后，不公布区块，而是继续挖矿。

- 诚实节点挖出来了，恶意节点还没挖出来；恶意节点被迫和诚实节点竞争，各半的概率获胜。
- 诚实节点还没挖出来，恶意节点挖出来了；恶意节点公布前一个区块，对最新的区块继续执行恶意策略。

这样的做法和上课讲的不完全一样，实际上会略微放大了a的影响：即当a大于1/3时，恶意节点的收益更大，a小于 1/3 时，恶意节点的收益更小。

测试：挖300个区块，测试十次。

1. 恶意节点占比1/2. 测试得到的平均收益为223个区块。
2. 恶意节点占比1/3. 测试得到的平均收益为107个区块。
3. 恶意节点占比1/5. 测试得到的平均收益为48个区块。

4 总结

1. 分叉攻击：在我的代码里面，为了保证运行足够快，实际上出块的速率定在约 0.1 块每秒每个用户。在有 10 个用户的情况下，出块的速度会到达 1 秒 1 块。于是，在这么快的出块速度下，代码中的共识部分（每轮挖矿结束和其他节点同步）所消耗的时间耗时占比会很大，无法忽略不计。这种情况下，在分叉攻击中，由于恶意节点极少需要执行共识协议，所以恶意节点能够拥有更多的算力用于运算。所以恶意节点分叉攻击的成功率也会大大提高。
2. 自私攻击：我的做法和上课讲的不完全一样，实际上会略微放大了a的影响：即当a大于1/3时，恶意节点的收益更大，a小于 1/3 时，恶意节点的收益更小。由于和上述类似的原因，定量的分析不是很准确，只是能看出来，在 a 很小的时候，确实自私攻击对恶意节点来说是负收益的。

5 代码

可见本人github：https://github.com/merak0514/NIS8020_Blockchain