

PROYECTO BITCOIN

**Taller de programación I
FIUBA
1° cuatrimestre 2023**



Proyecto realizado por:

- Martin Scazzola
- Valentín Santander
- Azul Fuentes
- Gastón Frenkel

1. Introducción

El presente informe tiene como objetivo detallar la implementación de un nodo de Bitcoin con funcionalidades acotadas y una wallet con interfaz gráfica la cual nos permite interactuar con el mismo. Las funcionalidades que implementa el nodo son las siguientes:

Nodo

- Conexión a red a través de una red peer to peer.
- Descarga completa de los headers de la blockchain.
- Descarga de bloques a partir de una fecha dada.
- Validación de bloques y transacciones.
- Merkle proof of inclusion
- Actualización en tiempo real de la blockchain.
- Actualización del set UTXO para poder realizar transacciones.
- Broadcasting de nuevas transacciones.

Wallet

- Interfaz Gráfica basada en bitcoin-qt.
- Creación de transacciones.
- Manejo de múltiples cuentas.
- Visualización del balance de cuenta.
- Actualización de las transacciones vinculadas a la cuenta.
- Admitir pedido de prueba de inclusión.

2. Conexión a la red P2P

2.1. Mensajes

A partir del comando “dig” a las direcciones de DNS otorgadas por la cátedra se obtuvieron las IPs de los nodos pertenecientes a la red de Testnet de Bitcoin. Para la conexión y comunicación con estas IPs se crearon distintas estructuras del tipo Mensaje, basados en el protocolo de la red:

- GetData: mensaje mediante el cual se le pide una transacción o un bloque a otro nodo.
- GetHeaders: mensaje mediante el cual se le solicita headers a otro nodo.
- Header: es la estructura del header que poseen todos los mensajes, este posee información acerca del mensaje asociado. Algunos mensajes como el “Verack” solo están compuestos por un Header.
- Headers: mensaje en respuesta a “GetHeaders” con los headers solicitados.
- Inv: mensaje mediante el cual los peers a los que el Nodo se encuentra conectado lo actualizan con nuevos bloques y transacciones.
- Ping: mensaje mediante el cual los peers comprueban la conexión con otro nodo enviando un número aleatorio.
- Pong: mensaje respuesta al Ping con el mismo número que se recibió.
- Version: mensaje mediante el cual se comunica a otro nodo sus datos de configuración.
- Tx: mensaje mediante el cual se envía una transacción a otro nodo.

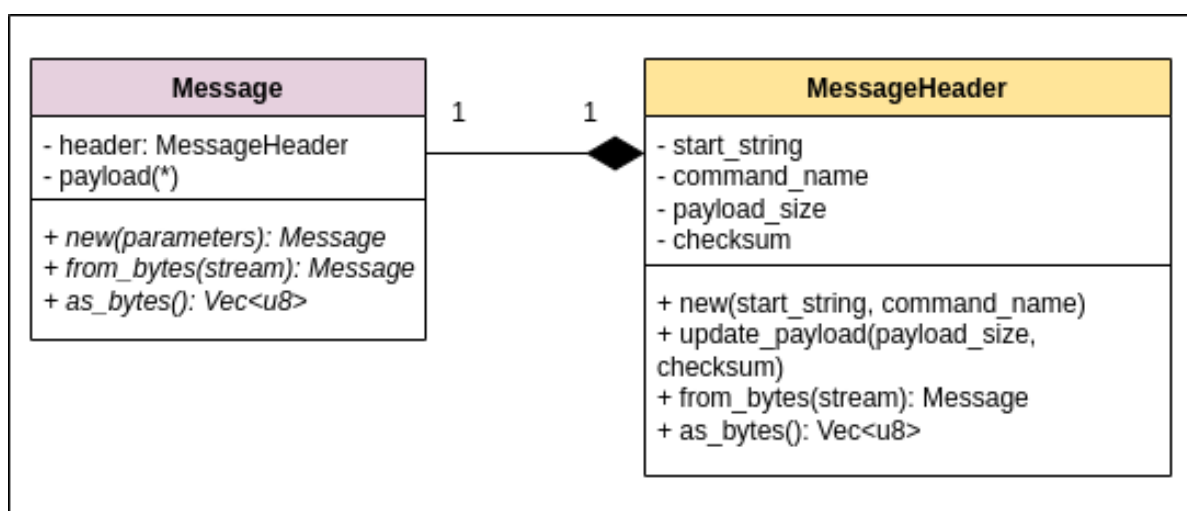


Figura 1: Estructura de los mensajes

2.2. Handshake

La conexión con otro nodo comienza a partir de lo que se denomina “handshake”, en donde se establecen versiones de protocolo y mensajes aceptados. En la imagen siguiente se puede observar un diagrama de secuencia de handshake, seguido el inmediato envío del mensaje “SendHeaders” para optar por la técnica de broadcasting 'Direct Headers Announcement'.

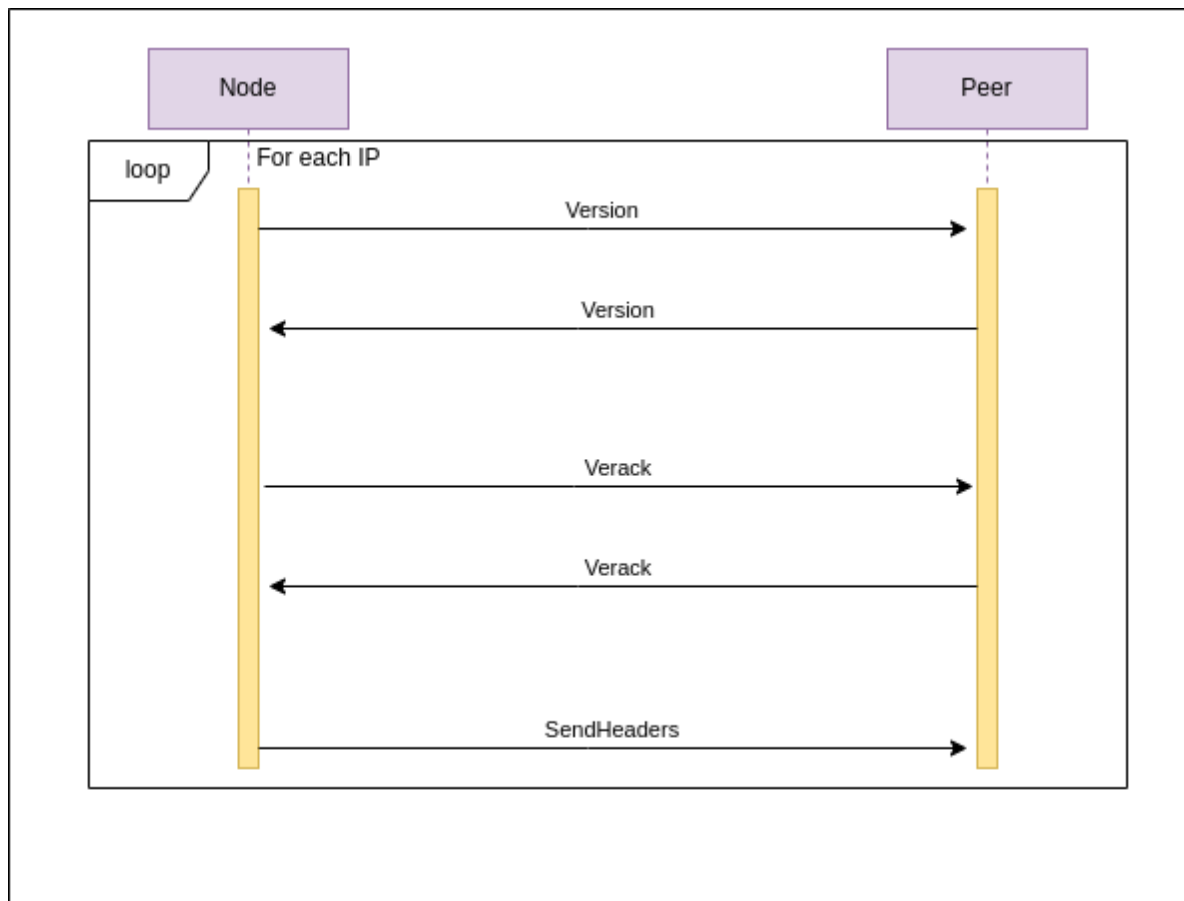


Figura 2: Diagrama de secuencia Handshake

2.3. Headers Download

Inmediatamente después de realizada la conexión inicial mediante el handshake, se inicia la descarga de headers, para la cual se toma a un peer de los que se estableció la conexión anteriormente, al cual se le llama “nodo sync”, y se le piden los headers mediante el mensaje GetHeaders. Cabe aclarar que no se pueden pedir la totalidad de los headers en un solo mensaje, como máximo se pueden pedir 2000, por lo que primeramente se le piden los primeros 2000 comenzando del bloque génesis, y luego se envía un GetHeaders con el header 2001 y se recibe un headers con los headers desde el 2001 al 4001, así sucesivamente hasta que el nodo sync nos responda con un headers con menos de 2000 headers, lo cual quiere decir que ya contamos con la totalidad de los headers de la red.

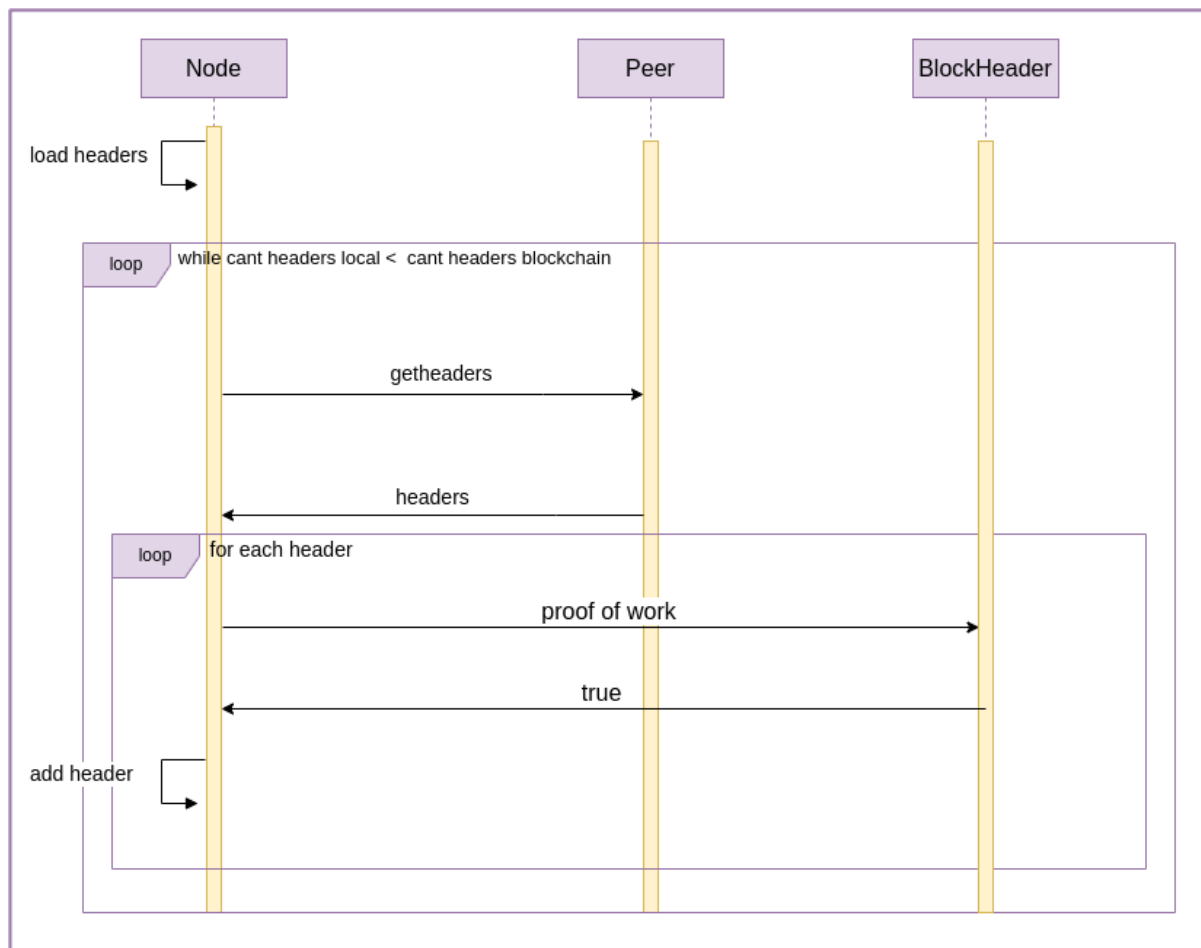


Figura 3: Diagrama de secuencia de la descarga de Headers

2.4. Blocks Download

Una vez que ya se cuenta con la totalidad de los headers descargados, se inicia la descarga de bloques, filtrando previamente los bloques anteriores a la fecha de inicio del proyecto. Esto se realiza mediante el mensaje GetData el cual es contestado con los bloques asociados a los headers pedidos.

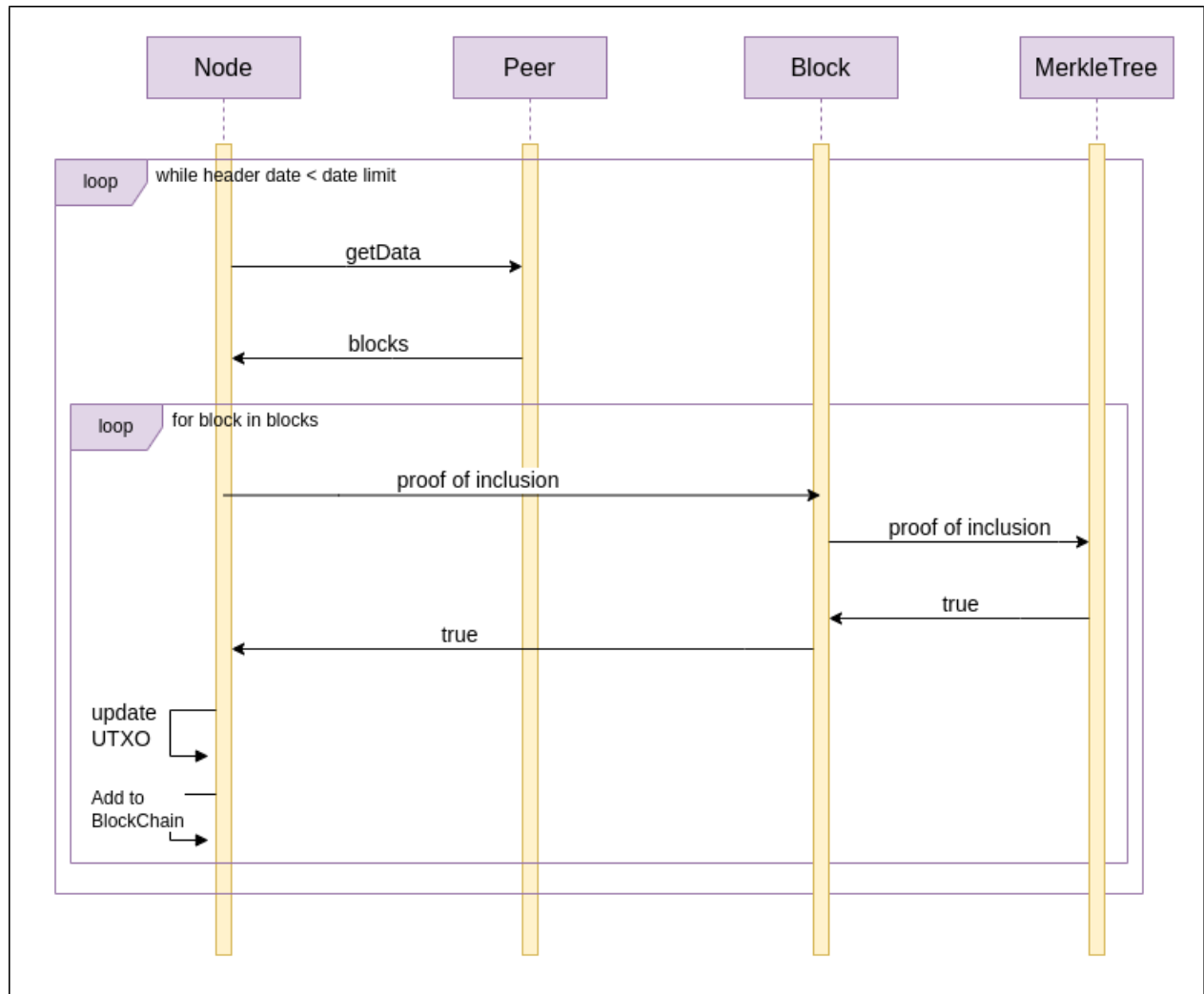


Figura 4: Diagrama de secuencia de la descarga de bloques

A diferencia de la descarga de headers, que se debe realizar de forma secuencial, la descarga de bloques puede ser optimizada utilizando múltiples hilos de ejecución. Para esto se implementó un thread pool, de forma que para cada una de las conexiones obtenidas en el handshake se lanza un hilo el cual se encarga de obtener una cantidad reducida de headers de una lista y envía el GetData correspondiente. Como estos hilos deben agregar los bloques a la blockchain y actualizar las unspent transactions, deberían bloquear estos recursos frecuentemente por lo tanto se decidió lanzar un hilo antes de iniciar la descarga que reciba los bloques por un channel y actualice estas estructuras. Esta

implementación evita que los hilos que descargan los bloques estén constantemente bloqueando estos recursos, lo cual generaría un enlentecimiento en la descarga.

2.5. Broadcasting

2.5.1 Block Broadcasting

Para comunicarle a los peers que se desean recibir headers de bloques en caso de que se haya minado uno recientemente, se envía un mensaje SendHeaders. El peer al minarse un nuevo bloque enviará al Nodo un mensaje Headers, y luego de realizarle la proof of work al mismo, se envía el mensaje GetData pidiendo información del bloque asociado. Una vez obtenido el bloque, se validan las transacciones del bloque mediante la Proof of Inclusion. Cabe destacar que se contempla el caso en que los peers actualicen al Nodo mediante el mensaje Inv, para el cual el procedimiento es el mismo.

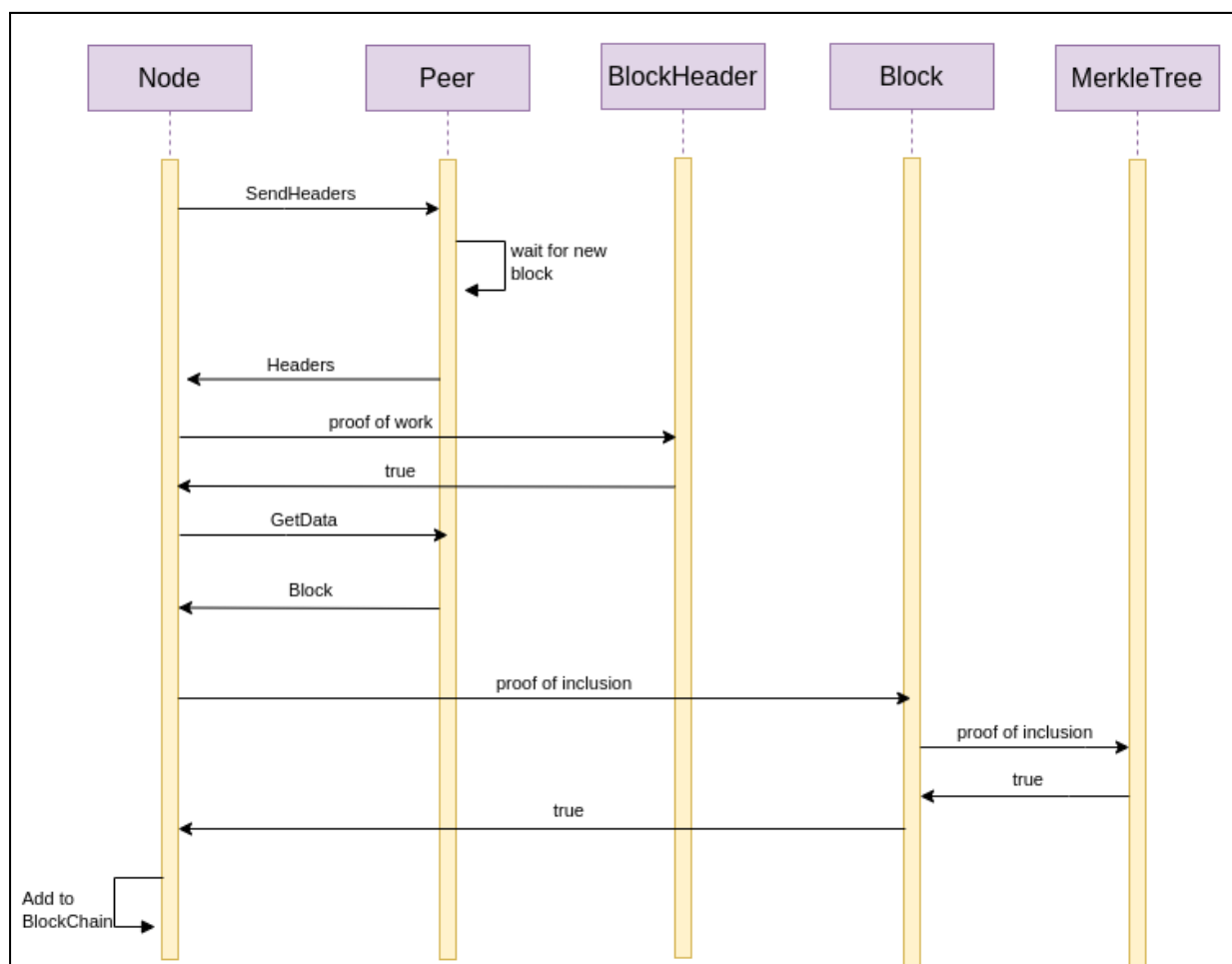


Figura 5: Diagrama de secuencia del block broadcasting

2.5.2 Transaction Broadcasting

Cada vez que se crea una transacción esta es broadcasteada entre los peers de la red para que se propague. A través del mensaje Inv con un tipo de dato de transacción, un peer le notifica al Nodo que una transacción ha sido creada. El Nodo responde con el mensaje GetData para obtener información asociada a la misma, y la agrega a la Mempool.

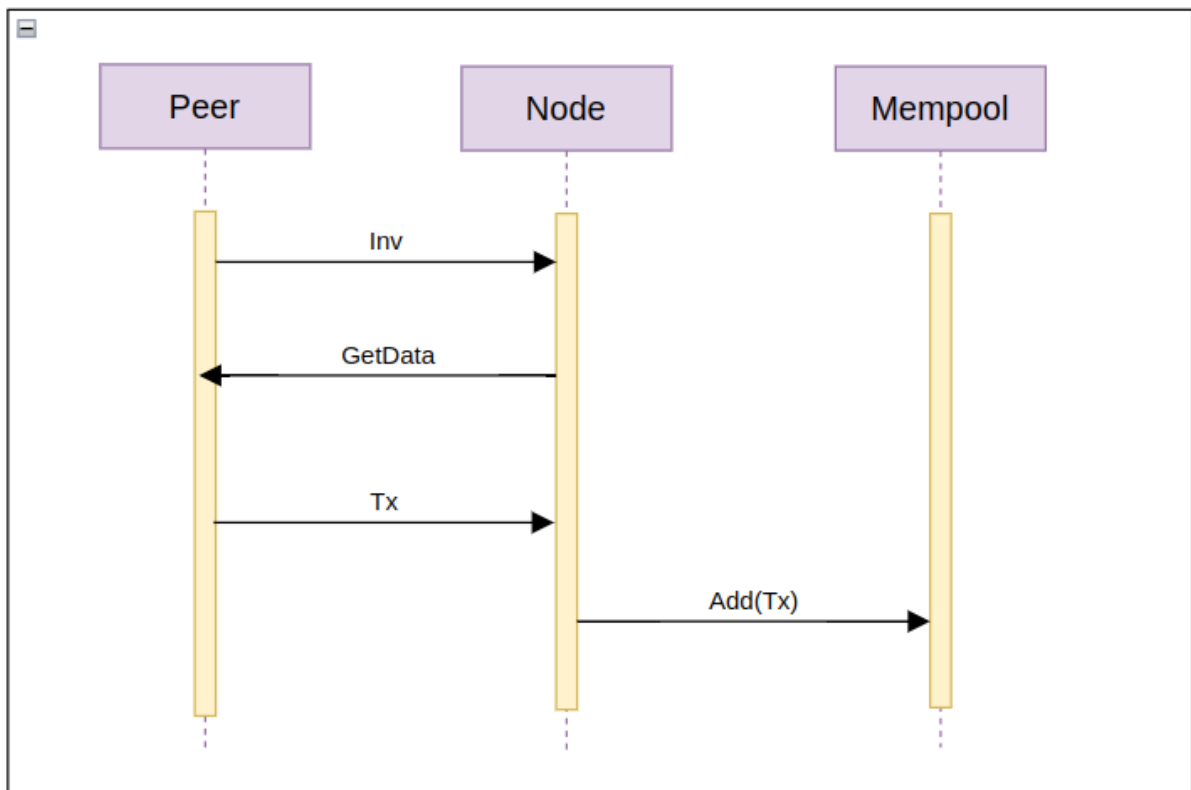


Figura 6: Diagrama de secuencia del tx broadcasting

3. Estructuras

3.1. Bloque

Un bloque se encuentra formado por un vector con las transacciones del mismo y un BlockHeader con su metadata.

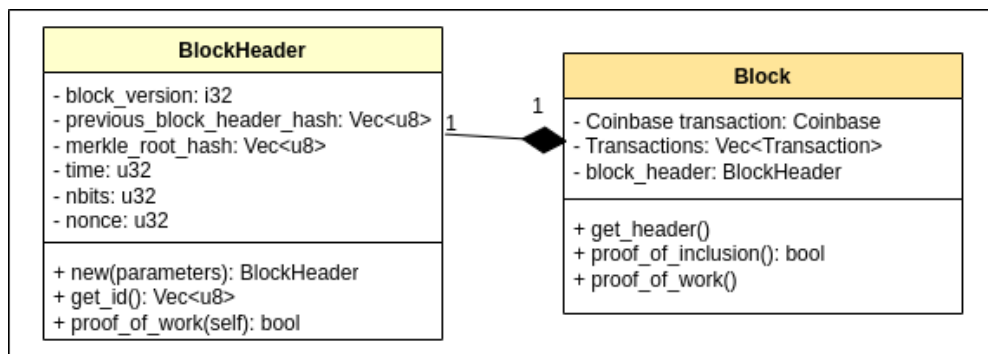


Figura 7: Estructura del bloque

3.2. Transacciones

Las transacciones se implementaron a partir de una estructura la cual contiene una lista de TxOut y una lista de TxIn.

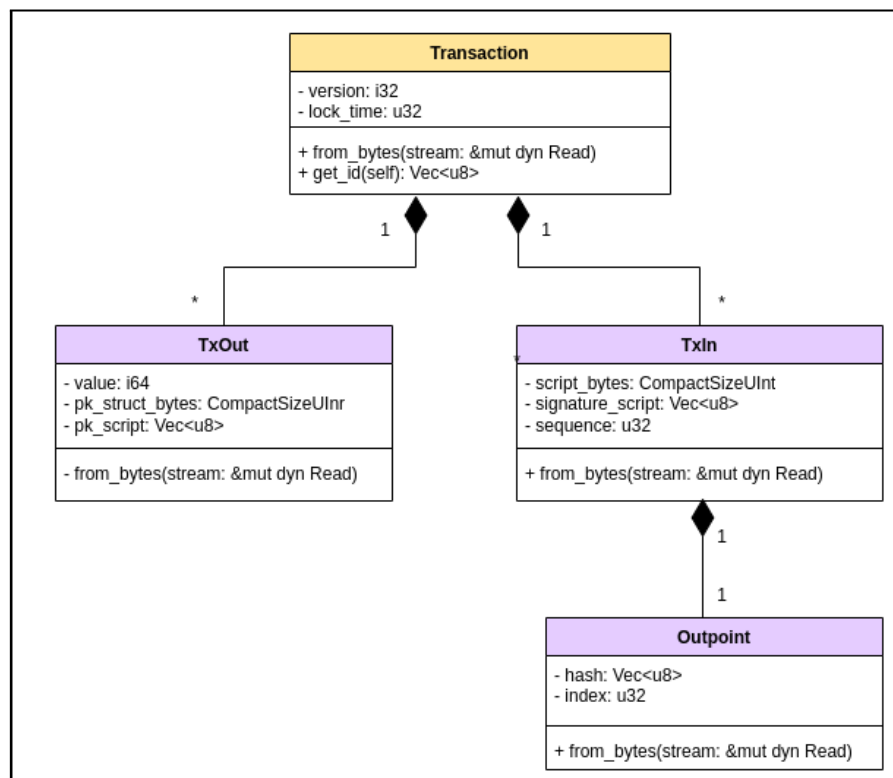


Figura 8: Estructura de una transacción

También existe la transacción Coinbase, la primera del bloque, mediante la cual el minero se cobra la recompensa por haber minado el mismo. Esta tiene una estructura distinta a las transacciones convencionales debido a que tiene una única TxIn que no posee Outpoint, y cuenta con la suma de las fees de todas las transacciones del bloque.

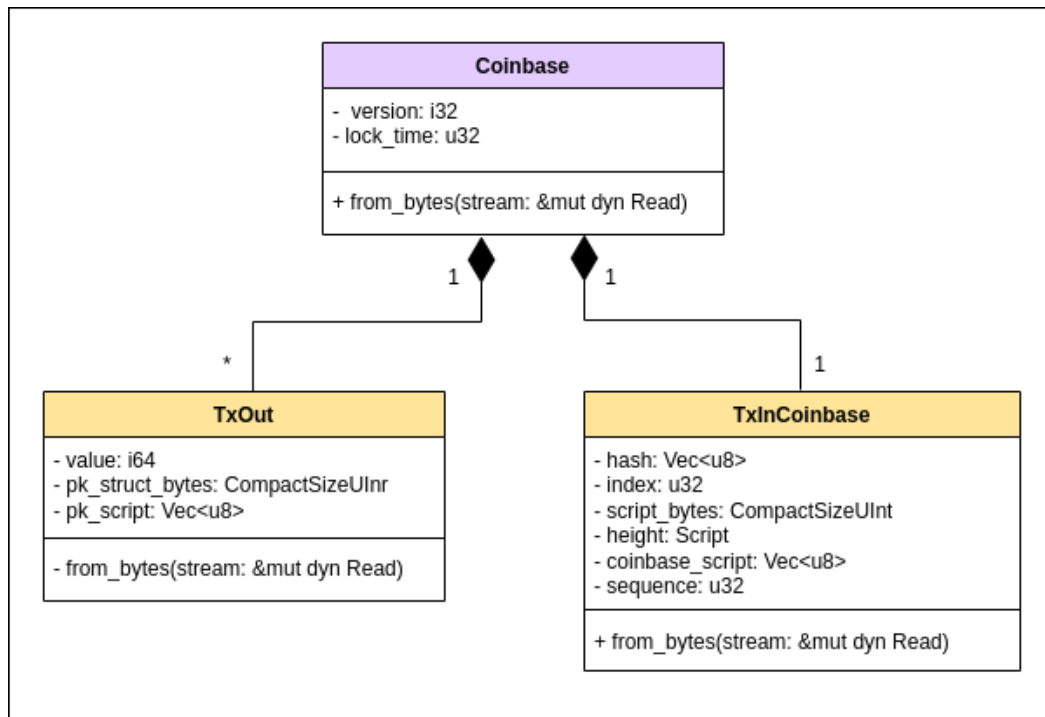


Figura 9: Estructura de una transacción Coinbase

3.2. UTXO set

Por cuestiones de performance, ya que el UTXO set es una estructura que se debe actualizar cada vez que se obtiene un bloque, se decidió implementarlo como un HashMap de HashMaps. Siendo las claves del primer hashmap los ids de las transacciones, y como valor otro hashmap donde las claves son los índices de las TxOut dentro de la transacción y como valor esta misma TxOut.

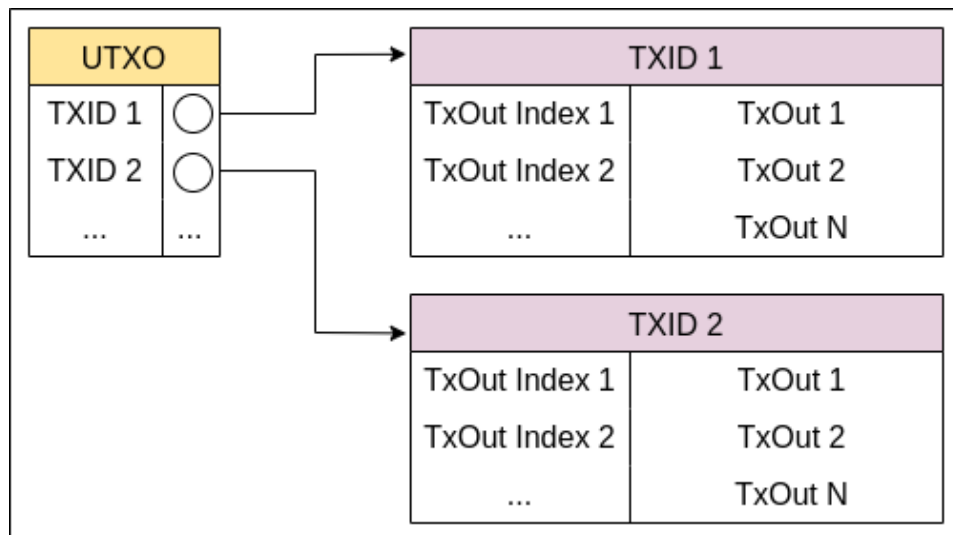


Figura 10: Estructura de las UTXO

Cada vez que obtiene un bloque, se recorren todas las transacciones del mismo eliminando las TxOut que han sido utilizadas en las TxIn de cada transacción y, a su vez, agregando las nuevas TxOut.

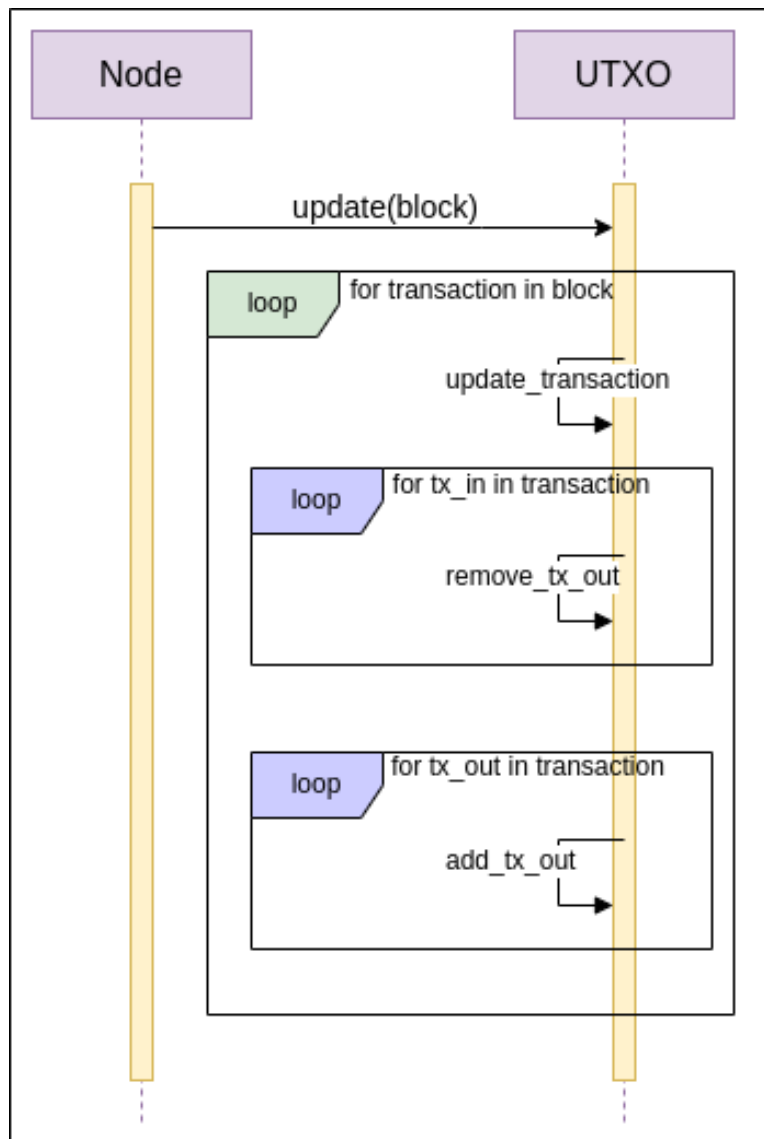


Figura 11: Diagrama de secuencia de actualización de UTXO

3.3. Blockchain

Debido a que es frecuente la búsqueda de un bloque particular en la Blockchain, por una cuestión de performance se decidió implementar su estructura a partir de un HashMap con el Id de los bloques como claves y los bloques correspondientes como valores. En vista de que también es necesario en ocasiones realizar un recorrido secuencial de la Blockchain, el Nodo guarda el header del último bloque descargado y, ya que los headers tienen el id del bloque anterior, es posible recorrer la misma de esta manera.

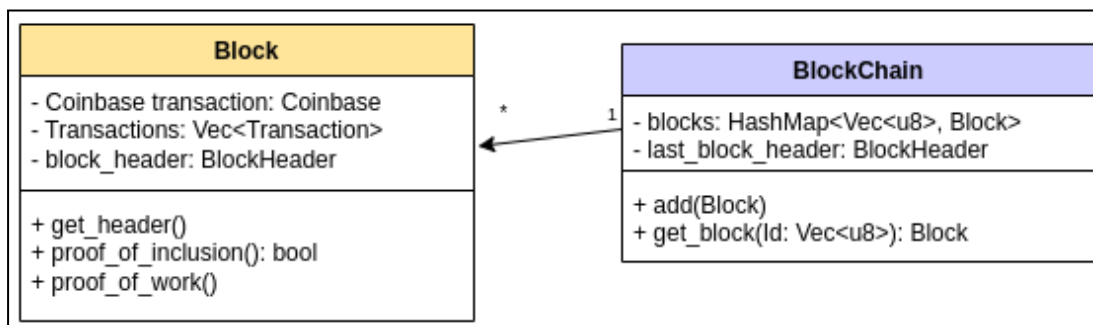


Figura 12: Diagrama de clases de la estructura Blockchain

4. Validación del bloque

4.1 Proof of Work

Mediante el campo nbits que posee el header del bloque se arma un número de 32 bytes representado como un vector, y se obtiene la threshold la cual luego es comparada byte a byte con el header del bloque el cual tiene que ser menor.

```
pub fn proof_of_work(&self) -> bool {
    let nbits: [u8; 4] = self.nbbits.to_be_bytes();
    let exp: u8 = nbits[0];
    let mantissa: Vec<u8> = nbits[1..].to_vec();

    let mut threshold: Vec<u8> = vec![0u8; (32 - exp) as usize];
    threshold.extend(iter::repeat(mantissa));
    threshold.extend(iter::repeat(0u8, (exp - 3) as usize));

    let mut block_header: Vec<u8> = self.get_header();
    block_header.reverse();

    for i: usize in 0..32 {
        match block_header[i].cmp(&threshold[i]) {
            Ordering::Greater => return false,
            Ordering::Less => return true,
            Ordering::Equal => {}
        }
    }
    true
}
```

Captura 1: Implementación de Proof of Work

4.2 Proof of Inclusion

Una vez que se tiene el bloque completo se toman todos los id de sus transacciones y se los va concatenando de a pares aplicándoles el doble hash 256 hasta llegar a un único hash, al cual se lo compara con la merkle root.

```
fn calculate_merkle_root(txn_list: &mut Vec<Vec<u8>>) -> Vec<u8> {
    let mut new_level_txn_list: Vec<Vec<u8>> = Vec::new();
    let mut merkle_root_hash: Vec<u8> = Vec::new();

    // Return when the length of the list is one
    if txn_list.len() == 1 {
        merkle_root_hash.extend(iter: txn_list[0].as_slice());
        return merkle_root_hash;
    }

    // If list is odd, duplicate and append the last item
    if txn_list.len() % 2 != 0 {
        if let Some(last_txn: &Vec<u8>) = txn_list.last() {
            txn_list.push(last_txn.clone());
        }
    }

    // Concatenation and hashing
    for i: usize in (0..(txn_list.len() - 1)).step_by(step: 2) {
        let mut concatenated_hash: Vec<u8> = Vec::new();
        concatenated_hash.extend(iter: &txn_list[i]);
        concatenated_hash.extend(iter: &txn_list[i + 1]);
        let hash: Vec<u8> = sha256d::Hash::hash(data: &concatenated_hash) Hash
            .to_byte_array() [u8; _]
            .to_vec();

        new_level_txn_list.push(hash);
    }

    calculate_merkle_root(&mut new_level_txn_list)
} fn calculate_merkle_root
```

Captura 2: Implementación de Proof of Inclusion

5. Wallet

Se implementó la Wallet en un crate aparte, dada esta implementación, se tendrá en ejecución el Nodo y la Wallet corriendo como dos procesos independientes. Estos se comunican a través de un socket TCP, siguiendo un protocolo de mensajes simple para realizar tres acciones:

- Actualizar las transacciones de la cuenta
- Broadcastear una transacción creada.
- Solicitar una Proof of Inclusion

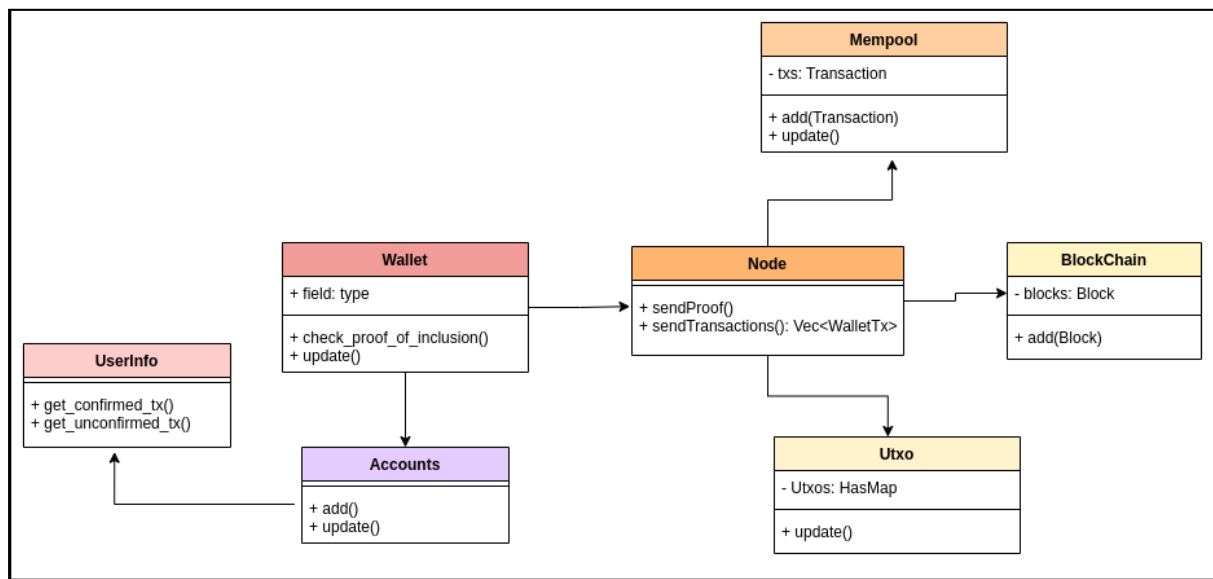


Figura 13: Diagrama de clases general del proyecto

5.1. Actualización de cuenta

Cada 5 segundos la Wallet le solicita al nodo las transacciones que se pudieron haber creado desde la última actualización. Se envía el pk_script y la pub_key de la cuenta para así poder identificar a las transacciones relacionadas a la misma, también el último epoch time de actualización para no tener que el Nodo no tenga que recorrer toda la blockchain.

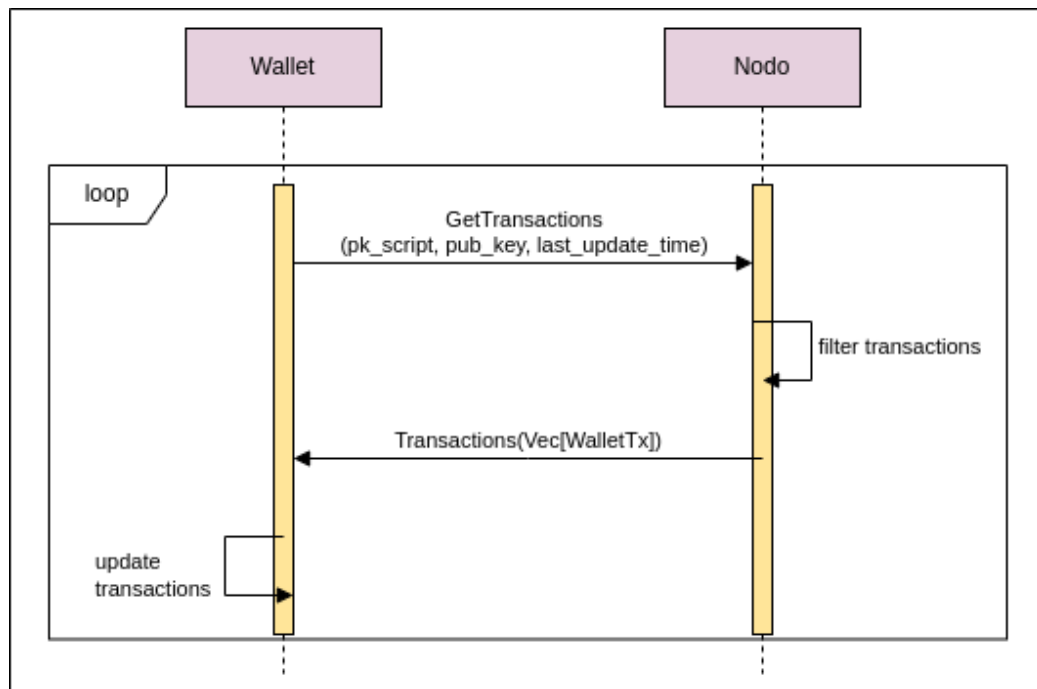


Figura 14: Diagrama de secuencia de actualización de cuenta

5.2. Broadcasting de nuevas transacciones

Cuando un usuario solicita la creación de una nueva transacción, la Wallet la crea y le solicita al nodo que broadcastee la misma. El nodo recibirá el mensaje “BroadcastTx” por parte de la Wallet con la transacción y este a su vez le enviará un mensaje Tx a todos los peers con los cuales se encuentra conectado. Esta transacción será eventualmente recibida por parte del broadcast de alguno de los peers y será agregada a la Mempool. El usuario observará la misma como una transacción no confirmada, una vez que se mine el bloque que contiene a la transacción, pasará a estar confirmada.

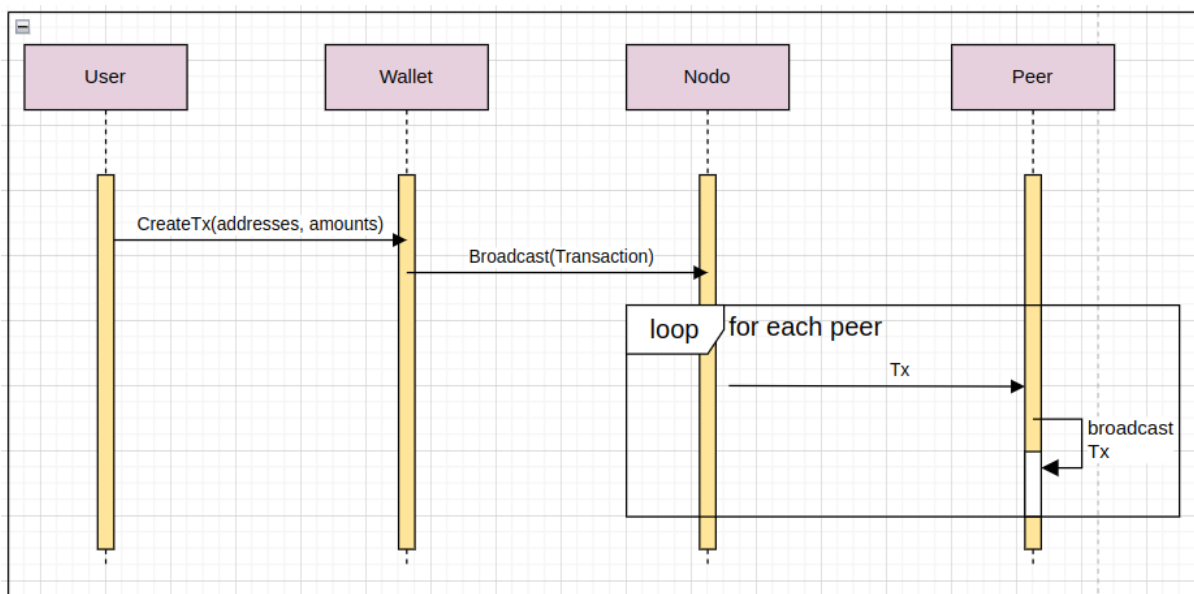


Figura 15: Diagrama de secuencia de actualización de cuenta

5.3. Proof of Inclusion

El usuario le pide a la Wallet una prueba de inclusión de una transacción a partir del Id de una transacción y el ID de un bloque. La Wallet a su vez le manda un mensaje "GetProof" al nodo, el cual calcula los Hashes y Flags necesarios para construir un merkle block y enviárselo en respuesta. Al recibir el merkle block, calcula la merkle root a partir del mismo y lo compara con la merkle root del block header para luego notificar al usuario el resultado.

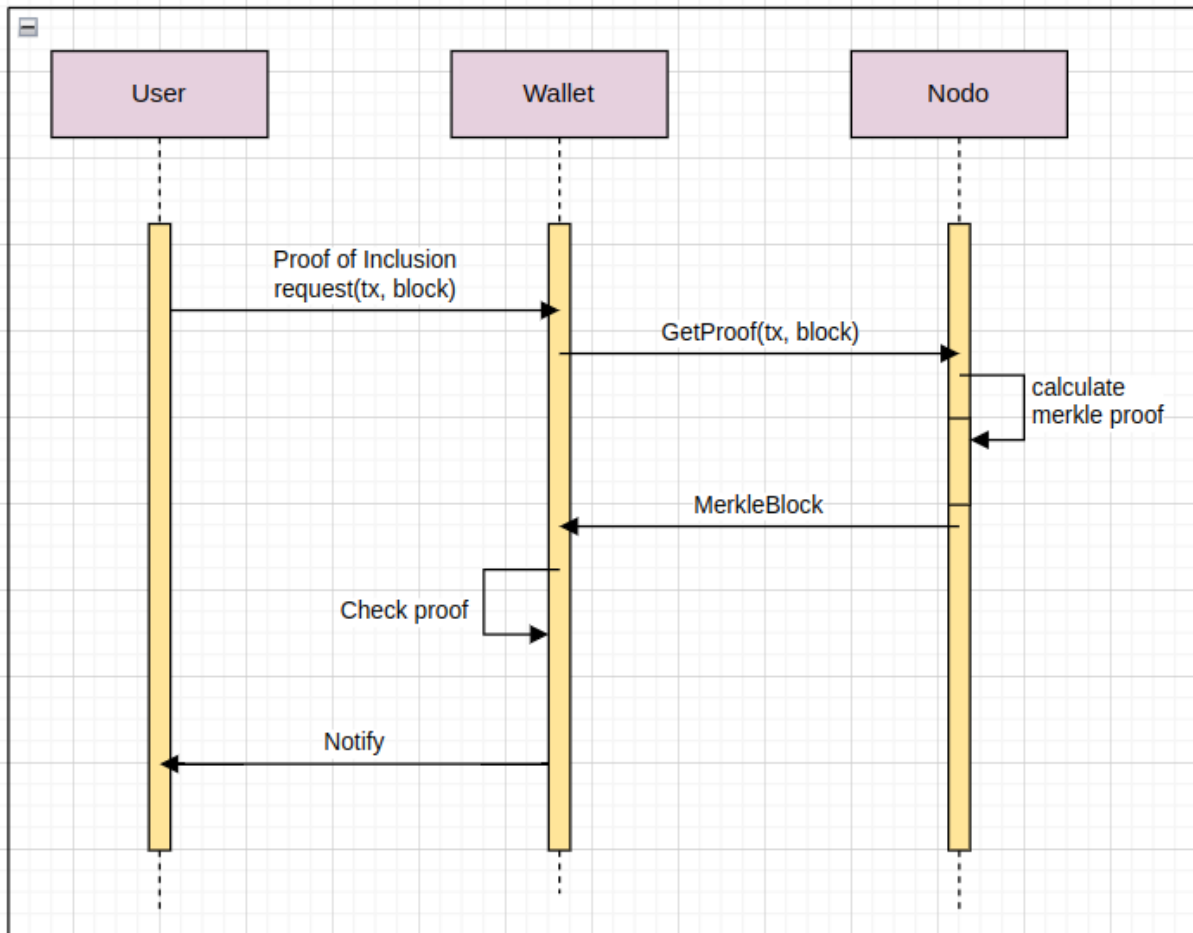


Figura 16: Diagrama de secuencia de la Proof of Inclusion

6. Interfaz Gráfica

Para la interfaz gráfica del proyecto se utilizaron las siguientes bibliotecas adicionales:

- gtk: Proporciona herramientas y widgets para diseñar interfaces coherentes en diferentes sistemas operativos.
- glib: Es una biblioteca de propósito general utilizada en el desarrollo de software de GNOME.
- gio: Es una biblioteca de alto nivel para manejar operaciones de entrada/salida y acceso a recursos del sistema.
- gdk: Se encarga de la representación gráfica y manejo de eventos en ventanas y objetos visuales.
- pango: Se utiliza para el manejo y renderización de texto multilingüe.

Además del uso nativo de gtk, se hizo uso del software Glade para facilitar el diseño de la misma.

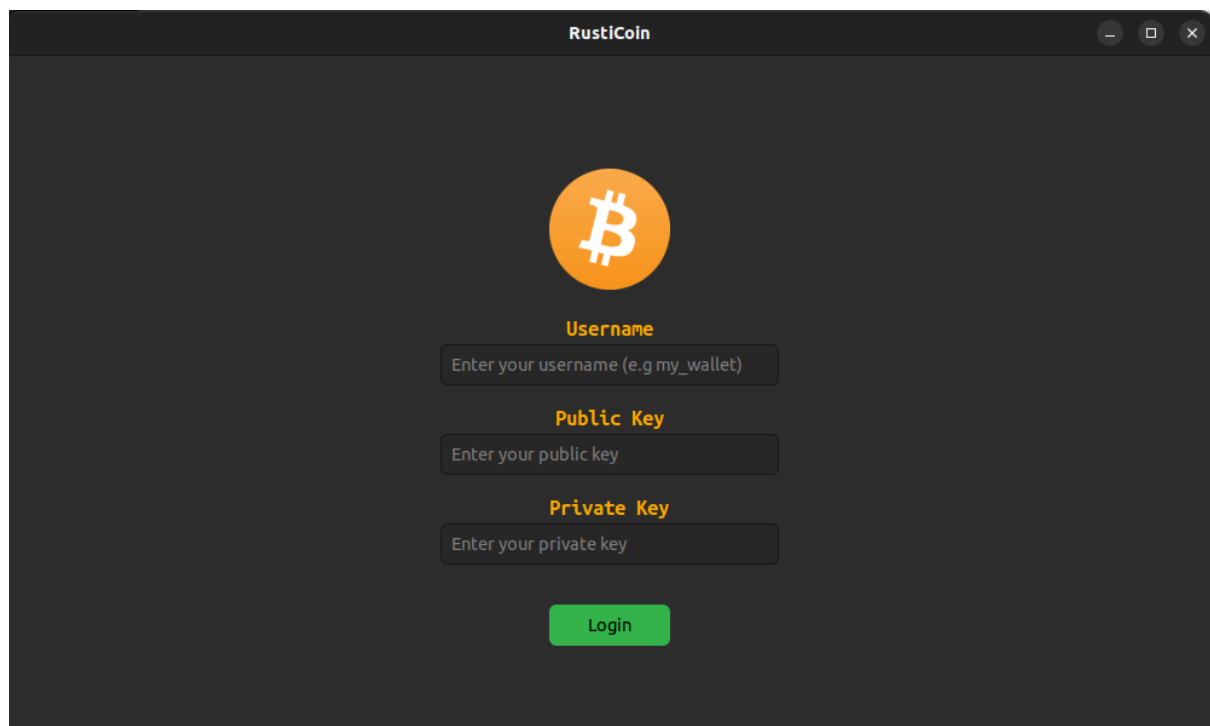
Se utilizó como base una aplicación de Wallet de Bitcoin ya existente llamada bitcoin-qt. Esta nos fue propuesta en el enunciado del proyecto.

Sin embargo solo se tomaron en cuenta las funcionalidades dentro de los requerimientos solicitados. Por lo tanto, se omitieron ciertas features que si bien facilitan algunos aspectos de la usabilidad de la UI, no son estrictamente necesarias para el correcto funcionamiento de la wallet.

6.1 Ventana de Login

Al ejecutarse la interfaz gráfica, en principio aparece la ventana de logueo. En ésta se le solicita al usuario que ingrese un nombre de usuario, asociado a una clave pública y una clave privada.

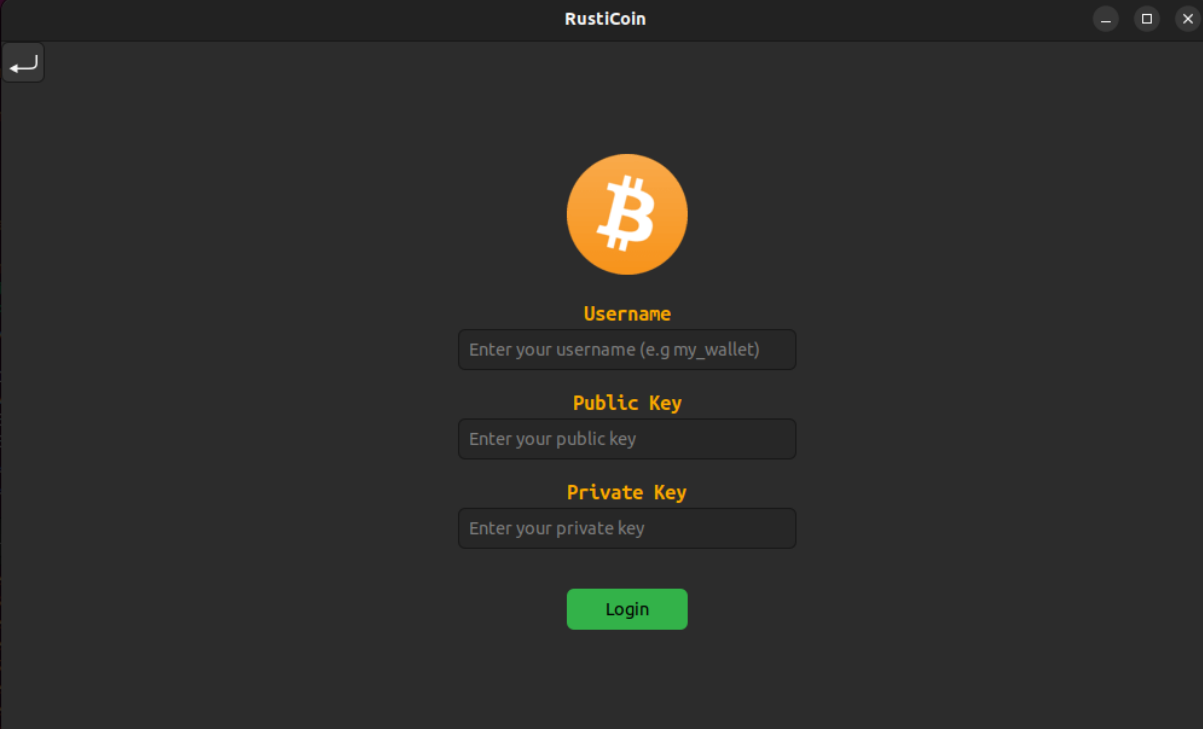
Se implementó una lógica simple para validar que el usuario ingrese claves de largo y contenido válido. Se asume que las claves pública y privada existen para garantizar el correcto funcionamiento de la Wallet.



6.2 Ventana de Principal (Main)

Una vez ingresados los datos, se pasa a la ventana principal de la Wallet donde se encuentran distintas pestañas. Cada una de ellas tiene una funcionalidad y propósito propio. Además se muestra el nombre del usuario actual junto con un botón para agregar una nueva. El botón de cuenta nueva vuelve a la ventana de login previamente explicada

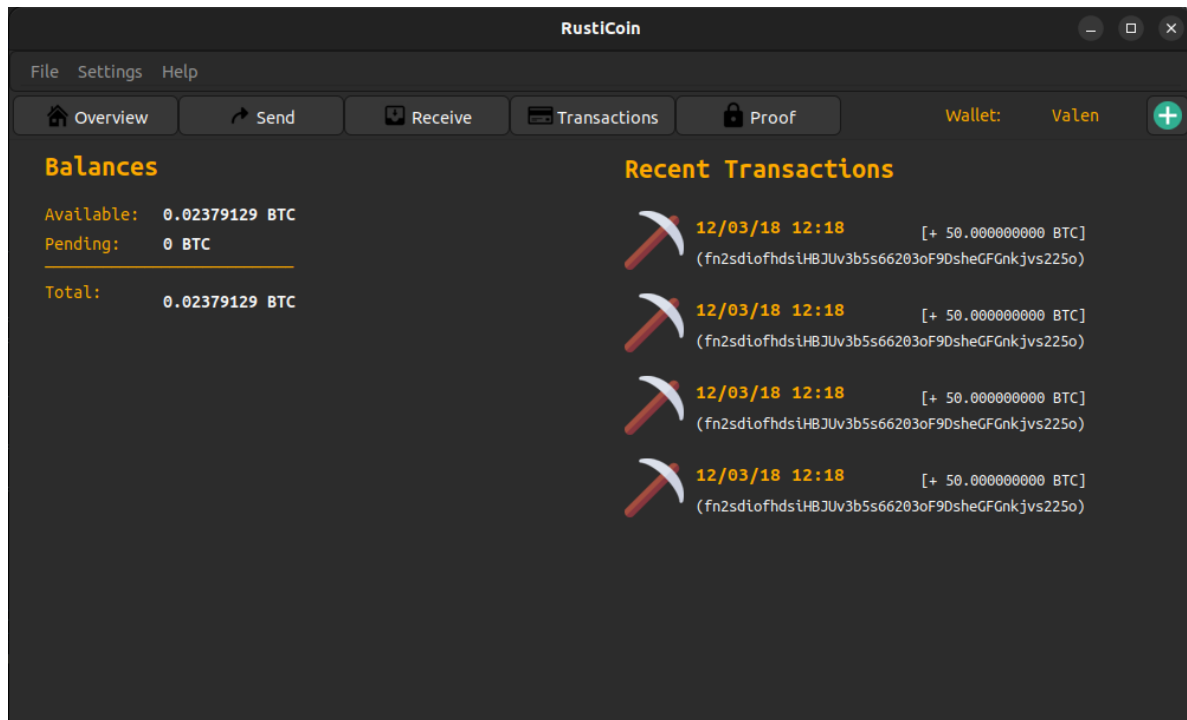
Si el usuario se arrepiente de crear una cuenta, también se le permite volver a la ventana principal nuevamente.



The screenshot shows the RustiCoin application window. The title bar at the top reads "RustiCoin" and includes standard window control buttons (minimize, maximize, close). The main content area has a dark background. At the top center is a large orange circle containing a white Bitcoin symbol. Below this, the word "Username" is displayed in orange. Underneath is a text input field with the placeholder "Enter your username (e.g my_wallet)". This is followed by "Public Key" in orange and another input field with the placeholder "Enter your public key". Then, "Private Key" is shown in orange with a third input field with the placeholder "Enter your private key". At the bottom center is a green button with the text "Login". A small back arrow icon is visible in the top-left corner of the application area.

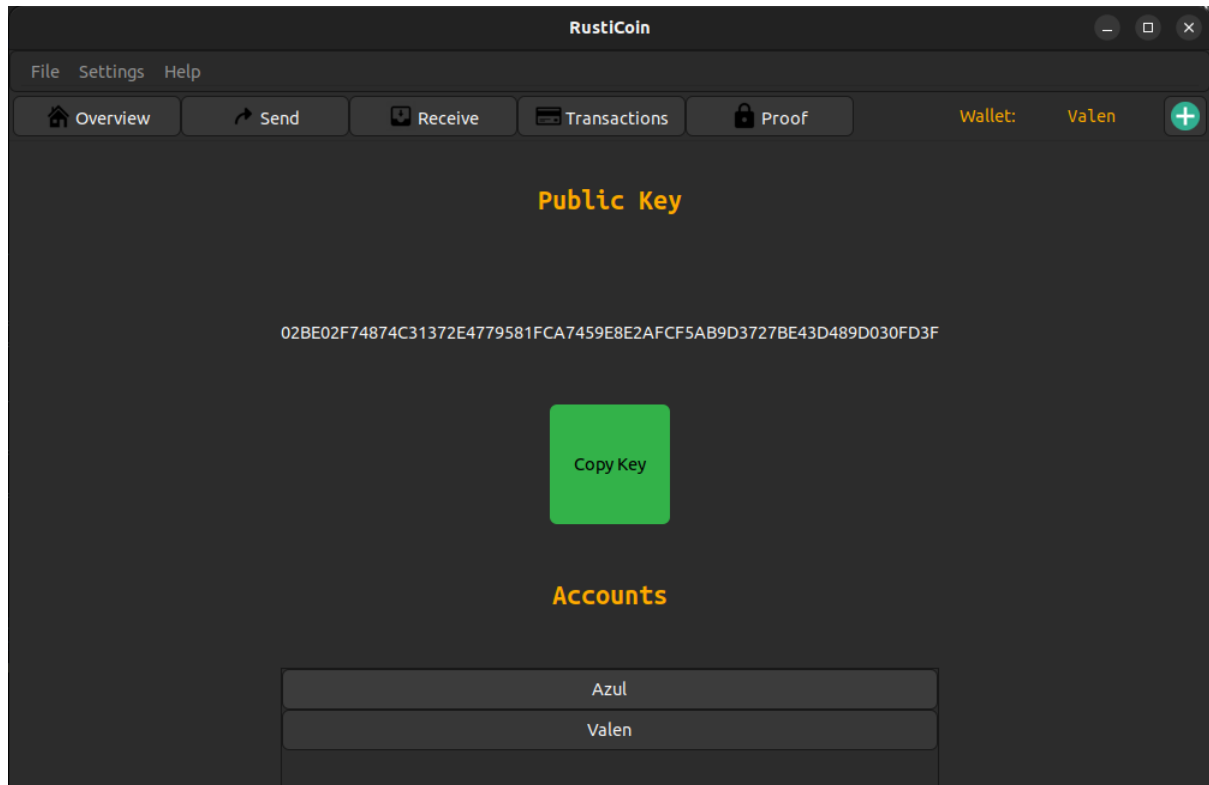
6.2.1 Pestaña de Overview

En esta pestaña se puede observar el balance de la cuenta y también un subconjunto de las últimas transacciones que involucran al usuario. Como su nombre indica, muestra una vista general de la Wallet en sí.



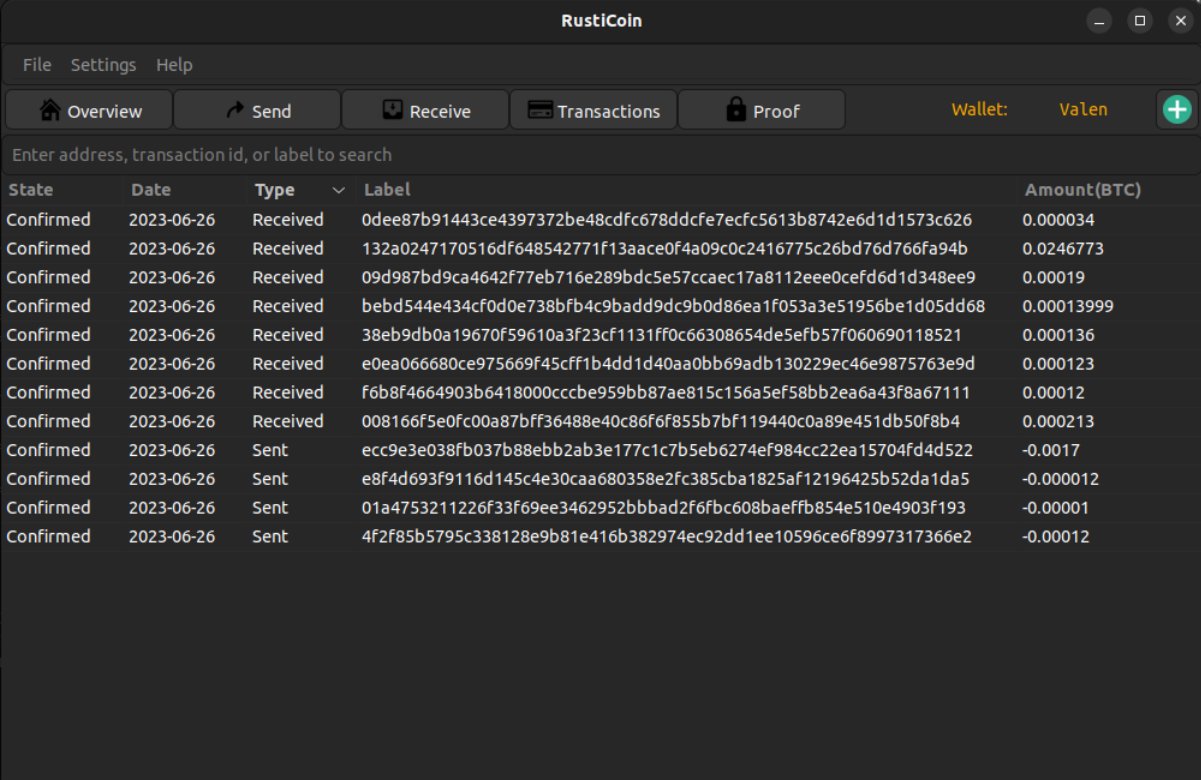
6.2.2 Pestaña de Receive

En esta pestaña podemos realizar el cambio de usuario (ya logueados) y también copiar de manera sencilla la public key.



6.2.3 Pestaña de Transacciones

En esta pestaña se observan las transacciones del usuario junto con su información. Las mismas pueden ser filtradas y también pueden ser ordenadas por campo.



State	Date	Type	Label	Amount(BTC)
Confirmed	2023-06-26	Received	0dee87b91443ce4397372be48cdfc678ddcfe7ecfc5613b8742e6d1d1573c626	0.000034
Confirmed	2023-06-26	Received	132a0247170516df648542771f13aace0f4a09c0c2416775c26bd76d766fa94b	0.0246773
Confirmed	2023-06-26	Received	09d987bd9ca4642f77eb716e289bdc5e57ccaec17a8112eee0cefd6d1d348ee9	0.00019
Confirmed	2023-06-26	Received	bebd544e434cf0d0e738bfb4c9badd9dc9b0d86ea1f053a3e51956be1d05dd68	0.00013999
Confirmed	2023-06-26	Received	38eb9db0a19670f59610a3f23cf1131ff0c66308654de5efb57f060690118521	0.000136
Confirmed	2023-06-26	Received	e0ea066680ce975669f45cff1b4dd1d40aa0bb69adb130229ec46e9875763e9d	0.000123
Confirmed	2023-06-26	Received	f6b8f4664903b6418000cccbe959bb87ae815c156a5ef58bb2ea6a43f8a67111	0.00012
Confirmed	2023-06-26	Received	008166f5e0fc00a87bff36488e40c86f6f855b7bf119440c0a89e451db50f8b4	0.000213
Confirmed	2023-06-26	Sent	ecc9e3e038fb037b88ebb2ab3e177c1c7b5eb6274ef984cc22ea15704fd4d522	-0.0017
Confirmed	2023-06-26	Sent	e8f4d693f9116d145c4e30caa680358e2fc385cba1825af12196425b52da1da5	-0.000012
Confirmed	2023-06-26	Sent	01a4753211226f33f69ee3462952bbbad2f6fbc608baeffb854e510e4903f193	-0.00001
Confirmed	2023-06-26	Sent	4f2f85b5795c338128e9b81e416b382974ec92dd1ee10596ce6f8997317366e2	-0.00012

6.2.4 Pestaña Send

En esta pestaña se puede crear una transaccion especificando los address destinatarios y el amount a depositar, tambien se puede aclarar el fee de la misma.

The screenshot shows the 'Send' tab of the RustiCoin application. The interface is dark-themed. At the top, there's a menu bar with 'File', 'Settings', and 'Help'. Below it, a navigation bar contains 'Overview', 'Send' (active), 'Receive', 'Transactions', and 'Proof'. On the right of the navigation bar, it says 'Wallet: Valen' with a green plus icon. The main area has two identical recipient entry forms. Each form has a 'Pay To:' label and a text input field with placeholder text 'Enter a BitCoin Address (e.g. n2wx0nfexkjwEPgd06iJA7T7RtzknHxhFc)'. Below each input is an 'Amount:' label, a numeric input field with '0,0000000', a minus/plus control, and a 'BTC' label with a red 'x' icon. At the bottom, there's a bar with 'Send', 'Clear all', and 'Add recipient' buttons. On the right of this bar, it shows 'Fee: 0,0000000' with a minus/plus control and a 'BTC' label.

6.2.5 Pestaña de Proof of Inclusion

En esta pestaña se le permite realizar al usuario una proof of inclusion. Dado un header de un bloque y un header de una transacción, al apretar el botón se avisa al usuario si los datos pasan la prueba. Solo lo hace si la transacción pertenece al bloque.

