Universidad Simón Bolívar Departamento de Computación y Tecnología de la Información

CI-4852 - Tópicos Especiales en Computación: Tecnologías Blockchain

Septiembre-Diciembre 2021

12-11250 Natascha Gamboa

14-10373 Henry Galue

# Implementación prototipo de Blockchain

## 1. Diseño

### 1.1. Generador de Identidades

Crear 'i' identidades:

- Crear clave privada y pública
- Asignar nombre, correo, clave privada y pública
- Asociar una dirección a la clave pública
- Crear UTXO con saldo inicial de 10.000.000 satoshis

Crear 'n' nodos:

- Crear clave privada y pública
- Asignar nombre, clave privada y pública
- Asociar una dirección a la clave pública

### 1.2. Generador de Transacciones

Mientras no se interrumpa la ejecución:

- Seleccionar remitente aleatorio
- Seleccionar destinatario aleatorio
- Seleccionar X satoshis para enviar
- Para cada UTXO del remitente
    a. Para cada salida
        - Seleccionar las no gastadas
        - Crear y Firmar**(*)** entrada
        - Agregar a las entradas de la nueva transacción
- Calcular vuelto de ser necesario
- Crear y firmar**(*)** salida para el destinatario
- Agregar nueva transacción a las UTXO del destinatario
- Enviar nueva transacción a un nodo aleatorio de la red

### 1.3. Nodo

- Crear e iniciar hilo que se encargará de comunicarse con la red
- Crear e iniciar hilo que se encargará de minar

Hilo de red

- Presentarse a la red
- Se mantiene escuchando mensajes
    - Recibe mensaje
    - Procesa mensaje

- Si recibe una transacción
  - Si es una transacción ya propagada por el nodo, esperar siguiente mensaje
  - Validar
    - Evaluar los scripts P2SH **(*)**
    - Tiene entradas y salidas
    - sum(entradas) > sum(salidas)
    - Cada entrada corresponde a una salida no gastada
  - Si es válida, agregar al mempool y propagar
- Si recibe un bloque
  - Si es un bloque ya propagado por el nodo, esperar siguiente mensaje
  - Validar
    - Hashes transacciones dan el root merkle tree
    - Hash bloque anterior concuerda con el de la cadena
    - Verificar la prueba de trabajo (prefijo '00')
  - Si es válido
    - Abortar minador
    - Recalcular mempool eliminando las transacciones presentes en el bloque ganador
    - Encadenar
    - Propagar
  - Si el minador anunció un bloque candidato, propagarlo

Hilo minador

- Mientras no haya que abortar
  - Tomar una transacción del mempool y agregarla al bloque candidato
  - Si el tamaño del bloque es igual al máximo establecido
    - Calcular root del árbol de merkle
    - Realizar prueba de trabajo
    - Anunciar bloque candidato al hilo de red
- Si hay que abortar, se regresan las transacciones al mempool y se vuelve a comenzar a minar otro bloque candidato

### 1.4. Explorador

- Transacciones
  - Busca en las transacciones de cada bloque de la cadena, si el hash de la transacción corresponde al de la solicitada
- Bloques
  - Por hash
    - Busca en la cadena si el hash del bloque corresponde a alguno de los bloques
  - Por altura
    - Busca si la altura contiene un bloque

### 1.5. Visualizador de Logs

En nuestro diseño inicial se tenía previsto implementar el visualizador de la siguiente manera:

Inicializar con el comando: visualizador -d <dirlogfile> { −mt | −mg).

En donde ‘dirlogfile’ es el directorio donde están almacenados los archivos .log de los nodos, y -mt y -mg representan la presentación deseada.

Para el modo secuencial:

- Leer todos los logs
- Ordenar los mensajes por timestamp
- Mostrar en pantalla

Para el modo paralelo:

- Leer todos los logs en listas diferentes
- Asignar cada lista a una casilla en donde se mostrará la información
- Configurar la velocidad de salida de la información
- Combobox que permita cambiar el nodo que está asociado a la casilla
- Capacidad de pausar visualización y de ejecutar por timeframe
- Guardar en una variable las acciones y en otra las operaciones que se realizan en transacciones/bloques
- Filtros
    - Por rango de timestamp
    - Por operación que se realizan en transacciones/bloques
    - Por acción

Estructura de los logs:

- [timestamp]: acción ejecutándose/ejecutada: posible data adicional...

  El ‘timestamp’ se representa en forma asc en inglés [Dia Mes Fecha Hora Año]

  Acciones:

    - Starting to generate transactions
    - Generating transaction: <#>
    - Sending: <amount> from <address> to <address>
    - Checking utxo: <hash transacción>
    - Checking output: <forma diccionario de la salida>
    - Input created: <forma diccionario de la entrada>
    - Transaction created
    - Change created
    - Sending transaction: <hash>
    - Connected: <puerto>
    - Request send: <nombre nodo>
    - Waiting ack
    - Got ack: <mensaje recibido>
    - Starting Node: <nombre nodo>
    - Got request: <petición (‘Presentacion’, Transaccion Nueva’, ‘Transaccion’, ‘Bloque’)>
    - Transaction received: <hash>
    - Transaction accepted: <hash>
    - Adding transaction to mempool: <hash>
    - Transaction added to mempool: <hash>
    - Transaction rejected: <hash>
    - Block received: <hash>
    - Block accepted: <hash>
    - Aborting mining
    - Block rejected: <hash>
    - Block mined: <hash>
    - Validating Transaction: <hash>

- Validating Block: <hash>
- Adding block to blockchain: <hash>
- Spreading
- Sending to <(nombre,puerto)>: <Transacción o bloque en forma diccionario>
- Updating mempool: <mempool (lista)>
- Mempool updated: <mempool (lista)>
- Mining started
- Checking transactions in mempool
- Adding transaction to new block: <hash>
- Adding transaction to merkle tree: <hash>
- Executing Proof of work
- Restarting mining
- Returning transactions to mempool

# 2. Implementaciones faltantes

Debido al haber comenzado tarde con el diseño e implementación del proyecto, las siguientes funcionalidades no se implementaron:

- Considerar el número máx/min de inputs/outputs que puede tener una transacción
- Firmar unidades de valor al crear transacciones
- Encriptar transacción
- Presentación de cada nodo a la red
- Encriptar mensajes enviados por los nodos
- Manejo de los forks
- Manejo de los bloques huerfanos
- Verificación de los scripts P2SH
- Visualizador paralelo de archivos .log

# 3. Consideraciones

Se realizaron las siguientes implementaciones para una implementación y testeo más rápida y eficiente:

- Los mensajes enviados por la red tienen la siguiente estructura:
    - (<mensaje>, forma diccionario de la transacción o bloque)
- Los nodos hacen referencia a una blockchain compartida, es decir hacen referencia a un mismo objeto
- Las conexiones entre nodos se crean de forma aleatoria con una función 'genNetwork', no se leen de un archivo.
- Se tiene un archivo de configuración 'config/variables.py' que contiene las variables:
    - FREQUENCY
    - MIN_INPUTS
    - MAX_INPUTS
    - MIN_OUTPUTS
    - MAX_OUTPUTS
    - BLOCK_MAX_SIZE
    - T_BLOCK_CREATION
    - INIT_DIFFICULTY
- Las funciones para explorar la blockchain se encuentran en la clase de la misma, por lo que para poder usarlas se debe usar la variable que contiene la 'Blockchain'. Imprimen en pantalla.
    - blockExplorer(self, flag: str, value: str) -> None
    - txExplorer(self, flag: str, hash: str) -> None

- El visualizador secuencial de archivos .log se implementó como una función que imprime por consola solamente

# 4. Fallas

- Pueden ocurrir forks
- En algunas ocasiones al escribir en el archivo .log no se escribe correctamente, causando que no se pueda leer correctamente

# 5. Ejecución y prueba

Se ofrecen 2 opciones para ejecutar el prototipo y probarlo:

## 5.1. Ejecución con archivo principal

En la carpeta principal del proyecto se proporciona un archivo de nombre 'blockchainSimulator.py', el cual contiene una función para ejecutar el proyecto completo con parámetros predeterminados que pueden ser suministrados a la función.

Para terminar esta ejecución se debe cerrar la terminal en donde se inició o usar '&' antes de ejecutar

## 5.2. Ejecución por separado

Se debe ejecutar el generador de identidades y proporcionar los nodos al generador de la red, luego crear una variable de tipo 'Blockchain' para luego iniciar cada nodo proporcionando el nombre, los nodos(generados), la red y la blockchain creadas. Finalmente se inicia el generador de transacciones.

Se pueden iniciar los nodos en el mismo terminal o en varios aparte

Se puede cambiar el directorio por defecto en el cual se guardan los logs, el cual es 'output/logs'.

## 5.3. Firmas de las funciones a usar

- blockchainSimulator(i: int, n: int, m: int, outDir: str) -> None
- genIdenti(identities: int, nodos: int) -> Tuple[list[User], dict[str,Node]]
- genNetwork(nodes: dict[str,Node], pairs: int) -> list[Tuple[str,int]]
- genTransac(users: list[User], nodes: dict[str,Node], logDir: str) -> None
- nodo(name: str, nodes: dict[str,Node], net: list[Tuple[str,int]], logDir: str, bc: Blockchain) -> None