TP Integrador Bitcoin Fase II. Networking.

ALGORITMOS Y ESTRUCTURA DE DATOS

2020 - 1° cuatrimestre

Requerimiento

El programa le deberá permitir al usuario:

- Crear nodos (ver sección Características de los nodos).
- Crear conexiones entre dos nodos ya creados en esta instancia del programa, y entre un nodo creado en esta instancia del programa y otro nodos creado en otra computadora.
- Enviar cualquiera de los 6 tipos de mensajes (ver sección Mensajes) entre cualquier par de nodos de la red que previamente hayan sido conectados por el usuario.

Las comunicaciones deben realizarse de manera asincrónica.

Funcionamiento típico del programa

Inicialmente el usuario crea cada uno de los nodos (un estado inicial de la GUI maneja la creación de un nodo a la vez), asignando el puerto que le corresponde. Luego, el usuario especifica entre qué pares de nodos hay una conexión. No es necesario hacer una representación gráfica de cada nodo

para que el usuario seleccione, sino que alcanza con identificarlos mediante IP y puerto a través de la GUI. Por último, el usuario indica a través de la GUI que se manden mensajes de un nodo a otro nodo vecino, especificando los detalles propios de cada mensajes según lo indicado en la sección Mensajes.

Características de los nodos

- Pueden ser de tipo Full o SPV. En esta fase, la diferencia principal está en que los mensajes que pueden enviar y recibir dependen de su tipo (ver sección Mensajes posibles para cada tipo de nodo).[1]
- Cada nodo puede mandar y recibir mensajes, por lo que debe ser tanto servidor como cliente HTTP. El servidor debe usar un puerto par, y el cliente debe usar el puerto siguiente. Por ejemplo, el servidor escucha el puerto 8330, y el cliente se comunica por el puerto 8331.
- Tienen guardada la dirección IP y el puerto del servidor[2] de sus vecinos.
- Están constantemente escuchando el puerto propio asignado a su servidor.
- No puede haber una conexión entre dos nodos SPV. Sí puede haber conexión entre dos nodos tipo Full y entre un nodo tipo Full y uno tipo SPV

Observación: la descripción de los nodos se ampliará en las siguientes fases.

Comunicación

Modo cliente

Para **enviar un mensaje** desde un nodo se deben seguir los siguientes pasos:

- El usuario elige por GUI el nodo vecino al que quiere enviarle un mensaje, el tipo de mensaje que quiere enviarle, y de ser necesario, información extra de contenido.
- El nodo se conecta al *socket* del nodo vecino (en este caso, el nodo actúa como cliente y el nodo vecino como servidor).
- El nodo envía al nodo vecino un request de tipo *GET* o *POST* según corresponda por el mensaje.
- Cuando recibe la respuesta, se cierra la conexión.

Modo servidor

Para **recibir un mensaje**:

- El nodo detecta un request de alguno de sus vecinos para cada mensaje (ver sección Mensajes) y empieza la comunicación (en este caso el nodo actúa como servidor y el nodo vecino como cliente).
- El servidor debe seguir escuchando el puerto y ser capaz de recibir mensajes en simultáneo.

---

[1] Aparte de los mensajes que pueden enviar y/o recibir, existen otras diferencias que se incluirán como requerimientos en las próximas fases.
[2] De nuevo, debe ser un número par.

- El servidor debe parsear el mensaje y responder acordemente según cada mensaje (ver respuestas en la sección *mensajes*).
- Se cierra la conexión con el cliente. El servidor luego se debe quedar escuchando el puerto para poder recibir más mensajes.

Mensajes

Existen 6 tipos de mensajes: *block*, *transaction*, *merkleblock*, *filter*, *get blocks* y *get block header*. En esta fase, el usuario controla el momento de envío de todos los mensajes, pero solo controla el contenido del mensaje de transacción.[3]

- Si desea mandar un mensaje transacción, debe indicar el monto que quiere transferir y la *public key* de la wallet a la que quiere transferir. Con esta información, el nodo debe armar una transacción:
    - El arreglo vout debe tener un único elemento en donde se guarde la información ingresada por el usuario.
    - El arreglo vin puede estar vacío, o puede tener elementos que cumplan con el formato JSON correspondiente. En el segundo caso, siempre que se cumpla el formato correcto, el contenido pueden ser valores *dummy*, o *placeholders*.
- Si desea mandar un mensaje que no sea transacción, debe mandar cualquier contenido que sea válido.

A continuación se detalla cómo se deben armar los mensajes para enviar por HTTP. En todos los casos el Host es la dirección IP del nodo al que se quiere enviar el mensaje.

Block (POST):

URL: Host/eda_coin/send_block/
Campo de datos: JSON del Bloque (igual que fase I)
Respuesta: JSON de Response (ver sección Nuevos JSONs).

Transaction (POST):

URL: Host/eda_coin/send_tx/
Campo de datos: JSON de la TX (igual que fase I)
Respuesta: JSON de Response (ver sección Nuevos JSONs).

MerkleBlock (POST):

URL: Host/eda_coin/send_merkle_block/
Campo de datos: JSON del Merkle Block (ver sección Nuevos JSONs)
Respuesta: JSON de Response (ver sección Nuevos JSONs).

---

[3] Una vez terminadas todas las fases, el usuario solamente tiene control sobre el contenido y el momento de envío de los mensajes transacción. Los demás mensajes son armados y enviados automáticamente por los nodos.

Filter (POST):

URL: Host/eda_coin/send_filter/
Campo de datos: JSON del Filter (ver sección Nuevos JSONs)
Respuesta: JSON de Response (ver sección Nuevos JSONs).

Get blocks (GET):

URL: Host/eda_coin/get_blocks?block_id=number&count=number
Parámetros:

- block_id: ID del primer bloque a recibir en la lista. Si se quiere recibir desde el principio se tiene almacenado ningún header, se envía block_id = 0x00000000
- count: número máximo de bloques a recibir (podrían recibirse menos).

Respuesta: JSON Response, donde el "result" es un arreglo de los bloques de los count bloques siguientes[4] al bloque de block_id.

Get block header (GET):

URL: Host/eda_coin/get_block_header?block_id=number&count=number
Parámetros:

- block_id: ID del último bloque cuyo header se tiene almacenado. Si todavía no se tiene almacenado ningún header, se envía block_id = 0x00000000
- count: número máximo de headers a recibir(podrían no existir tantos todavía, el nodo que solicita los headers no sabe)

Respuesta: JSON Response, donde el "result" es un arreglo de los headers de los count bloques siguientes[5] al bloque de block_id. La representación en JSON de un header es igual a la de un bloque pero sin el vector de transacciones. El arreglo está ordenado de forma secuencial: el primer header es el más antiguo, y el último es el más reciente. Si la cantidad de bloques después del bloque de block_id es n<count se mandan *n* headers. El nodo que hizo el GET entiende que si recibió menos de count headers, el último header de la lista corresponde con el bloque más recientemente agregado a la blockchain.

Nuevos JSONs

JSON de Response

```
{
      "status": bool,
      "result": object or number or null
}
```

"status" es true si no hubo error, false si hubo error.
"result" toma los siguientes valores:

---

[4] Siguientes en este contexto hace referencia a los bloques que vienen inmediatamente después (más recientes) en la blockchain.
[5] Siguientes en este contexto hace referencia a los bloques que vienen inmediatamente después (más recientes) en la blockchain.

- Si status == true
    - result es un objeto correspondiente a cada petición POST o GET, si está vacío es “null”
- Si status == false
    - result = 1 si hubo error de formato
    - result = 2 si hubo error de contenido

JSON de un MerkleBlock

```
{
      “blockid”: string,
      “tx” : {
            … JSON de una TX igual que en fase I ...
      },
      “txPos”: number,
      “merklePath”: [
            “Id”: “1234”,
            “Id”: “4321”,
            “Id”: “2314”
      ],
}
```

JSON de un Filter

```
{
      “Key”: “pubkey1”
}
```

Mensajes posibles para cada tipo de nodo

E: este tipo de nodo puede enviar este tipo de mensajes
R: este tipo de nodo puede recibir este tipo de mensajes

	Block	**Transaction**	**Merkleblock**	**Filter**	**Get block headers**	**Get blocks**
Nodo full	E / R	E / R	E	R	R	R / E
Nodo SPV	-	E	R	E	E	-

Interfaz gráfica

En esta fase, a la GUI se le agregan 5 funciones principales:

1. Permitir crear nodos. Para esto, el usuario ingresa:
    - La IP y el puerto del servidor del nuevo nodo.
    - El tipo del nuevo nodo (Full o SPV).
2. Permitir crear conexiones entre nodos. Para esto, el usuario ingresa:
    - Un par de nodos que puedan ser vecinos (Full-Full o Full-SPV, pero no SPV-SPV)
3. Representar gráficamente los nodos de la red que fueron creados desde el mismo programa.
4. Permitir enviar mensajes entre dos nodos. Para esto, el usuario ingresa:
    - El nodo que envía el mensaje.
    - El nodo vecino que recibe el mensaje (la GUI no debe permitir que el usuario seleccione un nodo que no es vecino del seleccionado en el paso anterior).
    - El mensaje que se quiere enviar (la GUI no debe permitir que el usuario seleccione un mensaje no permitido teniendo en cuenta el tipo de nodo que envía y el tipo de nodo que recibe).
        - Si el mensaje que se quiere enviar es TX, el usuario debe poder ingresar la cantidad de EDACoins que quieren transferirse y la *public key* a la cual quiere transferirse.
5. Mostrar el estado de las conexiones entre nodos:
    - Avisar cuando un nodo envía un mensaje
    - Avisar cuando un nodo recibe un mensaje y de quien lo recibió (el nodo que recibe debe saber quién se lo mandó)
    - Avisar cuando se establece una conexión entre nodos y mostrar los distintos estados que ésta conexión transita (los estados se ven reflejados en los pasos que se dan cuando el nodo funciona en modo Cliente/Servidor)

Cotización EDACoin al 23/5/2020:
1 EDACoin = 3 alfajores Guaymallén Triverde = 1.2 jorgelines blancos