UNIVERSIDAD DE GRANADA

Creando una criptomoneda

Fundamentos de Redes. Seminarios

Miguel Ángel Fernández Gutiérrez
Pedro Gallego López

Doble Grado en Ingeniería Informática y Matemáticas

Índice

¿Qué es una criptomoneda?

¿Qué queremos?

¿Qué queremos?

Queremos crear un sistema **anónimo**, **descentralizado** y **seguro**.

- **Anónimo:** usuarios identificables, pero no su verdadera identidad.
- **Descentralizado:** eliminar instituciones centrales.
- **Seguro:** no permitiremos transacciones fraudulentas.

¿Qué queremos?

Para que nuestro sistema tenga todo esto, deberá cumplir 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.

¿Qué queremos?

Nuestro sistema 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

1. Descentralización: redes P2P

- Solución para descentralización: **redes P2P**.
- Todos los ordenadores tienen el mismo peso, información compartida de igual a igual, al contrario que cliente-servidor.

1. Descentralización: redes P2P. Características

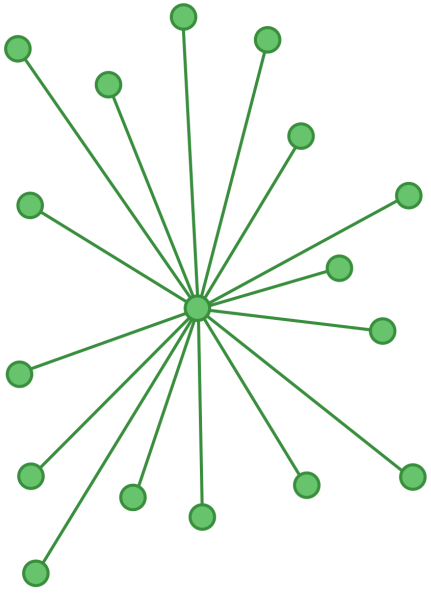
- Cada nodo actúa simultáneamente como cliente y como servidor.
- Permiten intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.

1. Descentralización: redes P2P. Características

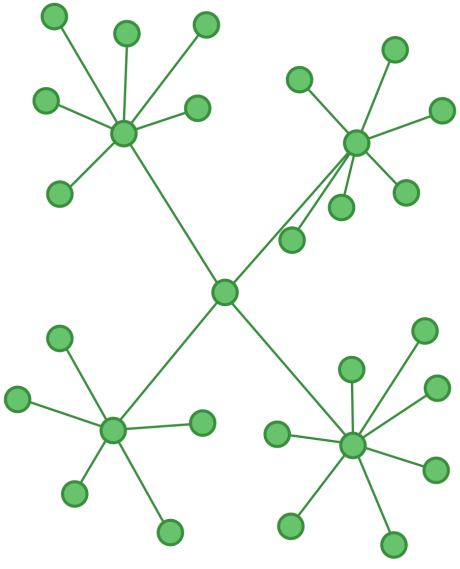
Rasgos a destacar:

- **Escalabilidad:** mientras más nodos estén conectados, mejor será el funcionamiento.
- **Descentralización:** ningún nodo es imprescindible para el funcionamiento de la red.
- **Distribución de costes entre los usuarios.**
- **Robustez:** menos puntos singulares de fallas en el sistema.
- **Anonimato.**
- **Seguridad:** investigar nodos maliciosos. Característica deseable, poco implementada. Algunas soluciones prometedoras: *cifrado multiclave, cajas de arena*...

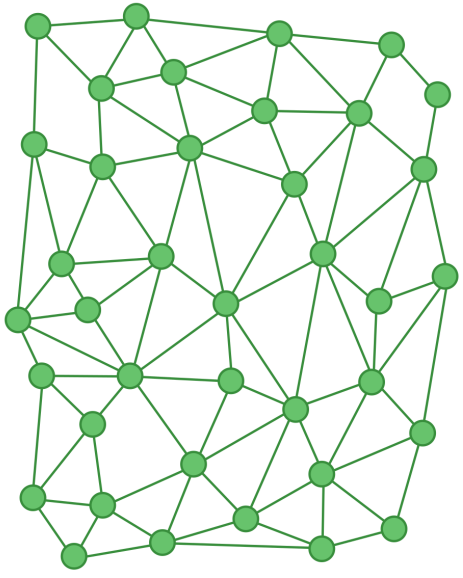
1. Descentralización: redes P2P. Topologías



centralizada



descentralizada
y estructurada



descentralizada
y no estructurada

1. Descentralización: redes P2P. Ventajas e inconvenientes

Ventajas

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

Inconvenientes

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

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. Firmas digitales, criptografía y hashing. Funciones hash

2. Firmas digitales, criptografía y hashing. Funciones hash

Una buena función hash debe garantizar:

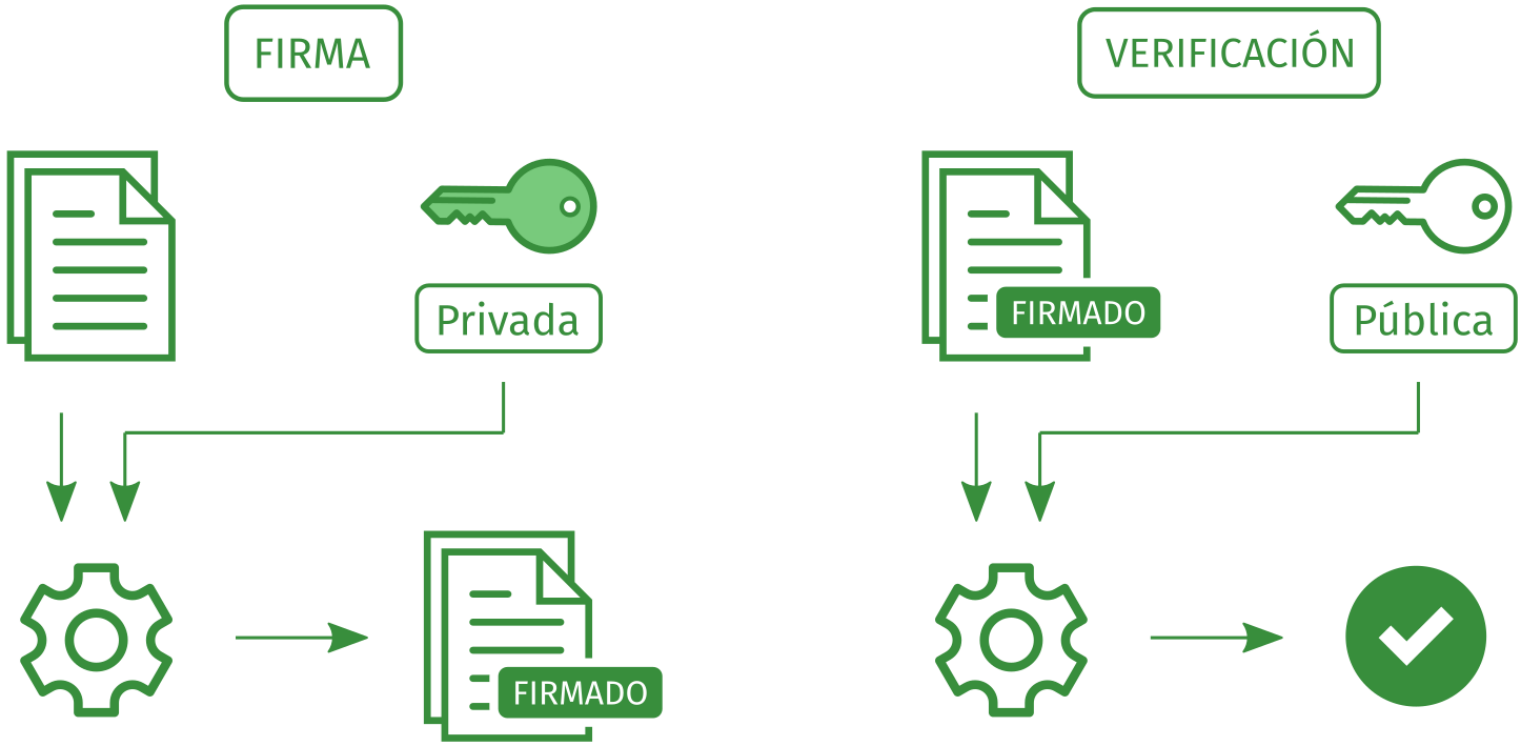
1. La salida de la función hash debe tener un tamaño fijo.
2. Un mínimo cambio en la entrada debe producir un enorme cambio en la salida.
3. Una misma entrada siempre producirá la misma salida.
4. No debe haber forma alguna de revertir el cambio, es decir, de que a partir de la salida se pueda encontrar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

2. Firmas digitales, criptografía y hashing. Firmas digitales

Las funciones hash las usan algoritmos criptográficos, que utilizaremos para garantizar que nadie diga nada en tu nombre: **firmas digitales**.

Usaremos el método de **criptografía asimétrica** (clave privada-clave pública).

2. Firmas digitales, criptografía y hashing. Firmas digitales



2. Firmas digitales, criptografía y hashing. Firmas digitales

Firma

Firmar(**Mensaje**, **Clave Privada**) = **Firma**

2. Firmas digitales, criptografía y hashing. Firmas digitales

Verificación

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

3. Enviando transacciones a la red

- Nos queda enviar información al sistema.
- No tenemos autoridad central que valide cuánto dinero tenemos, pero no hace falta: **la contabilidad es la propia moneda** (nos basta tener una lista de transacciones, *ledger*).

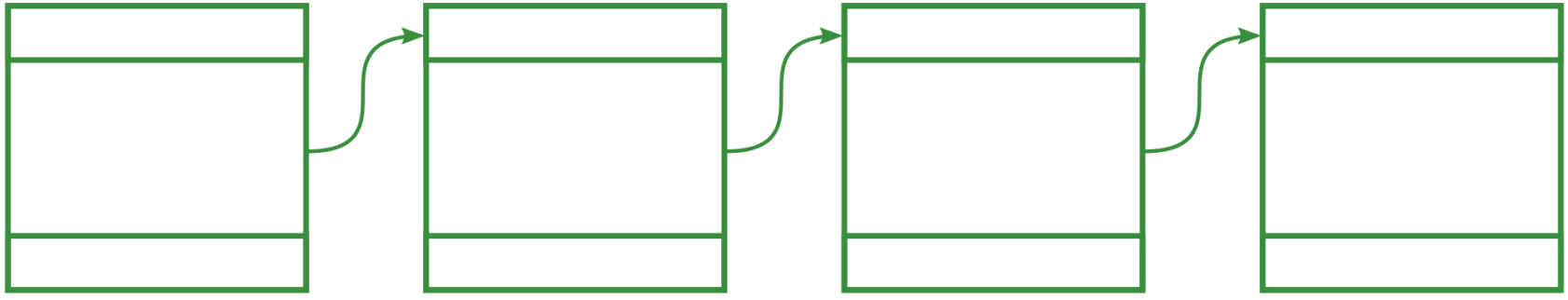
3. Enviando transacciones a la red

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).

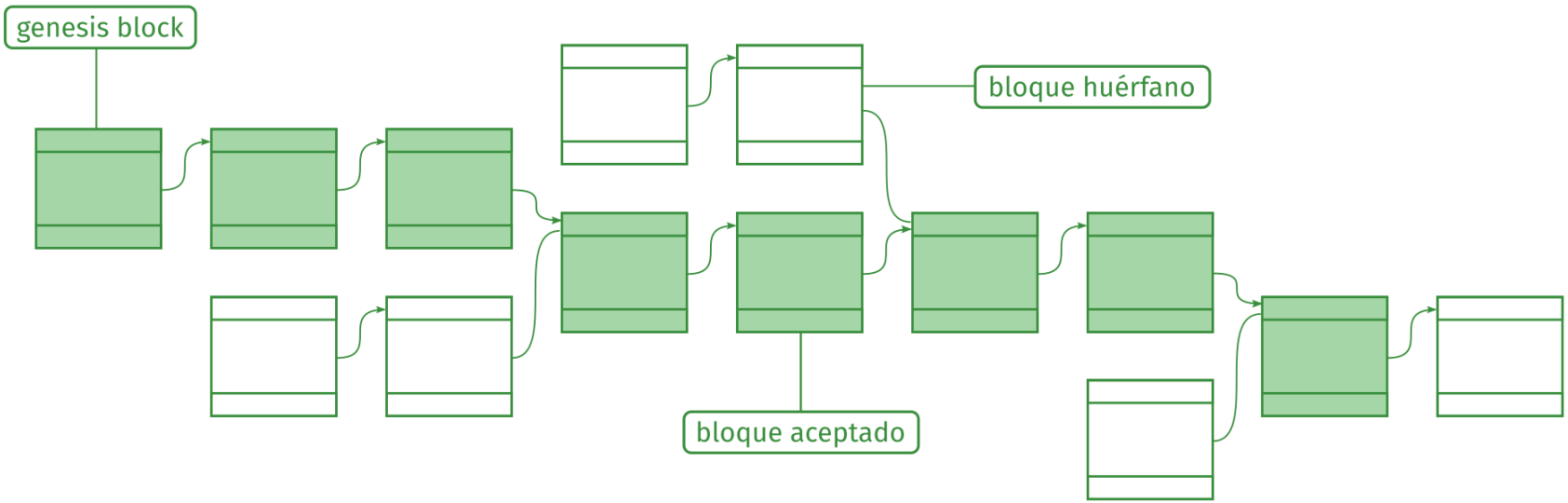
4. El blockchain

Necesitamos mantener un historial de las transacciones realizadas: **blockchain**.



- *Public ledger.*
- Enlazadas mediante hash del anterior: **seguridad**.

4. El blockchain. Los bloques



5. Mecanismos de verificación de transacciones

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

- Evitar bloques fraudulentos.
- El papel de los **mineros**.

5. Mecanismos de verificación de transacciones. PoW

Proof-of-work

- Premia a aquellos que realizan más trabajo de cómputo.
- Usado en Bitcoin: el minero premiado es el que consiga que el hash del bloque comience en el mayor número de ceros.

5. Mecanismos de verificación de transacciones. PoB

Proof-of-burn

- Criptomonedas se queman intencionadamente como una forma de “invertir” los recursos en la blockchain.
- El proceso de quema reduce la disponibilidad del mercado y aumenta el valor de la criptomoneda.
- Cuantas más monedas quema un usuario en favor del sistema, más poder de minería tiene.

5. Mecanismos de verificación de transacciones. PoS

Proof-of-stake

- Sistema centrado en propiedad de la moneda: más poder quien tenga en su cuenta más criptomonedas durante más tiempo.
- Un usuario compromete las criptomonedas como prueba de participación.

5. Mecanismos de verificación de transacciones. PoET

Proof-of-elapsed time

- Distribuir las posibilidades de ganar de manera justa entre el mayor número de participantes en la red.

5. Mecanismos de verificación de transacciones. PoET

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. Mecanismos de verificación de transacciones. PoI

Proof-of-importance

- Da prioridad a los mineros con mayor **reputación** en el sistema.
- La reputación se mide por:
  - Cantidad de dinero invertido.
  - Número de transacciones realizadas.
  - Cantidad transferida en dichas transacciones.

6. Controlando la fuente de dinero

¿De dónde viene el dinero?

6. Controlando la fuente de dinero

En el caso de Bitcoin, hay un límite superior de **21M BTC**, y el dinero incrementa con las **recompensas de los mineros**.

Esta recompensa se calcula a partir de tres factores:

- Dinero que hay en el sistema.
- Trabajo computacional de los mineros.
- Dificultad del mineo.

Además, hemos de incluir en el juego las **tarifas de transacción**.

Demo: Bitcoin Core

Procedimiento seguido

1. Descargamos **Bitcoin Core** de la página web oficial.
2. Iniciamos **Wireshark**.
3. Ejecutamos el cliente `bitcoin-qt`.
    - Se descargarán todos los bloques de Bitcoin en `~/.bitcoin`.
    - ¡Más de 200GB!

Procedimiento seguido

```
$> 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
```

Formato del bloque

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

Formato del bloque. Número mágico

Siempre es `0xD9B4BEF9`

```
$> 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
```

Formato del bloque. Tamaño de bloque

0x0000BFC2 ⟶ 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
```

Formato del bloque. Cabecera del bloque

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

Formato del bloque. Cabecera del bloque

```
$> 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
```

Formato del bloque. Cabecera del bloque

- **Versión:** `0x00000001`.
- **hashPrevBlock:** `0x000000000000300A790E447BD`...
- **hashMerkleRoot:** `0x336C78370A1B3862AA5AD0A0A8`...
- **Tiempo:** `0x4E23EBDE` $\longrightarrow$ 1310976990
  $\longrightarrow$ 18 de julio de 2011, a las 08:16:30 UTC.
- **Bits:** `0x1A0ABBCF`, codificación de punto flotante.
  - El exponente es `0x1A` = 26.
  - La mantisa es `0x0ABBCF`.
  - Exponente dice que este es un entero de 26 bytes y base 256. 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 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**: `0x9A6F5BE2`.

Formato del bloque. Contador de transacciones

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`

Formato del bloque. Contador de transacciones

0x9A ⟶ 154 transacciones

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

Formato del bloque. Transacciones

Primera transacción del bloque: **recompensa del 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
```

Formato del bloque. Transacciones

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

Formato del bloque. Transacciones

50 BTC, julio 2011

↓

12.5 BTC, hoy

↓ *halving*

6.25 BTC

23 mayo 2020 06:29:48

Formato del bloque. Transacciones

Input

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

Formato del bloque. Transacciones

Transacción *coinbase*

```
$> 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
```

Formato del bloque. Transacciones

Output

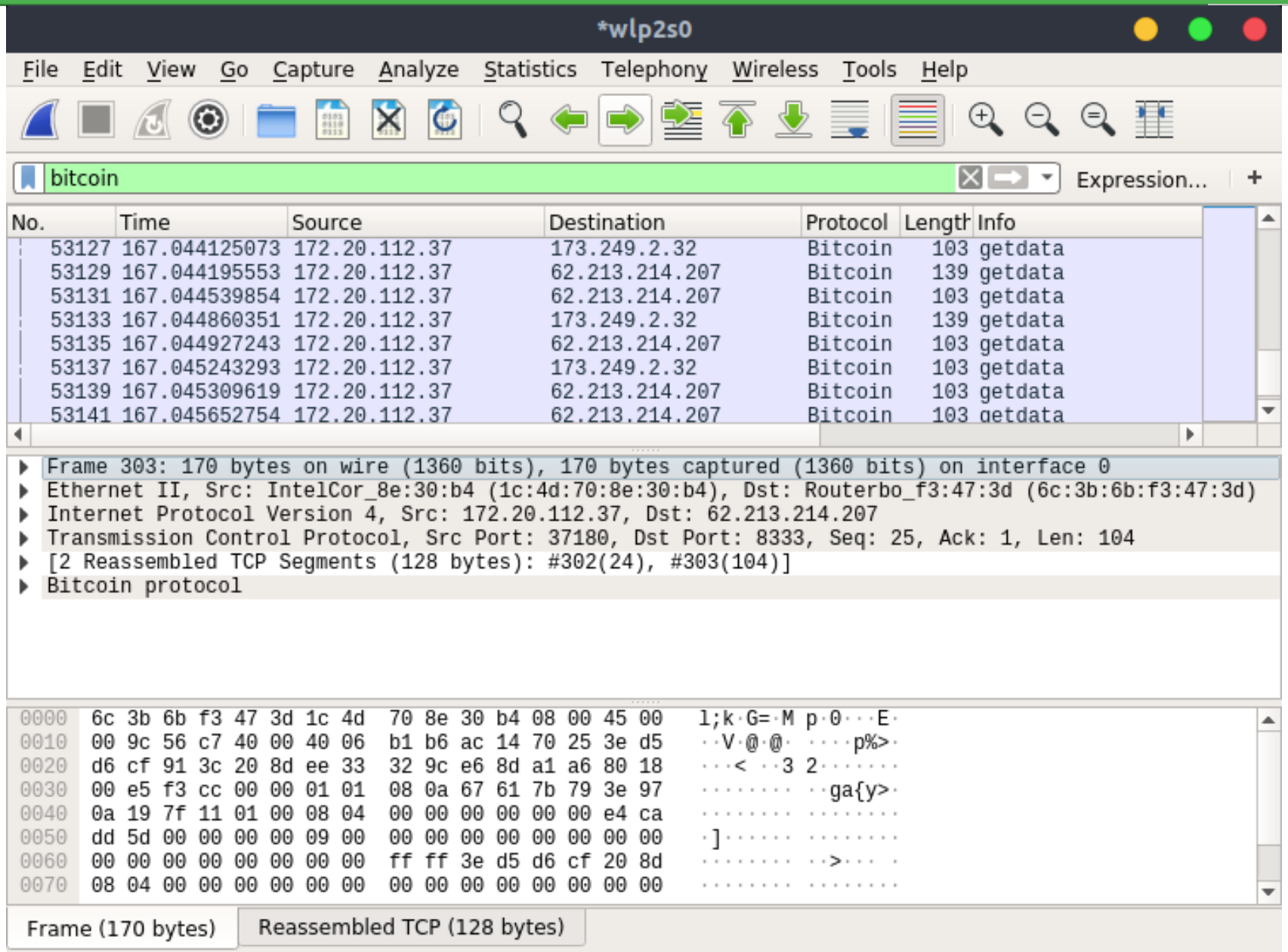
Campo	Descripción	Tamaño
Valor	Entero no negativo, especifica el número de Satoshis que serán transferidos (1 Satoshi = 10^{-8} BTC)	8 bytes
Longitud de script Txout	Entero no negativo `var_int`	1-9 bytes
Script Txout / scriptPubKey	**Script**: contiene información de la transacción	longitud de script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

Formato del bloque. Transacciones

0x000000012A5FA160 ⟶ 5005877600 Satoshis ⟶ 50.058776 BTC

```
$> 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
```

Análisis de mensajes



Análisis de mensajes. Estructura general

Nodos Bitcoin se conectan entre sí por TCP, buscando normalmente en el puerto 8333

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	

Análisis de mensajes. Estructura general

Número mágico, en *little endian*

```
▸ Frame 315: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
▸ Ethernet II, Src: Routerbo_f3:47:3d (6c:3b:6b:f3:47:3d), Dst: IntelCor_8e:30:b4 (1c:4d:70:8e:30:b4)
▸ Internet Protocol Version 4, Src: 62.213.214.207, Dst: 172.20.112.37
▸ Transmission Control Protocol, Src Port: 8333, Dst Port: 37180, Seq: 127, Ack: 129, Len: 24
▸ Bitcoin protocol
```

```
0000  1c 4d 70 8e 30 b4 6c 3b  6b f3 47 3d 08 00 45 28   ·Mp·0·l; k·G=··E(
0010  00 4c d1 bf 40 00 31 06  45 e6 3e d5 d6 cf ac 14   ·L··@·1· E·>·····
0020  70 25 20 8d 91 3c e6 8d  a2 24 ee 33 33 04 80 18   p% ··<·· ·$·33···
0030  00 db 80 a7 00 00 01 01  08 0a 3e 97 0c 72 67 61   ·····••· ··>··rga
0040  7b 79 f9 be b4 d9 76 65  72 61 63 6b 00 00 00 00   {y····ve rack····
0050  00 00 00 00 00 00 5d f6  e0 e2                     ·····]· ··
```

Análisis de mensajes. Estructura general

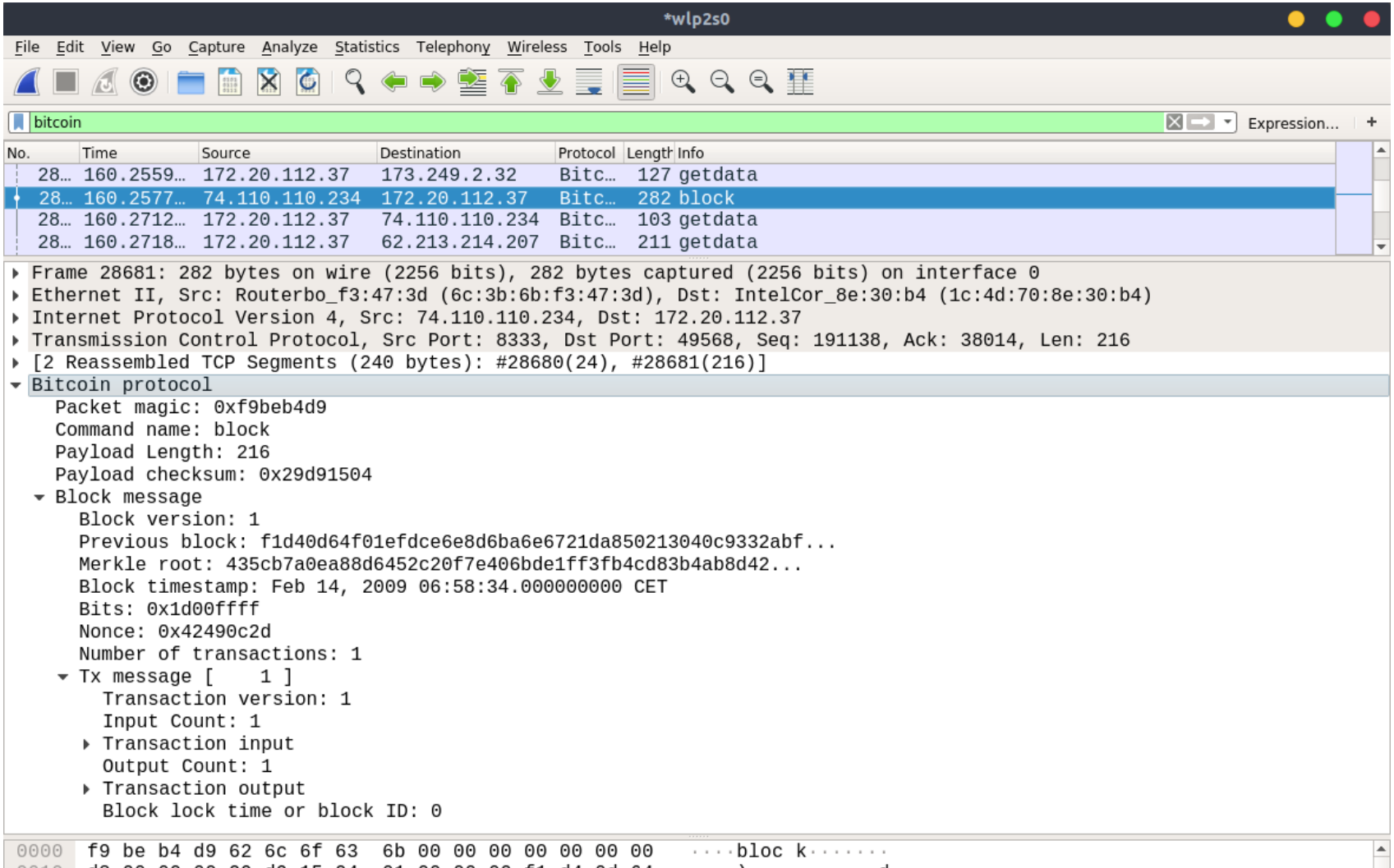
Mensaje `verack`, en respuesta a `version`

```
▸ Frame 315: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
▸ Ethernet II, Src: Routerbo_f3:47:3d (6c:3b:6b:f3:47:3d), Dst: IntelCor_8e:30:b4 (1c:4d:70:8e:30:b4)
▸ Internet Protocol Version 4, Src: 62.213.214.207, Dst: 172.20.112.37
▸ Transmission Control Protocol, Src Port: 8333, Dst Port: 37180, Seq: 127, Ack: 129, Len: 24
▸ Bitcoin protocol
```

```
0000  1c 4d 70 8e 30 b4 6c 3b  6b f3 47 3d 08 00 45 28   ·Mp·0·l; k·G=··E(
0010  00 4c d1 bf 40 00 31 06  45 e6 3e d5 d6 cf ac 14   ·L··@·1· E·>·····
0020  70 25 20 8d 91 3c e6 8d  a2 24 ee 33 33 04 80 18   p% ··<·· ·$·33···
0030  00 db 80 a7 00 00 01 01  08 0a 3e 97 0c 72 67 61   ·····••· ··>··rga
0040  7b 79 f9 be b4 d9 76 65  72 61 63 6b 00 00 00 00   {y····ve rack····
0050  00 00 00 00 00 00 5d f6  e0 e2                     ······]· ··
```

Análisis de mensajes. Analizando un mensaje: `block`

Mensaje `block`, en respuesta a `getdata`



Análisis de mensajes. Analizando un mensaje: `block`

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”	

Análisis de mensajes. Analizando un mensaje: `block`

5000000000 Satoshis ⟶ 50 BTC

```
▾ Tx message [    1 ]
    Transaction version: 1
    Input Count: 1
  ▾ Transaction input
    ▸ Previous output
      Script Length: 8
      Signature script: 04ffff001d025d07
      Sequence: 4294967295
    Output Count: 1
  ▾ Transaction output
      Value: 5000000000
      Script Length: 67
      Script: 41042ac3e0220de939a58ef4a756deb4833064d1aa4f5ec5...
    Block lock time or block ID: 0
```

```
0020  4c 21 33 a0 6c b8 af 41  cb 65 9b 70 d0 52 03 0b   L!3·l··A ·e·p·R··
0030  23 b8 e1 03 05 84 d4 e0  00 00 00 00 1f b7 cc 9e   #······ ········
0040  9b 0f c2 c7 8a b0 3d 5e  89 7a 0d 65 69 76 a1 6e   ·····=^ ·z·eiv·n
0050  44 e4 00 f6 12 ab 4d 4a  ab 77 07 be f2 51 96 49   D·····MJ ·w···Q·I
0060  ff ff 00 1d 02 eb 1f be  01 01 00 00 00 01 00 00   ········ ·.......
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff   ········ ········
0090  ff ff 08 04 ff ff 00 1d  02 5d 07 ff ff ff ff 01   ········ ·]......
00a0  00 f2 05 2a 01 00 00 00  43 41 04 2a c3 e0 22 0d   ···*···· CA·*··".
00b0  e9 39 a5 8e f4 a7 56 de  b4 83 30 64 d1 aa 4f 5e   ·9····V· ··0d··O^
00c0  c5 00 71 8d a6 df 14 b4  c4 88 38 66 f4 b7 9e c1   ··q····· ··8f····
00d0  41 7c 1d ab 0b ac fa 33  a5 c3 66 5e 58 21 6d f2   A|·····3 ··f^X!m·
00e0  05 ac a2 2d 6b a8 e2 7d  5d 8a 85 ac 00 00 00 00   ···-k··} ]·......
```

¡Gracias por vuestra atención!