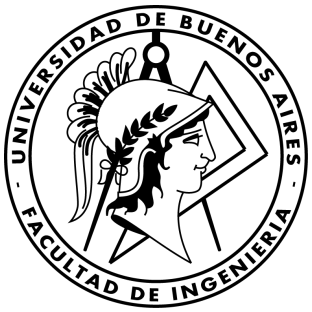

Trabajo Práctico Bitcoin — Informe
Examen Final

Taller de Programacion I
Primer cuatrimestre de 2023

Alumno	Número de padrón	Email
Masri, Noah	108814	noahmasri19@gmail.com
Ayala, Camila	107440	cayala@fi.uba.ar
Pol, Juan Manuel	108448	jpol@fi.uba.ar
Cominotti, Claudia	104891	ccominotti@fi.uba.ar

Índice

1. Introducción

Para la presente entrega fue pedida una extension del trabajo presentado anteriormente, donde ahora no solamente el nodo principal funciona como servidor para billeteras, sino que ahora tambien lo hace para otros nodos. Para que esto sea posible se debieron llevar a cabo diversas modificaciones en las estructuras con las que contabamos previamente para poder implementar las funcionalidades pedidas, manteniendo la complejidad de las operaciones lo mas bajas posibles.

2. Cambios al protocolo previo

Para esta entrega, se realizaron modificaciones a distintas partes del codigo de la entrega anterior, con el fin de mejorar ciertas cosas que habian quedado pendientes, o bien de ayudar a la implementacion de las funcionalidades requeridas.

2.1. Cambios en la interfaz grafica

En la entrega previa notamos que podria ser util tener como dato la direccion de la billetera que esta siendo utilizada, por lo que pusimos a la vista esta información, asi el usuario sabe donde pedir sus transacciones. Por otro lado, encontramos que seria practico y amigable para el usuario que se pueda copiar tanto esta información como los hashes de las transacciones, ya que son datos que se utilizan para otras funcionalidades, como hacer más fácil el pedido de proof of inclusion de estas o mismo transacciones entre distintas billeteras.

2.2. Cambios en Block Getter

Previamente contábamos con un recv timeout, que frenaba la descarga de bloques una vez que pasaba 20 segundos sin recibir nada. Esto hacía que fuese menos eficiente la descarga, ya que tardaba más de lo que debía, y a su vez era muy falible; si por algún motivo los peers estaban inactivos por unos momentos, abortaba la descarga. Se decidió cambiar la estructura en la que el Block Getter recibe los headers de los bloques que debe pedir, pasando de un vector de headers a un HashSet de los hashes de los headers. El orquestador envía todos los headers que contiene, y luego se guarda este HashSet como registro, en el que cada vez que recibe un bloque, lo borra. Una vez que este HashSet está vacío, sale del loop en el que ahora realiza un recv sin timeout, y finaliza la descarga de bloques. Esto no solo mejoro la eficiencia sino que tambien su complejidad espacial, ya que no era necesario clonar todos los headers para mandarlos, sino que su hash era suficiente.

2.3. Cambios en las estructuras de la Blockchain

En la entrega anterior nos bastaba con vectores para guardar tanto los headers como los bloques, pero ante la necesidad de enviarlos a un nodo cliente decidimos guardarlos de manera que podamos ubicarlos mediante su hash, que es lo que nos envían en los mensajes get headers y get data. Los bloques los guardamos en un hashmap donde la clave es el hash del encabezado del bloque. Con el propósito de contar con un orden en el caso de los headers, usamos dos estructuras para guardar la información. Por un lado tenemos un vector de headers, el cual está ordenado cronológicamente. Para facilitar la búsqueda agregamos un hashmap que consiste de pares de clave hash del header y valor el índice correspondiente a ese header en el vector ordenado. La inserción en la descarga inicial no generó ningún tipo de inconveniencia pero la llegada de bloques/headers posteriormente fue necesario analizar más cuidadosamente. Por último, antes contábamos con el vector de bloques ordenados, pero ahora ya no, por lo que en caso de querer recorrer los bloques de manera ordenada, se debe hacer a través del vector ordenado de encabezados, como es el caso de la creación de las UTXOs.

2.4. Adicion de persistencia de bloques

Nuestro programa ya contaba con persistencia de headers, pero dado que los bloques son más pesados, estos no eran guardados ya que hasta el momento no nos había sido necesario. Lo que hicimos fue ahora tener una carpeta con un nombre a elección del usuario en que se guardan archivos que contienen hasta 1000 bloques, con un nombre generado mediante una combinación totalmente aleatoria de 15 caracteres alfanuméricos. El hecho de que el nombre de los archivos sea de esta manera se debe a que si poniamos nombres enumerativos, entonces debíamos guardar el nombre que seguía por ejemplo, pero de esta manera era más fácil la creación de los archivos. Decidimos que el tope de bloques en un archivo sea de 1000 en vez de que se encuentren todos en el mismo archivo, ya que si descargasemos todos los bloques de la blockchain dicho archivo sería excesivamente pesado.

2.5. Cambios en el archivo de configuración

Se realizaron varias modificaciones a los campos recibidos por el archivo de configuración que resultaban necesarias para esta entrega. En primer lugar, se cambió el campo “dns domain” en que llegaba un dominio al que se le hacía un lookup, a contar con un struct peers, que ahora permite que llegue tanto una dirección dns como también direcciones IP de los nodos a los que se quiere conectar. En caso de que no se reciba nada, se toma por sentado que hará una conexión default a los nodos de la testnet. Luego, se cambio el campo “blockchain”, que era el path a un archivo binario de headers, para ahora tener dos campo “headers file”, que tiene lo dicho anteriormente,

y un nuevo campo “blocks dir” que contiene el path a un directorio en donde se encuentran o se encontraran los bloques.

2.6. Cambios al Network Listener

Anteriormente estaba determinado a que peers iba a escuchar el listener desde su creación y era imposible agregar uno, pero al agregar la funcionalidad de server necesitábamos que puedan agregarse peers a la estructura posterior a la creación, para que estos sean tratados como cualquier otro. Para poder crear un nuevo listener para las conexiones entrantes, es necesario que el orquestador cuente con el sender del canal de comunicación entre listeners y orquestador, para poder otorgarle este al listener que escuchara del stream del peer entrante para su creación, por lo que nos guardamos en la estructura del network listener ambos extremos de estos canales; uno para poder escucharlo, otro para poder asignarlo al nuevo listener.

3. Nodo como servidor

En la siguiente seccion se detallaran las tareas llevadas a cabo con el fin de cumplir con las funcionalidades requeridas.

3.1. Comportamiento de nodo cliente

Para la demostración de la funcionalidad de servidor de nuestro nodo creamos un segundo nodo que realiza la descarga inicial de la blockchain conectandose exclusivamente a nuestro nodo original. Cuenta también con el network listener, por lo que puede responder cualquier pedido que le llegue, tanto de bloques como de encabezados, inv messages, etc, pero no recibe conexiones entrantes, por lo que al estar conectado unicamente a nuestro nodo servidor, nadie le enviara esto. Se queda tambien escuchando este mismo listener para el anuncio de la llegada de nuevos bloques. Este nodo tambien guarda sus bloques y sus encabezados en memoria, pero para la demostracion se comienza con ambas cosas vacias.

3.2. Archivos de configuracion para los distintos nodos

Ahora que se tiene un nodo que funciona como cliente y otro que funciona como servidor, para la presentacion se crearon dos archivos de configuracion distintos, con la información necesaria para cada uno en el formato establecido, así al momento de correr cada proceso del programa se carga con sus respectivos datos de inicio. Al iniciar el programa con el nodo servidor usando su archivo de configuracion, este se conectara a la testnet, mientras que el cliente se conectara unica

y exclusivamente a nuestro nodo servidor. Cada nodo tiene su propio logfile y sus propios archivos en donde persisten los datos.

3.3. Barra de progreso de la descarga de bloques

La interfaz que realizamos con gtk y Glade no es del nodo, sino que es de la billetera que dentro de nuestro protocolo, no forman parte de lo mismo. Es por esto que no tenia sentido realizar una barra de progreso de carga de bloques en dicha interfaz. Para cumplir con este punto solicitado en el enunciado de la entrega final, se realizo una barra de progreso impresa por pantalla en la terminal del nodo servidor. Utilizamos una biblioteca llamada indicatif, que te permite ir observando la descarga de los bloques, de forma grafica por medio de dicha barra, e indica un tiempo estimado de demora hasta la finalizacion de la tarea. Cuando esta finaliza, informa download completed en la misma consola, al lado de la barra.