區塊鍊實做

40943254羅宇智
41043223黃子瑋

摘要

以現代IOT來說，如何整合各項資料尤為重要，為了方便處理資料，想到利用Block Chain來處理會是其一可行辦法，但考慮整個完成所需時間與成本過高，因此打算實作簡易交易系統來當此專題，順便精進利用python coding能力。

本次為利用區塊鏈來進行交易紀錄，模擬一個簡易虛擬貨幣系統，來瞭解資料存處流向，實現解算區塊(挖礦)、難度控制、p2p連線與利用廣播並行各節點資料同步，來了解整體運行方式。

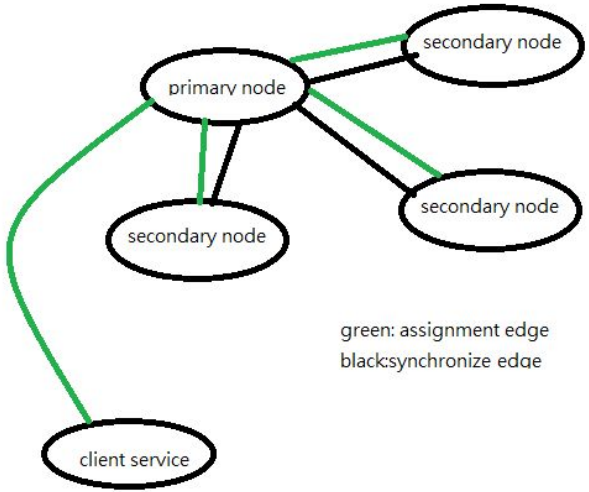
研究動機與目的

研究動機:近來處理IOT相關side project，需要對多筆資料進行梳理，若簡單存入資料庫，當多筆資料同時匯入時運行速度過慢，對終端消耗甚大，資料後續若做修正，可能因不當操作造成資料不可考證，想利用區塊鍊進行資料分散存處、分散運算、防竄改等優勢加入side project。

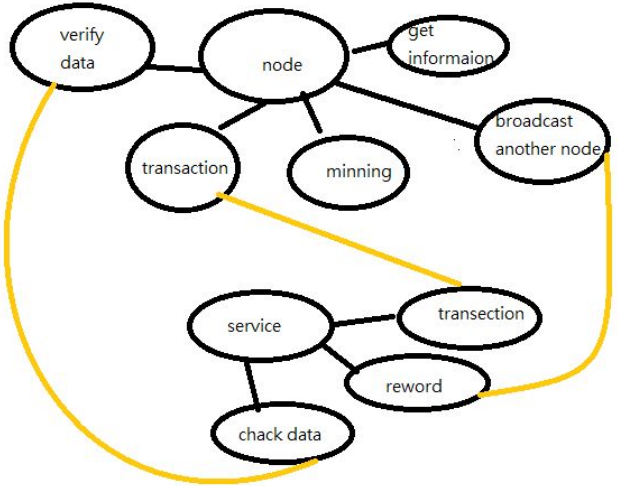
目的:利用本次專題，判斷整體效能，以及估算整理利用效益，判斷將IOT與Block Chain做結合的可能性，以及對後續發展方向進行開發

研究方法

step1:對Block Chain結構進行規劃



step2:規劃功能



研究方法與進行步驟

step3 功能結構化

```
primary node{
      secondary node{
            Transaction{
                  sender
                  receiver
                  amounts
                  fee
                  message
            }
            Block{
                  difficulty
                  time
                  miner
                  limit
                  transactions
            }
            BlockChain{
                  generate_block()
                  initialize()
                  hash()
                  min_block()
                  adjust_difficulty()
                  get_balence()
                  verify_blockchain()
                  start()
                  generate_address()
                  socket()
            }
      }
      broadcast_all_node()
}
```

```
client_service{
      Transaction{
            sender
            receiver
            amounts
            fee
            message
      }
      generate_address()
      get_address_from_public()
      extract_from_private()
      initialize_transaction()
}
```

研究方法與進行步驟

step4 撰寫程式碼

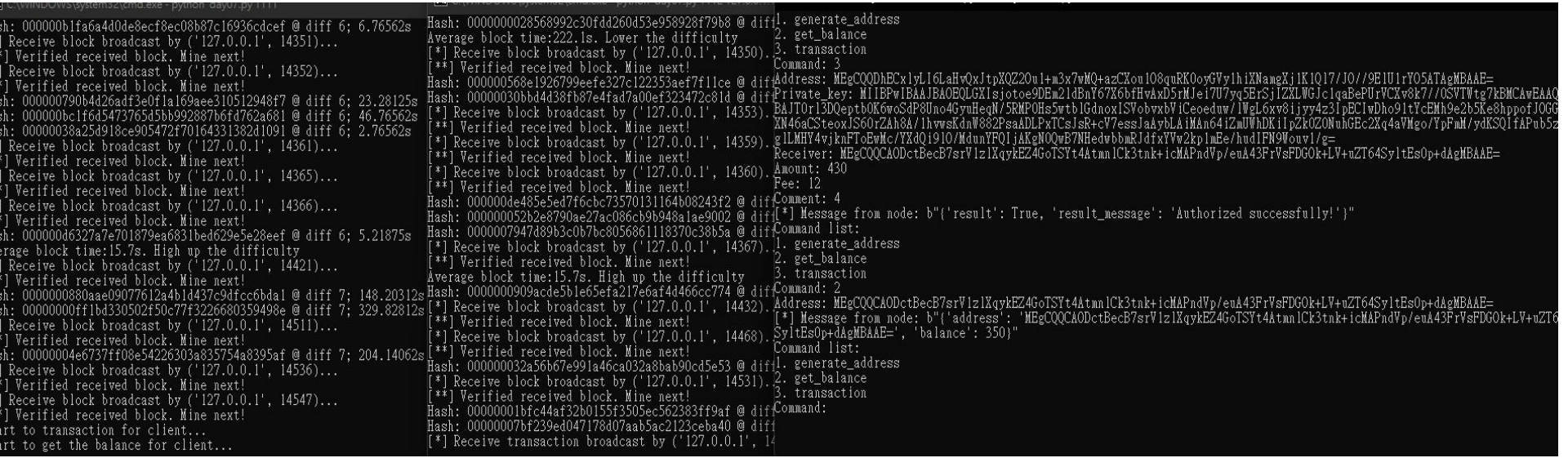
```
def clone_blockchain(self, address):
    print(f"Start to clone blockchain by {address}")
    target_host = address.split(":")[0]
    target_port = int(address.split(":")[1])
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((target_host, target_port))
    message = {"request": "clone_blockchain"}
    client.send(pickle.dumps(message))
    response = b""
    print(f"Start to receive blockchain data by {address}")
    while True:
        response += client.recv(4096)
        if len(response) % 4096:
            break
    client.close()
    response = pickle.loads(response)["blockchain_data"]

    self.adjust_difficulty_blocks = response.adjust_difficulty_blocks
    self.difficulty = response.difficulty
    self.block_time = response.block_time
    self.miner_rewards = response.miner_rewards
    self.block_limitation = response.block_limitation
    self.chain = response.chain
    self.pending_transactions = response.pending_transactions
    self.node_address.update(response.node_address)

def broadcast_block(self, new_block):
    self.broadcast_message_to_nodes("broadcast_block", new_block)

def broadcast_transaction(self, new_transaction):
    self.broadcast_message_to_nodes("broadcast_transaction", new_transaction)

def broadcast_message_to_nodes(self, request, data=None):
    address_concat = self.socket_host + ":" + str(self.socket_port)
    message = {
        "request": request,
        "data": data
    }
    for node_address in self.node_address:
        if node_address != address_concat:
            target_host = node_address.split(":")[0]
            target_port = int(node_address.split(":")[1])
            client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client.connect((target_host, target_port))
            client.sendall(pickle.dumps(message))
            client.close()
```

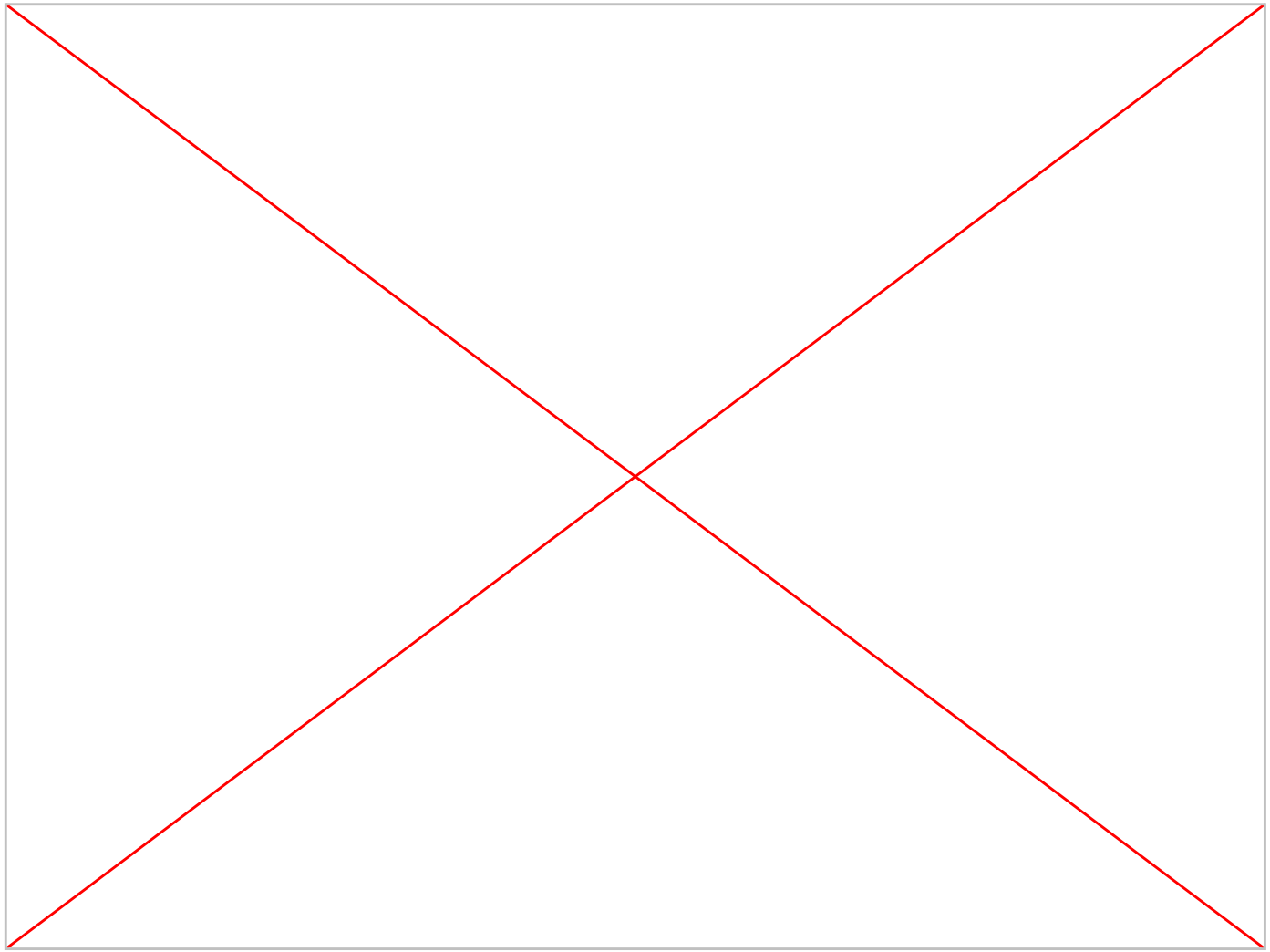
```
def verify_blockchain(self):
    previous_hash = ''
    for idx,block in enumerate(self.chain):
        if self.get_hash(block, block.nonce) != block.hash:
            print("Error:Hash not matched!")
            return False
        elif previous_hash != block.previous_hash and idx:
            print("Error:Hash not matched to previous_hash")
            return False
        previous_hash = block.hash
    print("Hash correct!")
    return True

def generate_address(self):
    public, private = rsa.newkeys(512)
    public_key = public.save_pkcs1()
    private_key = private.save_pkcs1()
    return self.get_address_from_public(public_key), \
        self.extract_from_private(private_key)

def get_address_from_public(self, public):
    address = str(public).replace('\\n','')
    address = address.replace("b'-----BEGIN RSA PUBLIC KEY-----", '')
    address = address.replace("-----END RSA PUBLIC KEY-----'", '')
    address = address.replace(' ', '')
    return address

def extract_from_private(self, private):
    private_key = str(private).replace('\\n','')
    private_key = private_key.replace("b'-----BEGIN RSA PRIVATE KEY-----", '')
    private_key = private_key.replace("-----END RSA PRIVATE KEY-----'", '')
    private_key = private_key.replace(' ', '')
    return private_key

def add_transaction(self, transaction, signature):
    public_key = '-----BEGIN RSA PUBLIC KEY-----\n'
    public_key += transaction.sender
    public_key += '\n-----END RSA PUBLIC KEY-----\n'
    public_key_pkcs = rsa.PublicKey.load_pkcs1(public_key.encode('utf-8'))
    transaction_str = self.transaction_to_string(transaction)
    if transaction.fee + transaction.amounts > self.get_balance(transaction.sender):
        return False, "Balance not enough!"
    try:
```

```
class BlockChain:
    def __init__(self):
        self.adjust_difficulty_blocks = 10
        self.difficulty = 1
        self.block_time = 30
        self.miner_rewards = 10
        self.block_limitation = 32
        self.chain = []
        self.pending_transactions = []

        # For P2P connection
        self.socket_host = "127.0.0.1"
        self.socket_port = int(sys.argv[1])
        self.node_address = {f"{self.socket_host}:{self.socket_port}"}
        self.connection_nodes = {}
        if len(sys.argv) == 3:
            self.clone_blockchain(sys.argv[2])
            print(f"Node list: {self.node_address}")
            self.broadcast_message_to_nodes("add_node", self.socket_host+":"+str(self.socket_port))
        # For broadcast block
        self.receive_verified_block = False
        self.start_socket_server()

    def create_genesis_block(self):
        print("Create genesis block...")
        new_block = Block('Hello World!', self.difficulty, '1km543', self.miner_rewards)
        new_block.hash = self.get_hash(new_block, 0)
        self.chain.append(new_block)

    def initialise_transaction(self, sender, receiver, amount, fee, message):
        # No need to check balance
        new_transaction = Transaction(sender, receiver, amount, fee, message)
        return new_transaction

    def transaction_to_string(self, transaction):
        transaction_dict = {
            'sender': str(transaction.sender),
            'receiver': str(transaction.receiver),
            'amounts': transaction.amounts,
            'fee': transaction.fee,
            'message': transaction.message
        }
        return str(transaction_dict)

    def get_transactions_string(self, block):
        transaction_str = ''
        for transaction in block.transactions:
            transaction_str += self.transaction_to_string(transaction)
        return transaction_str

    def get_hash(self, block, nonce):
        s = hashlib.sha1()
        s.update(
            (
                block.previous_hash
                + str(block.timestamp)
                + self.get_transactions_string(block)
                + str(nonce)
            ).encode("utf-8")
        )
        h = s.hexdigest()
        return h
```

研究方法與進行步驟

step5 測試





組員工作分配

羅宇智:非對稱加密設計，防護網設計，資料處裡，系統架構設計，撰寫簡報

黃子瑋:各節點網路連線設定，排程設定，系統架構設計，資料蒐集，撰寫簡報

系統特色

- 能快速加簽交易比序並具備多點簽證防偽
- 對於不明交易來源進行阻擋
- 具有防斷鍊修正，以避免整體受駭客迫害
- 已完成互斥定選以避免簽章被解碼

問題與討論

在本次專題中用來產生Hash code的rsa套件中，對加權碼長度有一定要求，而當我們實際套用IOT時，出現位數不足的狀況，還需要對雙方做妥善協定，以避面這項問題，不能直接套用也證明這套系統還不夠完善，還能繼續優化。

連線方面可能因Service異常而影響PrimaryNode運行，對於完成各層獨立執行還需要做些調整，可能整個網路架構還需做對應調整甚至大改。

本次實習心得

透過這次的小專題讓我們對於區塊鏈有更深入的了解的同時也很大程度上的增進我們在python程式編寫方面的能力，這對於我們未來在相關的技術方面的應用有很大的幫助。