Creando una criptomoneda

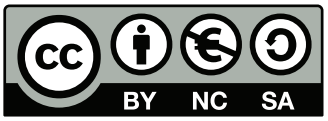
Fundamentos de Redes

Seminarios

Doble Grado en Ingeniería Informática y Matemáticas

Miguel Ángel Fernández Gutiérrez
Pedro Gallego López

UNIVERSIDAD DE GRANADA



Creando una criptomoneda

Fundamentos de Redes

Seminarios

Doble Grado en Ingeniería Informática y Matemáticas

Miguel Ángel Fernández Gutiérrez
Pedro Gallego López

UNIVERSIDAD
DE GRANADA

Índice

I | ¿Qué es una criptomoneda?

Bitcoin es una de las **criptomonedas** más conocidas. Sepamos bien cómo funciona o no, todos hemos oído hablar de Bitcoin. Sin embargo, hay muchas otras, que se basan en los mismos principios básicos. Por tanto, para explicar las criptomonedas, veamos cuáles son estos principios.

Supongamos que queremos crear una criptomoneda, ***DGIIMCoin***. La primera pregunta sería para qué, y la segunda –en caso de que el para qué nos haya convencido– es qué necesitamos para ello.

Crear todo esto tendría sentido si queremos crear un sistema anónimo, descentralizado y seguro para intercambiar dinero o información. Veamos bien qué quiere decir esto:

- **Anónimo:** los usuarios deben ser identificables para el sistema, pero su identidad debe ser custodiada con recelo.
- **Descentralizado:** los sistemas convencionales depositan la confianza de instituciones en terceros –i.e. Amazon en el envío de bienes, un banco, etc.–, en el caso de las criptomonedas su estado es mantenido gracias al consenso de los usuarios.
- **Seguro:** el sistema no permite la realización de transacciones fraudulentas.

Bien, para todo esto nuestro sistema debería tener una serie de requisitos:

Req. 1. El sistema no requiere de una **autoridad central**: su estado es mantenido mediante un **consenso** distribuido.

Req. 2. El sistema mantiene un **seguimiento** de todas las unidades de la criptomoneda y de sus propietarios.

Req. 3. El sistema define si se pueden crear **nuevas unidades** de la criptomoneda. Si esto es así, el sistema debe definir las circunstancias de su origen y cómo determinar quién será el propietario de éstas.

Req. 4. La propiedad de las monedas puede probarse de forma exclusivamente **criptográfica**.

Req. 5. El sistema permite la realización de **transacciones** de forma controlada.

Nuestro sistema, por tanto, girará en torno a las siguientes ideas clave:

1. **Descentralización.**
2. **Firmas digitales.**

3. **La contabilidad (*ledger*) es la propia moneda.**
4. **Mecanismo de consenso.**
5. **Blockchain.**

Explicaremos cada una de ellas con más detalle.

1 Descentralización: redes P2P

Para poder cumplir los dos primeros requisitos de nuestra lista, eliminando así autoridades centralizadas, temos ya una solución disponible: las **redes *peer-to-peer***, o **redes P2P**.

La idea principal de estas redes es que la información se comparta de igual a igual, al contrario a las redes cliente-servidor. En este caso, todos los ordenadores tienen el mismo peso.

1.1 Características de las redes P2P

Una red ***peer-to-peer*** (P2P) es una red de nodos en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino que cada nodo actúa simultáneamente como cliente y servidor respecto a los demás nodos de la red, todos ellos por igual. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.

Las redes *peer-to-peer* aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, y obtienen así más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

Los rasgos a destacar de este tipo de red son:

- **Escalabilidad.** Las redes P2P tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos estén conectados a una red P2P, mejor será su funcionamiento. Así, cuando los nodos llegan y comparten sus propios recursos, los recursos totales del sistema aumentan. Esto es radicalmente opuesto al caso de la arquitectura servidor-cliente con un sistema fijo de servidores, en los cuales la adición de clientes podría significar una transferencia de datos más lenta para todos los usuarios.
- **Descentralización.** Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red. Nótese sin embargo que, en realidad, algunas redes comúnmente llamadas P2P no cumplen esta característica, como en Napster, eDonkey, BitTorrent, etc.
- **Distribución de costes entre los usuarios.** Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.
- **Robustez.** La naturaleza distribuida de las redes *peer-to-peer* también incrementa la robustez en caso de haber fallos en la réplica excesiva de los datos hacia múltiples destinos, y en sistemas P2P

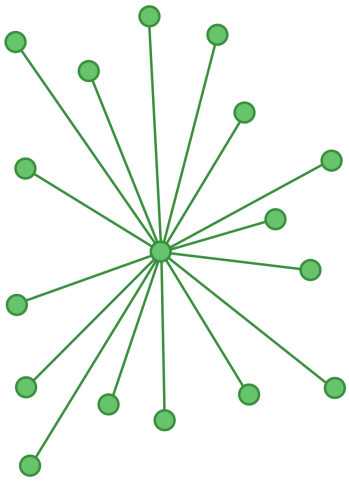
puros permitiendo a los peers encontrar la información sin hacer peticiones a ningún servidor centralizado de indexado. En el último caso, no hay ningún punto singular de falla en el sistema.

- **Anonimato.** Es deseable que en estas redes quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo, siempre que así lo necesiten los usuarios. Muchas veces el derecho al anonimato y los derechos de autor son incompatibles entre sí, y la industria propone mecanismos como el DRM para limitar ambos.
- **Seguridad.** Es una de las características deseables de las redes P2P menos implementada. Los objetivos de un P2P seguro serían identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje de las comunicaciones entre nodos, creación de grupos seguros de nodos dentro de la red y protección de los recursos de la red, entre otros. La mayor parte de estas características aún están bajo investigación, pero los mecanismos más prometedores son: cifrado multiclave, cajas de arena, gestión de derechos de autor, etc.

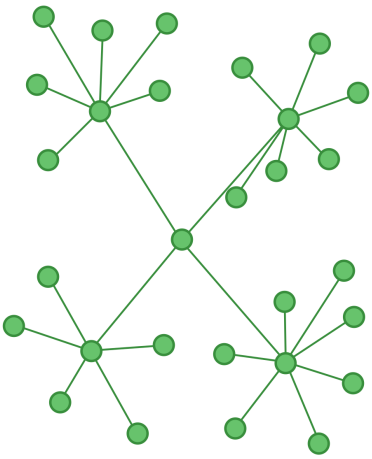
1.2 Topologías de las redes P2P

Una red *peer-to-peer* puede tener diversas topologías diferentes:

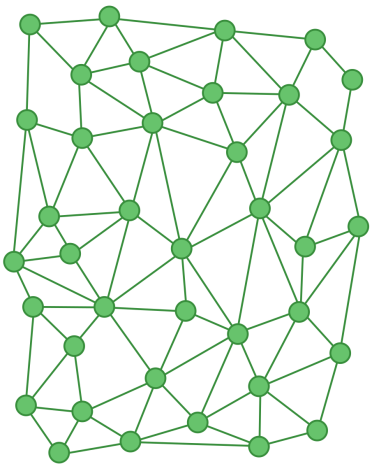
- **Centralizada.** Se mantiene un directorio en un servidor central, al cual las computadoras conectadas hacen peticiones para encontrar los nodos que contienen los contenidos deseados. Su principal defecto es que ese servidor central es un punto crítico.
- **Descentralizada y estructurada**, también conocida como **P2P híbrida**. No existe un directorio en un servidor central, sino en varias computadoras colocadas en lugares de la red que hacen fácil su acceso a otras computadoras.
- **Descentralizada y no estructurada.** No existen computadoras o nodos que funcionen como controladores centrales de peticiones. Todos los nodos funcionan como clientes y como servidores.



centralizada



descentralizada y estructurada



descentralizada y no estructurada

1.3 Ventajas e inconvenientes de las redes P2P

La principal ventaja que presenta P2P es la creación de grandes bases de datos de manera gratuita, ya que todos los ordenadores conectados en línea pueden descargarse archivos de otros ordenadores también conectados. Con el aumento de la velocidad de conexión de Internet, propiciada por la instalación del ADSL, los programas de intercambio de archivos y la frecuencia de este tipo de operaciones aumenta de forma considerable.

Muchos de los programas P2P son gratuitos, lo cual los hace una opción atractiva para quienes buscan contenido gratuito (la legalidad de esa práctia es cuestionable). Existen programas P2P con contenido legal y que conllevan una suscripción de pago, lo cual sigue siendo una buena opción si se busca un precio económico. Es por tanto la legalidad de las funcionalidades que te ofrece una red P2P determinada lo que puede cuestionar el uso de ésta.

En conclusión, tendríamos las siguientes ventajas:

- **Costo:** muchos de los programas P2P son gratuitos.
- **Eficiencia:** compartir archivos en P2P es fácil y rápido.

Sin olvidar los inconvenientes:

- **Legalidad:** compartir archivos ilegales por estas redes.

En definitiva, podemos usar los protocolos P2P existentes para implementar *DGIIMCoin*. Genial, continuemos.

2 Firmas digitales, criptografía y *hashing*

Para que nuestro sistema funcione son esenciales las **identidades digitales**, es decir, formas de verificar que una transacción se ha hecho realmente (alguien de verdad ha enviado dinero a otra persona). Para eso, usamos la **criptografía**.

2.1 Funciones hash

En concreto, necesitaremos un algoritmo de **hashing**. Esencialmente, una función hash es una función que, dada una entrada de un tamaño arbitrario, da una salida de tamaño fijo, e ilegible.

Una buena función hash debe garantizar:

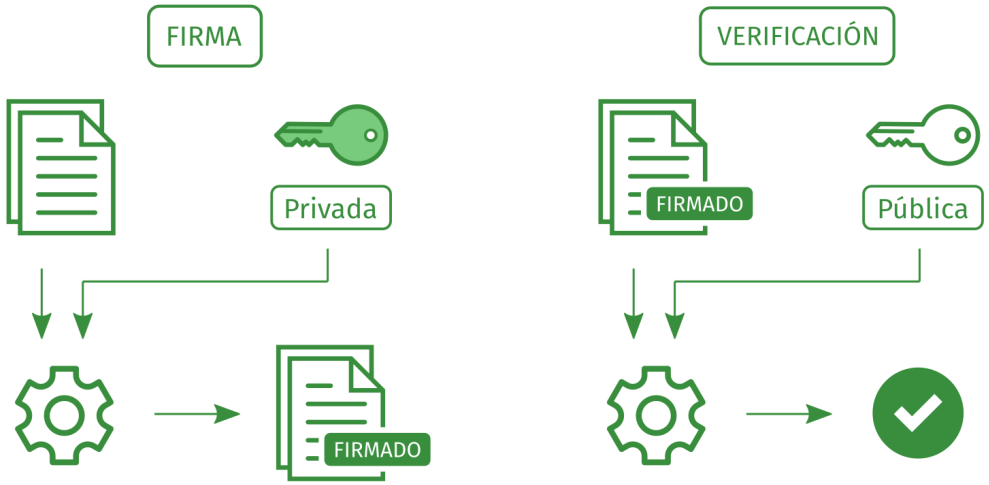
1. La salida de la función hash debe tener un tamaño fijo (por ejemplo, en el caso de SHA256 el tamaño es de 256 bits).
2. Un mínimo cambio en la entrada debe producir un enorme cambio en la salida.
3. Una misma entrada siempre producirá la misma salida.
4. No debe haber forma alguna de revertir el cambio, es decir, de que a partir de la salida se pueda encontrar la entrada.
5. Calcular el valor hash debe ser rápido; no debe requerir de un gran trabajo computacional.

Es importante mencionar que, aunque el número de valores hash posibles es limitado, las colisiones son muy poco probables, e incluso en caso de que se produzca alguna, sería imposible encontrar el patrón que dichas colisiones siguen.

2.2 Firmas digitales

Ahora bien, estas funciones son importantes especialmente a la hora de garantizar que las transacciones en el sistema son ciertas. Las funciones hash son utilizadas por algoritmos criptográficos que garantizan la veracidad de las transacciones.

Para asegurarnos de que nadie puede hacer una transacción en tu nombre, usaremos el concepto de **firma digital**. La mejor forma de hacerlo es mediante el método de **clave privada-clave pública**, también llamado **criptografía asimétrica**.



Cada usuario tendrá dicho conjunto de claves, cada una de ellas será un conjunto de bits determinado. Nos aseguraremos que nadie tiene acceso a nuestra clave privada. Para producir una firma, usaremos la siguiente función:

$$\textit{Firmar}(\textbf{Mensaje}, \textbf{Clave Privada}) = \textbf{Firma}$$

Que dependa de la clave privada significa que tú eres el único capaz de firmar, y que también dependa del mensaje asegura que los mensajes no puedan modificarse una vez firmados, porque el resultado de esta función sería radicalmente diferente. Para comprobar nos basta usar la clave pública, pues ésta tiene una relación con la clave privada.

$$\textit{Verificar}(\textbf{Mensaje}, \textbf{Firma}, \textbf{Clave Pública}) = \begin{cases} \textit{true} \\ \textit{false} \end{cases}$$

3 Enviando transacciones a la red

Ya casi estamos. Hemos implementado comunicación P2P, mecanismos para crear identidades digitales y formas para que los usuarios puedan firmar y garantizar que la información es correcta. Ahora sólo nos queda enviar información al sistema.

Como ya sabemos, no tenemos una autoridad central que valide cuánto dinero tenemos, pero tampoco hace falta: aquí reside una de las ideas, que la **contabilidad es la propia moneda**. Esto se debe a que para saber cuánto dinero tenemos, simplemente nos basta con tener una lista de todas las transacciones que hemos efectuado. Supongamos que tu historial de transacciones contiene la información:

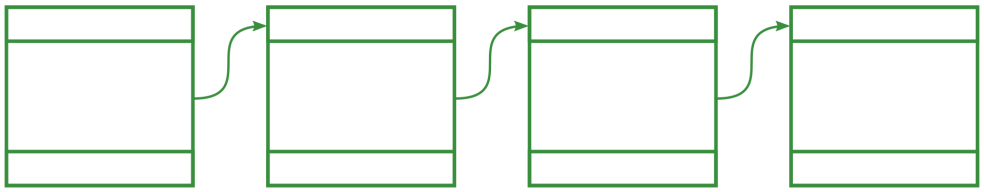
> **Ejemplo.** Listado de transacciones de un usuario en *DGIIMCoin*
>
> 1. Tengo 500Đ.
> 2. Envío 20Đ a alguien para unos apuntes de Modelos de Computación (incluiremos su clave pública).
> 3. Quiero enviar 1Đ como impuesto de transacción al sistema (lo veremos más adelante).

Lo único que queda es usar la red P2P para enviar estas transacciones al resto de usuarios del sistema, sin olvidar de firmarlas usando nuestra clave privada. Tras esto, todo el mundo podrá ver la transacción (aunque las identidades de ambos estarán encriptadas).

Listo. Sin embargo, no tendrás los apuntes hasta que la red coincida en que inicialmente tenías 500Đ, y por tanto dicha transacción puede efectuarse. Una vez que la transacción se valide, tu compañero te dará los apuntes.

4 El blockchain

Para poder mantener nuestro sistema necesitamos tener un historial de las transacciones que han sido realizadas. Para esto, usamos la tecnología **blockchain**, que es esencialmente un conjunto de bloques que contienen información (que veremos más adelante), que son enlazados usando criptografía: cada bloque tiene un hash del bloque anterior, así como metadatos de su creación.



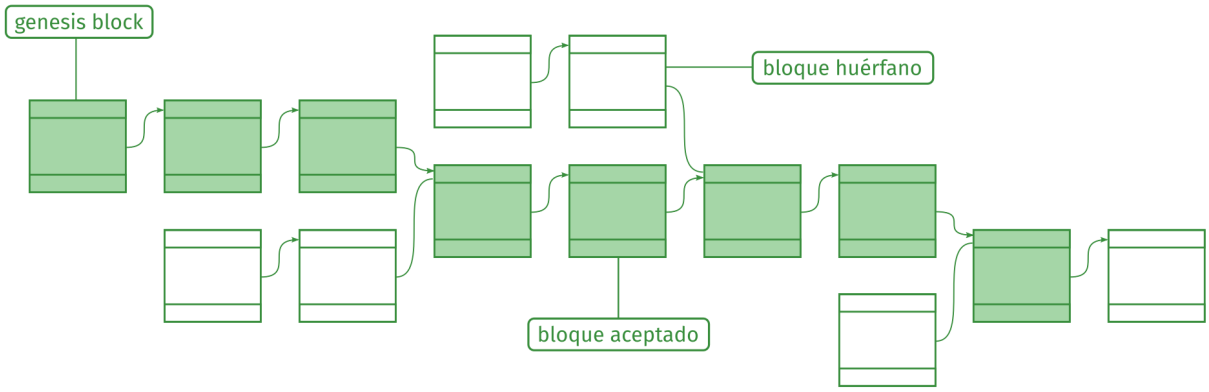
Esto precisamente hace que nuestro libro de cuentas público (*public ledger*) sea resistente a modificaciones: la alteración de un bloque invalida todos los sucesivos. Aunque es posible efectuar estas alteraciones, esta tecnología es segura y tolerante a faltas bizantinas[1].

4.1 Los bloques

Los bloques contienen un conjunto de transacciones que son hasheadas y codificadas en un **árbol de Merkle** (la estructura de datos que mantiene los enlaces entre los bloques mediante funciones hash).

[1]La **tolerancia a faltas bizantinas** es la resistencia de un sistema informático a fallas de componentes electrónicos donde hay información imperfecta sobre si un componente falla. Es de especial importancia en sistemas distribuidos, como el que nos concierne.

A veces dos bloques pueden ser creados de forma concurrente, generando un *fork* temporal. Por esto, necesitaremos una forma de "puntuar" diferentes versiones, para que uno con una puntuación más alta pueda ser seleccionado antes que el resto. Los bloques no seleccionados para la inclusión son los denominados **bloques huérfanos**.



Más adelante, veremos la estructura exacta de estos bloques, en el caso de Bitcoin.

5 Mecanismos de verificación de transacciones: *mining*

Bien, ya que tenemos mecanismos para generar y guardar transacciones de forma descentralizada, nos falta el factor de la **confianza**: ¿cómo sabemos que una transacción es cierta?

Supongamos la siguiente situación: María debe pagarle 100Đ a Jorge, y de hecho tiene el dinero disponible, pero quiere timar a Jorge haciéndole creer que en efecto se los ha pagado, enviando esta transacción a Jorge, pero no al resto de usuarios de nuestro sistema. Para Jorge, María habrá saldado sus deudas con él, mientras que realmente Jorge nunca tendrá ese dinero.

Para evitar esto, es necesario establecer un sistema mediante el cual se pueda establecer un criterio para discernir qué bloque de nuestro blockchain es más correcto. Deberíamos, por tanto, implementar en este momento un mecaniso de autenticación sustentado por el interés colectivo.

En nuestro blockchain hay unos actores especialmente destacables, que aportan una funcionalidad esencial para el buen funcionamiento de nuestra criptomoneda. Estos son los **mineros**, que son los encargados de crear bloques nuevos donde poder guardar las transacciones. Estos mineros no son otra cosa que un usuario de nuestra criptomoneda cuya aportación es potencia de cómputo a un problema matemático.

Pero, un segundo, *¿un problema matemático en un bloque?* Efectivamente, como antes hemos comentado, tenemos una función hash SHA-256 (en el caso de Bitcoin), que tiene como output un número de 256 bits, es decir, se pueden generar 2^{256} resultados. Y es aquí donde se nos plantea el problema matemático:

> Encontrar un número que, al ser introducido en la función hash junto con los datos del bloque que se quiere crear, el resultado sea un número que empiece por muchos 0.

Nuestros mineros tratarán de resolver este problema y el número del minero que consiga más ceros será el escogido. Pero, ¿qué motivación tienen estos mineros para trabajar en formar bloques y no en corromperlos?

Pensemos en una forma de beneficiar a los que se ofrecen a hacer crecer a la comunidad. Obtendremos diversos esquemas para ello, y de ese modo hacer que la blockchain funcione de manera independiente, segura, transparente y democrática. Además, premia a los mineros con las mismas criptomonedas y promueve que sigan minando criptomonedas y que lo hagan de forma honesta.

5.1 Proof-of-work (PoW)

El esquema de **proof-of-work (PoW)**, o **prueba de trabajo**, es un algoritmo que premia a los participantes que resuelvan acertijos criptográficos para validar las transacciones y crear nuevos bloques. Este es el esquema que utiliza Bitcoin. **Tienen más posibilidad de resolver los acertijos aquellos mineros que tengan mayor poder computacional trabajando a disposición del sistema**. De esto el nombre del sistema, pues da prioridad a aquellos que más trabajan.

En este esquema, los mineros compiten por resolver los acertijos para crear nuevos bloques de la blockchain, en el caso de Bitcoin cada 10 minutos. Los mineros que inviertan más trabajo en crear los bloques tienen más posibilidades de encontrar la respuesta y como recompensa obtendrán criptomonedas.

Una de las mayores desventajas del PoW es que gasta mucha energía, pues se desperdicia el trabajo de todos los mineros que intentaron encontrar la respuesta correcta pero que no lo lograron.

5.2 Proof-of-burn (PoB)

Proof-of-burn (PoB) es un algoritmo similar al de prueba de trabajo, pero con tasas reducidas de consumo de energía. El proceso de validación de bloques de las redes basadas en PoB no requiere el uso de recursos computacionales potentes y no depende de hardware minero potente (como los ASIC). En cambio, las criptomonedas se queman intencionadamente como una forma de "invertir" los recursos en la blockchain, por lo que los candidatos mineros no están obligados a invertir recursos físicos. En los sistemas PoB, **los mineros invierten en plataformas de minería virtual** (o potencia de minería virtual).

En otras palabras, al realizar quemas de monedas, los usuarios pueden demostrar su compromiso con la red, obtener el derecho a "minar" y validar las transacciones. Dado que el proceso de quemar monedas representa el poder de minería virtual, cuantas más monedas quema un usuario en favor del sistema, más poder de minería tiene, y por lo tanto mayores son las posibilidades de ser elegido como el siguiente validador de bloques.

En definitiva, el proceso de quemar monedas consiste en enviarlas a un público verificable donde se vuelven inaccesibles e inútiles. Normalmente, estas direcciones (también conocidas como *eater addresses*[2]) se generan de forma aleatoria sin tener ninguna clave privada asociada a ellas. Naturalmente, el proceso de quema de monedas reduce la disponibilidad del mercado y crea una escasez económica, causando

[2]Más información en `https://www.blockchain.com/btc/address/1BitcoinEaterAddressDontSendf59kuE`

un aumento potencial en su valor. Pero más que eso, la quema de monedas es otra forma de invertir en la seguridad de la red.

5.3 Proof-of-stake (PoS)

Proof-of-stake (PoS), o **prueba de participación**, es un tipo de algoritmo de consenso que depende de la participación económica de un validador en la red. **La importancia de este sistema se centra en la propiedad de la moneda en cuestión**. No requiere trabajo o poder computacional, más bien requiere poseer la criptomoneda.

Tener criptomonedas en la cartera digital es aquello que refleja la participación de los usuarios y **tienen más poder en la criptomoneda los usuarios que tengan en su cuenta más criptomonedas durante más tiempo**.

Mientras más criptomonedas posean los usuarios, más poder de minería poseen para validar los bloques de blockchain y por lo tanto más criptomonedas recibirán como recompensa. Es un sistema que no desperdicia tanta energía como PoW.

Sin embargo, si un usuario compromete sus criptomonedas como prueba de participación, entonces no las puede gastar en otras cosas, por lo tanto, un inconveniente de este sistema es que podría inhibir la comercialización con las criptomonedas minadas.

5.4 Proof-of-elapsed time (PoET)

En el caso del **proof-of-elapsed time (PoET)**, el algoritmo se basa en el principio de un sistema de lotería justo, donde cada nodo tiene la misma probabilidad de ganar. El mecanismo PoET se basa en distribuir las posibilidades de ganar de manera justa entre el mayor número de participantes de la red.

El funcionamiento del algoritmo PoET es el siguiente:

Algoritmo. Algoritmo PoET

1. Se requiere que cada nodo participante en la red espere un período de tiempo elegido al azar, y el primero en completar el tiempo de espera designado gana el nuevo bloque.
2. Cada nodo en la red blockchain genera un tiempo de espera aleatorio y se va a dormir durante esa duración especificada.
3. El nodo que tiene el menor tiempo de espera, se despierta y envía un nuevo bloque a la cadena de bloques, transmitiendo la información necesaria a toda la red de pares.
4. El mismo proceso se repite para el descubrimiento del siguiente bloque.

5.5 Proof-of-importance (PoI)

Proof-of-importance (PoI), o **prueba de importancia**, da prioridad a los mineros que tengan **mejor reputación en el sistema**. La reputación se mide por la cantidad de dinero invertido, el número de transacciones realizadas y la cantidad transferida en dichas transacciones. También se toma en cuenta para medirla la reputación de las cuentas con las que se realizan los intercambios. Este es el sistema que se utiliza para la criptomoneda NEM.

Por lo tanto, si María quisiera crear un bloque fraudulento para timar a Jorge, tendría que seguir con su mentira indefinidamente, algo que es casi imposible, ya que para ello debería equiparar su capacidad de cómputo a la del resto de mineros.

6 Controlando la fuente de dinero

Ahora bien, ¿de dónde viene el dinero en este tipo de sistemas? Tras la mayoría de las criptomonedas, la idea es que se genera más dinero conforme hay más trabajo, es decir, conforme hay más paquetes y más mineros. Este dinero se genera como recompensa al trabajo de los mineros, como hemos explicado antes.

Un caso particular es el de Bitcoin, en el que hay un límite superior de 21 millones de bitcoins. Todavía no se ha llegado a ese número, y se calcula que, al ritmo actual, llegaremos a él en el año 2140.

Este límite no se superará, sin embargo, debido a que la recompensa de los mineros va disminuyendo conforme hay más dinero en el sistema, llegando a ser una recompensa ausente.

Esta recompensa se calcula a partir de tres factores:

- **El dinero que ya hay en el sistema:** conforme nos acercamos a los 21 millones de bitcoins, la recompensa cada vez será menor.
- **El trabajo computacional de los mineros:** conforme más mineros trabajan, más complicado es llegar a la solución (véase el factor siguiente). Esto también se tiene en cuenta a la hora de calcular la recompensa.
- **La dificultad del mineo:** Bitcoin acepta bloques cada 10 minutos, y ajusta la dificultad (el número de ceros base) en función del número de mineros que haya, y del dinero que hay en la plataforma.

Una vez que lleguemos al límite, la importancia recaerá en las **tarifas de transacción**. Cada vez que un usuario realiza una transacción en Bitcoin, puede incluir una tarifa de transacción, que el minero obtendrá como recompensa, incentivando de esta forma el mineo. Además, debido a que el número de transacciones en un bloque de Bitcoin es limitado (realmente no se mide el número de transacciones, sino el "peso" de cada transacción)[3], los mineros insertarán en los bloques las transacciones más beneficiosas para ellos.

[3]Más información aquí: `https://bitcoinmagazine.com/guides/what-is-the-bitcoin-block-size-limit`

II | Demo: Bitcoin Core

Para ejemplificar cuál es la estructura de los bloques de la blockchain de una criptomoneda, así como el protocolo de comunicación de red que se utiliza, usaremos la aplicación **Bitcoin Core**, la implementación oficial de Bitcoin, que puede ser usada como cartera.

1 Procedimiento seguido

Sencillamente, descargaremos **Bitcoin Core** de la página web oficial de Bitcoin: `https://bitcoin.org/en/bitcoin-core/`, y seguiremos las instrucciones que aparecen en el archivo descargado para ejecutar el cliente `bitcoin-qt`. Como es la primera vez que ejecutamos `bitcoin-qt`, se conectará con la red Bitcoin y se descargará todos los bloques de la blockchain: ¡más de 200GB! Pero no te preocupes, pararemos la descarga para poder inspeccionar algunos bloques. Además, durante esta descarga ejecutaremos **Wireshark** para escuchar los mensajes intercambiados en Bitcoin.

Los datos descargados, por defecto, serán guardados en

```
~/.bitcoin
```

Analizaremos por tanto el resultado de lo descrito anteriormente desde dos perspectivas:

- **Los bloques:** analizaremos los bloques descargados por `bitcoin-qt` y veremos cómo los componentes de una criptomoneda explicados anteriormente están en éstos.
- **Los mensajes:** veremos cómo funciona el protocolo de Bitcoin analizando los mensajes enviados y recibidos en Wireshark.

2 Análisis de bloques

Entraremos en el directorio `.bitcoin`. Vemos que tiene un subdirectorio llamado `blocks`.

```
$> pwd
~/.bitcoin
$> ls
banlist.dat  chainstate/  fee_estimates.dat  peers.dat
blocks/      debug.log    mempool.dat        wallets/
$> cd blocks; ls
blk00000.dat  blk00003.dat  index/        rev00002.dat  rev00005.dat
blk00001.dat  blk00004.dat  rev00000.dat  rev00003.dat
blk00002.dat  blk00005.dat  rev00001.dat  rev00004.dat
```

Cada archivo `blk00*.dat` es una colección de bloques de cierto tamaño. Analizaremos uno de ellos,

por ejemplo, `blk00003.dat`.

Cada bloque sigue un formato especificado por Bitcoin:

Campo	Descripción	Tamaño
Número mágico	El valor siempre es `0xD9B4BEF9`	4 bytes
Tamaño de bloque	Longitud del bloque en bytes	4 bytes
Cabecera de bloque	6 elementos	80 bytes
Contador de transacciones	Entero positivo (`var_int`)	1-9 bytes
Transacciones	La lista de transacciones (no vacía)	Variable

Veamos con más detalle cada uno de ellos.

2.1 Número mágico

El primer elemento es una secuencia de 4 bytes fija, el **número mágico**, cuyo valor siempre es `0xD9B4BEF9`. Como el protocolo de Bitcoin usa *little endian*, la lectura del archivo binario resultará en la secuencia de bytes `0xF9 0xBE 0xB4 0xD9`. Haciendo `hexdump`:

```
$> hexdump -C -n 32 blk00003.dat
00000000  f9 be b4 d9 c2 bf 00 00  01 00 00 00 47 0c 5f a9
00000010  f8 d7 29 d0 ec c7 04 d0  b7 d1 23 31 c5 f4 15 bd
00000020
```

2.2 Tamaño de bloque

El bloque es seguido por cuatro bytes, que especifican el tamaño del bloque en bytes. En nuestro caso, convertir de *little endian* `0xC2BF0000` nos da `0x0000BFC2`, que convertido a decimal son 49090 bytes.

```
$> hexdump -C -n 32 blk00003.dat
00000000  f9 be b4 d9 c2 bf 00 00  01 00 00 00 47 0c 5f a9
00000010  f8 d7 29 d0 ec c7 04 d0  b7 d1 23 31 c5 f4 15 bd
00000020
```

2.3 Cabecera de bloque

Los siguientes 80 bytes son la cabecera, que se compone a su vez de los siguientes bloques:

Campo	Descripción	Actualizado cuando...	Tamaño
Versión	Número de versión del bloque	Actualizas el software y especifica una nueva versión	4 bytes
hashPrevBlock	Hash de 256 bits de la cabecera del bloque anterior	Llega un nuevo bloque	32 bytes
hashMerkleRoot	Hash de 256 bits de todas las transacciones del bloque	Se acepta una transacción	32 bytes
Tiempo	Tiempo actual en segundos desde 1970-01-01T00:00 UTC	Cada pocos segundos	4 bytes
Bits	*Target* actual, en formato compacto	Se ajusta la dificultad	4 bytes
Nonce	Número de 32 bits (desde 0)	Un hash es intentado (incrementa)	4 bytes

```
$> hexdump -C -n 80 -s8 blk00003.dat
00000008  01 00 00 00 47 0c 5f a9  f8 d7 29 d0 ec c7 04 d0
00000018  b7 d1 23 31 c5 f4 15 bd  47 e4 90 a7 00 03 00 00
00000028  00 00 00 00 10 f5 06 59  27 d8 ec 42 cf d8 04 54
00000038  e8 a7 76 85 fb 44 d6 a8  a0 d0 5a aa 62 38 1b 0a
00000048  37 78 6c 33 de eb 23 4e  cf bb 0a 1a e2 5b 6f 9a
00000058
```

De nuevo, todos los enteros son *little endian*. De este modo:

- **Versión** es `0x00000001`.
- **hashPrevBlock** es `0x000000000000300A790E447BD15F4C53123D1B7D007C7ECD029D7F8A95F0C`.
- **hashMerkleRoot** es `0x336C78370A1B3862AA5AD0A0A8D644FB8576A7E85404D8CF42ECD8275906F510`.
- **Tiempo** es `0x4E23EBDE`, en decimal, 1310976990. Sumando estos segundos al tiempo base, tenemos que el tiempo del bloque es el 18 de julio de 2011, a las 08:16:30 UTC.
- **Bits:** `0x1A0ABBCF` es un formato compacto del *target*, que es un número de 256 bits que todos los clientes de Bitcoin comparten. El hash SHA256 de la cabecera de un bloque debe ser menor o igual al *target* actual del bloque para ser aceptado por la red. Mientras menor sea el *target*, más complicado será generar un bloque. Es una especie especial de codificación de punto flotante, usando una mantisa de 3 bytes, el byte inicial es un exponente (donde sólo los 5 bits menos significativos se usan), y su base es 256. De este modo:
  - En este caso el exponente es `0x1A` = 26.
  - La mantisa es `0x0ABBCF`.

- Así que el exponente dice que este es un entero de 26 bytes y base 256. Para convertirlo a su valor entero, le añadimos 23 ceros hasta obtener: `0a bb cf 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`.
- Este número, convertido a decimal, es:

$$0x0a * 256^{26} + 0xbb * 256^{25} + 0xcf * 256^{24} \approx 4.4155582e63$$

- **Nonce** es `0x9A6F5BE2`, que es el número que es incrementado/cambiado en el mining para crear diferentes cabeceras de bloques, y de ese modo diferentes hashes.

2.4 Contador de transacciones

Los siguientes 1 a 9 bytes son un contador de transacciones de longitud variable. Para decodificarlo, recurrimos a la documentación de Bitcoin:

Valor	Tamaño	Formato
< `0xFD`	1 byte	`uint8_t`
≤ `0xFFFF`	3 bytes	`0xFD` seguido del tamaño como `unit16_t`
≤ `0xFFFFFFFF`	5 bytes	`0xFE` seguido del tamaño como `unit32_t`
	9 bytes	`0xFD` seguido del tamaño como `unit16_t`

Veamos esta sección en nuestro bloque:

```
$> hexdump -C -n 9 -s88 blk00003.dat
00000058  9a 01 00 00 00 01 00 00  00
00000061
```

Siguiendo el método descrito en la tabla, como el primer byte es menor que `0xFD`, el tamaño de almacenamiento de este entero es 1 byte, y el valor lo representa el primer byte, `0x9A`, 154 en decimal.

2.5 Transacciones

Basados en la decodificación anterior, sabemos que este bloque contiene 154 transacciones. Veamos la primera para entender la información que nuestros bloques almacenan. Hemos analizado ya 89 bytes del bloque, así que veremos qué es lo que le sigue.

```
$> hexdump -C -n 135 -s89 blk00003.dat
00000059  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00
00000069  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000079  00 00 00 00 00 ff ff ff  ff 08 04 cf bb 0a 1a 02
00000089  c0 02 ff ff ff ff 01 60  a1 5f 2a 01 00 00 00 43
00000099  41 04 c3 bb 8e 00 3d 61  7f 17 b9 ea 0f 2d 0d 62
000000a9  6a f6 63 34 22 11 bd 85  4f 61 c9 81 8b 87 ec d6
000000b9  c6 1c 2e 4c da 44 8d 91  14 7c 9d 24 d1 0d c7 a7
000000c9  2e 7e 7a b0 be ba 4d 44  d3 fa 95 09 80 7c b9 9a
000000d9  a3 2c ac 00 00 00 00
000000e0
```

Esto nos muestra la primera transacción de las 154. La primera transacción de un bloque siempre es especial: es en la que el minero se paga una recompensa por minar el bloque correctamente. En julio de 2011, la recompensa era de 50 bitcoins. En la fecha en la que se editó este documento, la recompensa era de 12.5 BTC. El próximo halving será el 23 mayo 2020 a las 06:29:48, que modificará la recompensa por bloque a los 6.25 BTC.

El formato general de una transacción Bitcoin es:

Campo	Descripción	Tamaño
Versión	Actualmente 1	4 bytes
In-counter	Entero positivo `var_int`	1-9 bytes
Lista de inputs	El primer input de la primera transacción también se llama *coinbase* (su contenido era ignorado en primeras versiones)	in-counter inputs
Out-counter	Entero positivo `var_int`	1-9 bytes
Lista de outputs	Las output de la primera transacción gastan los bitcoins minados para el bloque	out-counter outputs
lock_time	Si no es cero y los números de la secuencia son < `0xFFFFFFFF`: altura de bloque o timestamp si la transacción es definitiva	4 bytes

Podemos ver cada uno de los campos destacados en el `hexdump`. Ahora sólo nos queda entender los inputs y outputs.

2.5.1 Inputs

Cada input tiene el siguiente formato:

Campo	Descripción	Tamaño
Hash transacción previa	SHA256 doble de una transacción anterior	32 bytes
Índice Txout anterior	Entero no negativo indicando una salida de la transacción a ser usada	4 bytes
Longitud de script Txin	Entero no negativo `var_int`	1-9 bytes
Script Txin / scriptSig	**Script**: contiene información de la transacción	longitud de script bytes
sequence_no	Normalmente `0xFFFFFFFF`, irrelevante a menos que el lock_time de la transacción sea positivo	4 bytes

En nuestro caso, como estamos en una transacción *coinbase*, no hay transacción previa, por eso son todos ceros.

```
$> hexdump -C -n 135 -s89 blk00003.dat
00000059  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00
00000069  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000079  00 00 00 00 00 ff ff ff  ff 08 04 cf bb 0a 1a 02
00000089  c0 02 ff ff ff ff 01 60  a1 5f 2a 01 00 00 00 43
00000099  41 04 c3 bb 8e 00 3d 61  7f 17 b9 ea 0f 2d 0d 62
000000a9  6a f6 63 34 22 11 bd 85  4f 61 c9 81 8b 87 ec d6
000000b9  c6 1c 2e 4c da 44 8d 91  14 7c 9d 24 d1 0d c7 a7
000000c9  2e 7e 7a b0 be ba 4d 44  d3 fa 95 09 80 7c b9 9a
000000d9  a3 2c ac 00 00 00 00
000000e0
```

Para las transacciones *coinbase*, el scriptSig se ignora: su valor es irrelevante.

El valor `0x01`, subrayado, indica que hay un output en nuestra transacción.

2.5.2 Outputs

Cada output de la lista de outputs sigue el siguiente formato:

Campo	Descripción	Tamaño
Valor	Entero no negativo, especifica el número de Satoshis que serán transferidos (1 Satoshi = 10^{-8} BTC)	8 bytes
Longitud de script Txout	Entero no negativo `var_int`	1-9 bytes
Script Txout / scriptPubKey	**Script**: contiene información de la transacción	longitud de script bytes

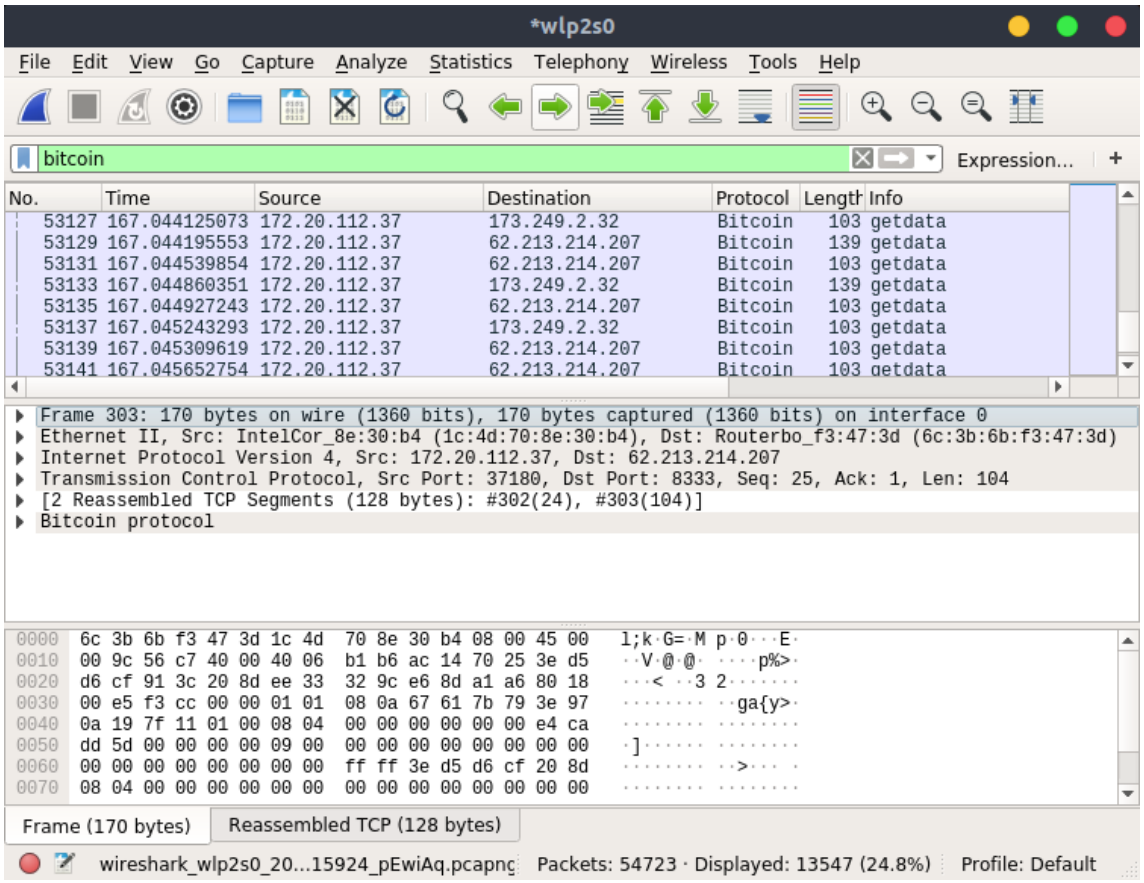
Veamos qué cantidad ha ganado el minero:

```
$> hexdump -C -n 135 -s89 blk00003.dat
00000059  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00
00000069  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000079  00 00 00 00 00 ff ff ff  ff 08 04 cf bb 0a 1a 02
00000089  c0 02 ff ff ff ff 01 60  a1 5f 2a 01 00 00 00 43
00000099  41 04 c3 bb 8e 00 3d 61  7f 17 b9 ea 0f 2d 0d 62
000000a9  6a f6 63 34 22 11 bd 85  4f 61 c9 81 8b 87 ec d6
000000b9  c6 1c 2e 4c da 44 8d 91  14 7c 9d 24 d1 0d c7 a7
000000c9  2e 7e 7a b0 be ba 4d 44  d3 fa 95 09 80 7c b9 9a
000000d9  a3 2c ac 00 00 00 00
000000e0
```

Podemos ver que el minero ha ganado `0x000000012A5FA160`, es decir, 5005877600 Satoshis, un total de 50.05877600 BTC.

3 Análisis de mensajes

Una vez que nos conectamos con Wireshark durante la descarga de los paquetes, vemos cómo aparecen mensajes en la ventana de Wireshark.

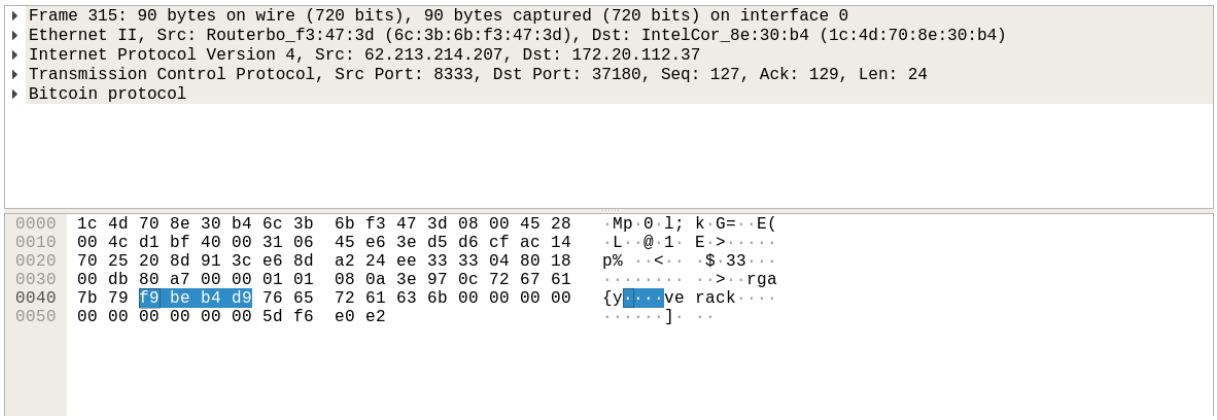


3.1 Estructura general

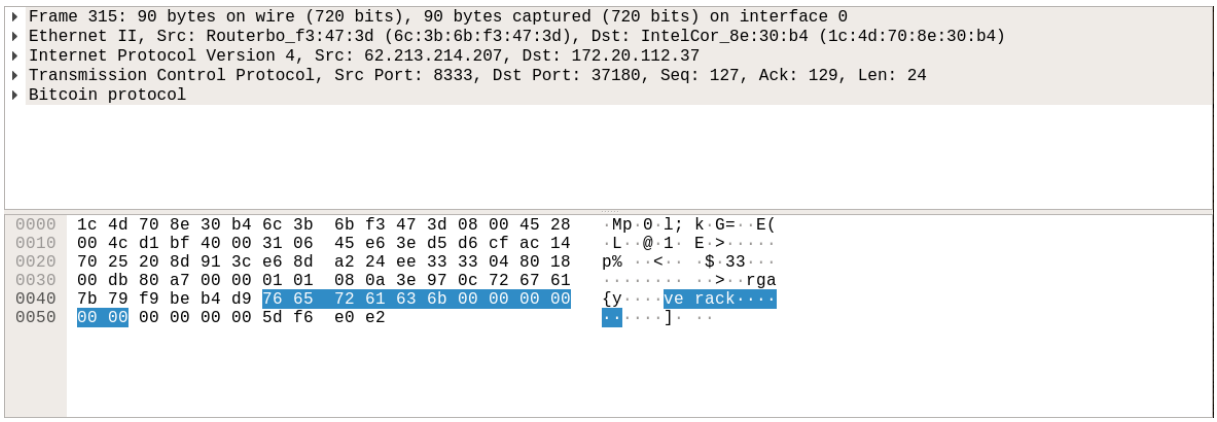
Los nodos de Bitcoin se conectan entre ellos por TCP. Normalmente, los nodos buscan mensajes en el puerto 8333, aunque esto puede ser modificado. Cada mensaje de la red tiene una estructura definida.

Descripción	Tipo de dato	Comentarios	Tamaño
magic	`uint32_t`	Valor mágico indicando la red de origen del mensaje, y usado para buscar el estado del siguiente mensaje cuando el estado del stream no se conoce.	4 bytes
command	`char[12]`	String ASCII identificando el contenido del paquete	12 bytes
length	`uint32_t`	Longitud del payload en número de bytes	4 bytes
checksum	`unit32_t`	Primeros 4 bytes de *sha*256(*sha*256(*payload*))	4 bytes
payload	`uchar[]`	La información que se quiere transmitir	

El número mágico de 4 bytes para la red Bitcoin es `0xD9B4BEF9`, aquí lo vemos en *little endian*:



Tras el número mágico, aparece un conjunto de 12 bytes, que describen el comando que es enviado en el mensaje, usualmente un texto ASCII. Por ejemplo, en la captura anterior vemos que se trata de un mensaje `verack`. En la documentación del protocolo Bitcoin podemos ver que se trata del mensaje enviado como respuesta a `version`.



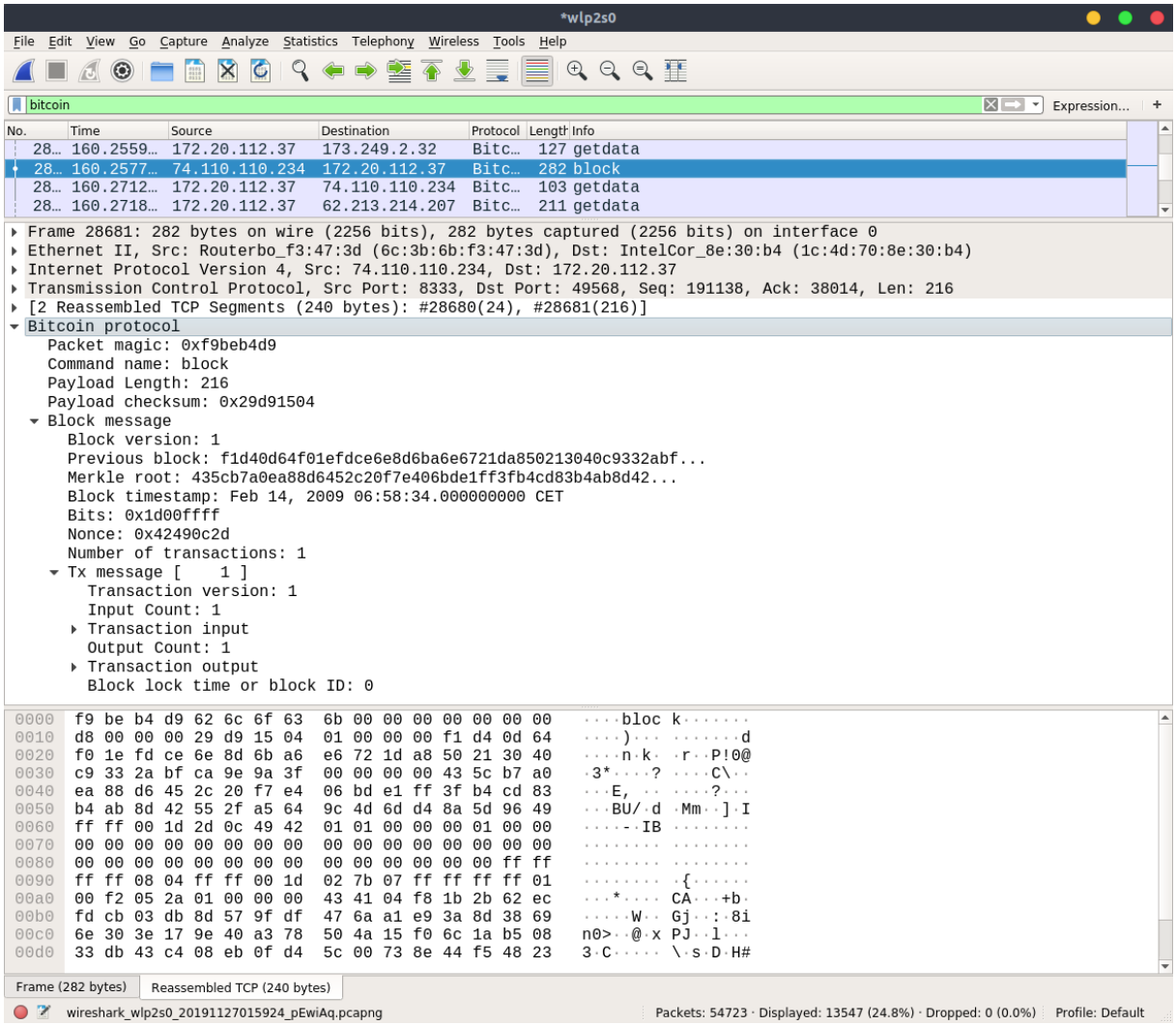
3.2 Analizando un mensaje: `block`

Una vez que hemos visto la estructura general de un mensaje en Bitcoin, veamos con más detenimiento un mensaje, esta vez de tipo `block`.

Estos mensajes se generan como respuesta a un mensaje `getdata`, que requiere información de una transacción a partir del hash de un bloque.

Buscaremos un mensaje `block` en Wireshark, uno de ellos es el mensaje interceptado número 28681. Nótese en la captura siguiente que el mensaje anterior, el 28680, es un `getdata`.

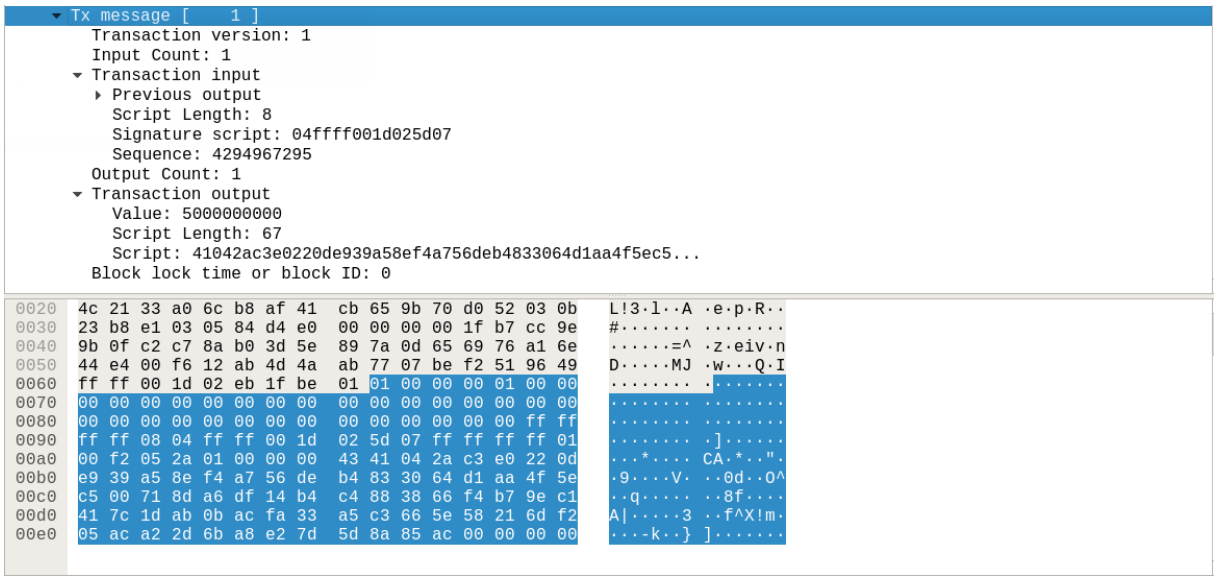
Del mismo modo, podemos apreciar cómo todos los campos especificados anteriormente se encuentran en el mensaje (número mágico, comando, longitud, etc.).



Centrémonos, sin embargo, en el *payload*, en la información que el mensaje pretende enviar. ¡Tenemos la cabecera del bloque que habíamos visto antes! Simplemente, añadimos dos campos más:

Descripción	Tipo de dato	Comentarios	Tamaño
txn_count	`var_int`	Número de transacciones	1+ bytes
txns	`tx[]`	Bloques de transacción, en el formato de un comando "tx"	

Vemos que estos campos, en efecto, contienen información de transacciones, de acuerdo a la codificación explicada en la sección anterior:



Analizando el mensaje, vemos que la transacción era por valor de 5000000000 Satoshis, es decir, 50 BTC. Además, vemos que esta transacción no es resultado de un *mining*, pues el input no es todo cero.

III | Bibliografía

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*.
https://bitcoin.org/bitcoin.pdf

[2] D. Selmanovic. *Bitcoin and cryptocurrency algorithms implementation tutorial*, Toptal Developers.
https://www.toptal.com/bitcoin/cryptocurrency-for-dummies-bitcoin-and-beyond

[3] 3Blue1Brown, *But how does Bitcoin actually work?*.
https://www.youtube.com/watch?v=bBC-nXj3Ng4

[4] Bitcoin Core.
https://bitcoincore.org/

[5] Bitcoin Protocol Documentation.
https://en.bitcoin.it/wiki/Protocol_documentation

[6] Binance Academy, *Proof of Burn explicada*.
https://www.binance.vision/es/blockchain/proof-of-burn-explained

[7] HackerNoon, *Proof of Work, Proof of Burn and Proof of Stake*.
https://hackernoon.com/proof-of-work-proof-of-stake-and-proof-of-burn-6823eac2776e

Nota. Los enlaces que aparecen aquí fueron actualizados a fecha de 27 de noviembre de 2019. Es posible que sufran algún tipo de modificación, incluso llegando a no estar disponibles.