Universidad de Granada

Doble Grado en Ingeniería Informática y Matemáticas

Fundamentos de Redes

La web distribuida: el protocolo IPFS

Autores:
José María Martín Luque
Adolfo Soto Werner

Profesor:
Antonio Ruiz Moya

12 de noviembre de 2017

Índice

Introducción

El Protocolo de transferencia de hipertexto (HTTP por sus siglas en inglés) es uno de los protocolos fundamentales de Internet. El desarrollo de HTTP comenzó en 1989 en el CERN por parte de Tim Berners-Lee. El desarrollo de un estándar fue un trabajo colaborativo entre el Grupo de Trabajo de Ingeniería de Internet (Internet Engineering Task Force, IETF) y el Consorcio WWW (World Wide Web Consortium, W3C), proceso que culminó con la publicación de una serie de *Request for Comments* (RFCs)[1]. La primera definición de `HTTP/1.1`, la versión de HTTP más utilizada, apareció en el RFC 2068 de 1997.

Estamos hablando por tanto de un protocolo cuyo diseño comenzó hace más de 25 años. En aquel momento era inimaginable pensar que la tecnología que se estaba desarrollando fuese a ser usada por miles de millones de personas (3.885.567.619 a nivel mundial según las últimas estadísticas[1] de junio de 2017) ni se esperaba que tuviese tal repercusión sobre nuestras vidas. HTTP ha simplificado y facilitado la transmisión de información a nivel mundial. Gracias a ello hemos avanzado hacia una sociedad conectada donde la información y la cultura fluye libremente.

Pero como es de esperar, un protocolo que no fue diseñado con la visión del mundo actual presenta una serie de problemas a resolver. La intención de este trabajo es mostrar las deficiencias de HTTP y explorar una alternativa a la web actual, la web distribuida y el protocolo IPFS.

[1]Los RFCs son un tipo de documentos técnicos del IETF que datallan técnicamente diversos aspectos del funcionamiento de Internet y otros protocolos.

1 Problemas de HTTP

1.1 Fragilidad

Figura 1: Primer servidor de HTTP del mundo. Se trata del ordenador personal de Tim Berners-Lee durante su estancia en el CERN.

Para entender por qué decimos que HTTP es *frágil* solo hay que observar la pegatina del primer servidor de HTTP: "*Esta máquina es un servidor. ¡¡No apagar!!*". Está ahí para recordarnos que si se apagaba el servidor no se podía acceder al contenido. Otros sitios web en distintos servidores enlazaban a su contenido, de forma que si dejaba de estar en la red, todos esos enlaces no servían para nada. Por otro lado, si ese servidor se movía a otra ubicación, con otra dirección, todos esos enlaces habían muerto.

Hablamos en pasado pero efectivamente este sigue siendo un problema en la actualidad. No es raro entrar en algún enlace y encotrarnos con el error que podemos ver en la figura **??**. Incluso si no conoces la especificación del protocolo

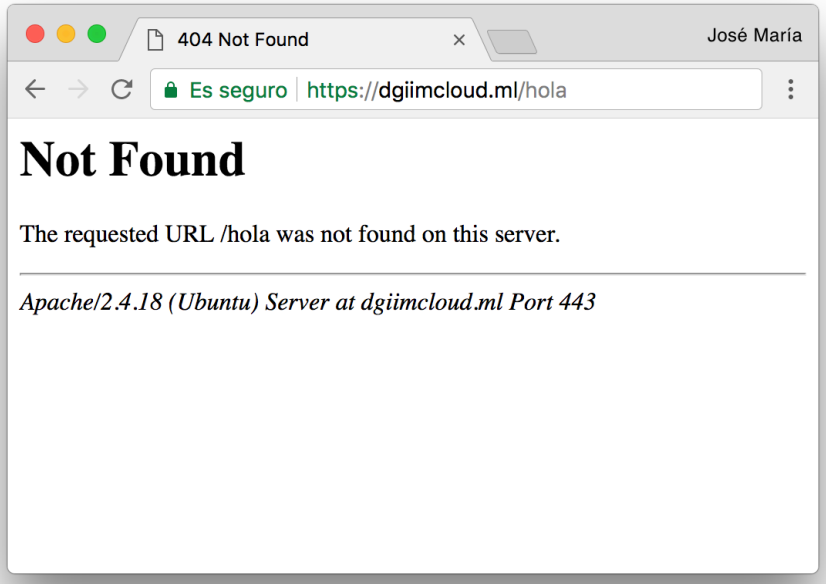

Figura 2: Error 404.

HTTP es probable que ya sepas que 404 es el código de error que nos indica que no hay nada que ver en una dirección.

La desaparición de enlaces (o *link rot* en inglés) es mucho más habitual de lo que se pueda pensar. El creador de Pinboard, Maciej Cegłowski, estima que alrededor del 5 % de los enlaces que almacenan los usuarios en este servicio dejan de funcionar cada año[2]. Añade además que uno de sus clientes ha visto cómo dejaban de funcionar el 90 % de los enlaces que llevaba almacenados desde 1997. En 2014, un estudio de Jonathan Zittrain, Kendra Albert y Lawrence Lessig de la Harvard Law School concluyó que aproximadamente el 50 % de los enlaces que aparecen en resoluciones del Tribunal Supremo de los Estados Unidos ya no redireccionan a la información original[3]. Por estos motivos existen servicios como The Internet Archive, que se dedica a guardar copias de las páginas web para preservarlas de cara al futuro, o el propio Pinboard, que ofrece a sus usuarios la posibilidad de almacenar copias de los artículos que añadan al servicio.

1.2 Hipercentralización

En la *Declaración de Independencia del Ciberespacio*[4] John Perry Barlow describía una utopía digital en la que los ciudadanos de la red se autogobiernan y las antiguas instituciones no tienen nada que hacer. "De parte del futuro, os pido a vosotros del pasado que nos dejéis en paz. No sois bienvenidos entre nosotros. No tenéis la soberanía del lugar donde nos reunimos."

Por desgracia, esta no es la realidad en el año 2017. En la actualidad la web está altamente centralizada. La práctica totalidad de los usuarios de Internet dependen de una serie de servicios concretos. Por poner algunos ejemlos, en 2015 Facebook anunció que más de mil millones de usuarios utilizaron el servicio en un mismo día[5], mientras que una caída del servicio de Google en 2013 provocó una reducción del tráfico de internet del 40 %[6].

El pasado 26 de octubre una inundación en la sala de servidores de la Universidad de Granada provocó que todos los servicios digitales dejasen de funcionar. Puede parecer algo leve pero gran parte de la actividad de la Universidad depdende de que funcione su infraestructura informática. Las secretarías dependen de la red, así como el servicio de comedores, las redes Wi-Fi (tienen que autenticar los usuarios en el servidor) y la Plataforma de Recursos de Apoyo a la Docencia (PRADO), entre otros.

La *hipercentralización* de la red trae otras consecuencias negativas. Organizaciones como la NSA sólo tienen que intervenir el tráfico de unas pocas empresas para espiarnos, tal y como revelan las filtraciones de Edward Snowden[7]. La censura es mucho más fácil de establecer ya que sólo hay que bloquear el acceso a una serie de sitios concretos.

1.3 Ineficiencia

Para comprobar la ineficiencia de transmitir información por HTTP vamos a poner un ejemplo. El vídeo más visto de YouTube según Wikipedia a 17 de octubre de 2017 es "Luis Fonsi - Despacito ft. Daddy Yankee"[8], con

4.055.733.709 visualizaciones a las 11:47.

Supongamos que el vídeo siempre se reprodujese en 720p, de tal forma que el archivo pesa exactamente 67,1MB. 4.055.733.709 visualizaciones de un archivo de 67,1MB son 272.139.731.874MB descargados. Suponiendo que a Google le costase 1 céntimo transmitir 1GB de información (incluyendo todos los gastos del servidor), ya se habría gastado más de 272.139.731,874€ en transmitir un único vídeo.

Este precio de 1 céntimo por GB quizás sea posible para Google pero no lo es ni mucho menos para el ciudadano medio. La tabla de precios del CDN de Amazon, CloudFront es la siguiente[9]:

	Estados Unidos	Europa	Japón	India
Primeros 10TB/mes	0,085USD	0,085USD	0,140USD	0,170USD
Siguientes 40TB/mes	0,080USD	0,080USD	0,135USD	0,130USD
Siguientes 100TB/mes	0,060USD	0,060USD	0,120USD	0,110USD
Siguientes 350TB/mes	0,040USD	0,040USD	0,100USD	0,100USD

Figura 3: Tabla de precios de Amazon CloudFront en algunas regiones.

Podemos observar fácilmente que los precios son mucho mayores, ya que además se cobran las peticiones HTTP y HTTPS. La conclusión a la que queremos llegar es que a pesar de que HTTP ha abaratado muchísimo los costes de distribución de información, siguen siendo altos. Si el contenido a distribuir se encuentra en un servidor concreto, es necesario pagar los gastos generados tanto por distribuir el contenido como por el propio mantenimiento del servidor.

1.4 Dependencia

Como ya hemos visto, Internet está actualmente *hipercentralizado*. Esto implica que dependemos de una serie de infraestructuras clave para su correcto funcionamiento. En 2008 hubo problemas con una serie de cables submarinos de Internet, lo que conllevó que 14 países perdiesen la conexión, total o parcial-

mente[10].

2 La web distribuida

2.1 Tecnologías de web distribuida

3 El protocolo IPFS

IPFS es un sistema de archivos distribuido que recoge algunas de las ideas más exitosas de otros sistemas *peer to peer*. Los *nodos* IPFS almacenan objetos en el almacenamiento local y se conectan entre sí para transferir dichos objetos, que representan archivos y otras estructuras de datos. El protocolo IPFS está dividido en una serie de sub-protocolos que se encargan de proporcionar distintas funciones.

3.1 Identidades

Se encarga de gestionar la generación y verificación de la identidad de los nodos. Los nodos se identifican por un `NodeId`, el *hash* criptográfico de una clave pública, generado con S/Kademlia[2]. Los nodos almacenan su clave pública y su clave privada, encriptada con una *frase de contraseña*.

```
// Hash criptográfico
type NodeId Multihash
type Multihash []byte

// Claves
type PublicKey []byte
type PrivateKey []byte
```

[2]Kademlia es un protocolo de la capa de aplicación diseñado para redes P2P descentralizadas. S/Kademlia es una mejora de este protocolo que pretende resolver algunos problemas que presentaba el original, entre ellos la asignación segura de `nodeId`.[11]

```
type Node struct {
  NodeId NodeID
  PubKey PublicKey
  PriKey PrivateKey
}
```

Listing 1: Definición de Nodo.

```
difficulty = <parámetro integer>
n = Node{}

for cond := true; cond; cond = p < difficulty {
  n.PubKey, n.PrivKey = PKI.genKeyPair()
  n.NodeId = hash(n.PubKey)
  p = count_preceding_zero_bits(hash(n.NodeId))
}
```

Listing 2: Generación de identidad con S/Kamdelia.

Cuando dos *peers* se conectan, intercambian sus claves públicas y comprueban: `hash(other.PublicKey) == other.NodeId`. Si no es así, se cierra la conexión.

IPFS no utiliza una *función hash* concreta, sino que utiliza valores *autodescriptivos*. Así, los valores del *hash digest* se guardan en el formato *multihash*, que es de la forma:

```
<código función><longitud del digest><bytes del digest>
```

Esto permite que el sistema escoja la mejor función para cada caso y evolucione conforme cambien las opciones.

3.2 Red

Los nodos IPFS se comunican frecuentemente con cientos de otros nodos en la red, potencialmente a través del vasto Internet. El subsistema de red de IPFS incluye las siguientes funciones:

- Transporte: IPFS puede utilizar cualquier protocolo de transporte, y es más adecuado para WebRTC DataChannels[3] (para conexión con navegadores) o µTP[4].
- Fiabilidad: IPFS puede proporcionar fiabilidad si las redes subyacentes no lo proporcionan, usando µTP o SCTP.
- Conectividad: IPFS también utiliza las técnicas transversales ICE NAT[5].
- Integridad: opcionalmente se puede comprobar la integridad de los mensajes utilizando un *checksum hash*.
- Autenticidad: opcionalmente se puede comprobar la autenticidad de los mensajes utilizando HMAC[6] con la clave pública del remitente.

IPFS puede utilizar cualquier red: no depende ni asume acceso a IP. Esto permite utilizar IPFS en redes superpuestas[7]. IPFS almacena las direcciones como strings con formato byte `multiaddr` para que la red subyacente las utilice. `multiaddr` proporciona una forma de expresar direcciones y sus protocolos incluyendo soporte para encapsulación.

[3]Un canal de datos WebRTC te permite enviar texto o datos binarios a través de una conexión activa a un punto.[12]

[4]Micro Transport Protocol (µTP) es un protocolo libre multiplataforma diseñado para ser usado en las conexiones P2P del protocolo BitTorrent. Está implementado sobre el protocolo UDP, como alternativa a TCP para la transferencia de datos.[13]

[5]Interactive Connectivity Establishment (ICE) es una técnica usada en redes informáticas para encontrar formas de que dos ordenadores se comuniquen entre sí de la forma más directa posible en redes *peer-to-peer*.[14]

[6]Un código de autentificación de mensajes en clave-*hash* (HMAC) es una construcción específica para calcular un código de autentificación de mensaje (MAC) que implica una función *hash* criptográfica en combinación con una llave criptográfica secreta. Como cualquier MAC, puede ser utilizado para verificar simultáneamente la integridad de los datos y la autentificación de un mensaje.[15]

[7]Una red superpuesta (*overlay network*) es una red virtual de nodos enlazados lógicamente, que está construida sobre una o más redes subyacentes (*underlying network*).[16]

3.3 Enrutamiento

Los nodos IPFS requieren un sistema de enrutado que pueda encontrar tanto las direcciones de red de otros *peers* como *peers* que puedan distribuir objetos concretos. IPFS consigue esto utilizando una Tabla Hash Distribuida (DSHT por sus siglas en inglés) basada en S/Kamdelia y Coral[8]. Los datos pequeños (menores o iguales a `1KB`) se almacenan directamente en la DSHT. Para datos mayores, la DSHT almacena referencias, que son los `NodeIds` de los *peers* que pueden distribuir el bloque en cuestión.

```
type IPFSRouting interface {
  // Obtiene la dirección de un nodo concreto
  FindPeer(node NodeId)

  // Almacena un pequeño dato en la DSHT
  SetValue(key []byte, value []byte)

  // Obtiene un pequeño dato de la DSHT
  GetValue(key []byte)

  // Anuncia que el nodo puede distribuir un dato grande
  ProvideValue(key Multihash)

  // Obtiene el número de peers distribuyendo un dato
     grande
  FindValuePeers(key Multihash, min int)
}
```

Listing 3: Interfaz de la DSHT.

Distintos casos de uso pueden requerir diferentes sistemas de enrutamiento, por lo que el sistema de enrutamiento de IPFS puede cambiarse por uno que satisfaga las necesidades del usuario. Mientras que la interfaz descrita arriba se cumpla, el sistema continuará funcionando.

[8]CoralCDN es una red de distribución de contenido (CDN por sus siglas en inglés) gratuita y abierta basada en tecnologías *peer-to-peer* compuesta por una red mundial de *proxies* web y *nameservers*.

3.4 Intercambio de bloques: el protocolo BitSwap

En IPFS la distribución ocurre intercambiando bloques con *peers* utilizando un protocolo inspirado por BitTorrent: BitSwap. Como BitTorrent, los *peers* de BitSwap buscan conseguir un conjunto de bloques (`want_list`) y tienen otro conjunto de bloques que ofrecer (`have_list`). Sin embargo, a diferencia de BitTorrent, BitSwap no se limita a una serie de bloques en un *torrent*, sino que opera como un "mercado" donde los nodos pueden adquirir los bloques que necesitan, independientemente del archivo al que esos bloques pertenezcan. Esta idea requeriría que se implementase una moneda virtual, lo que haría necesario un control central sobre dicha moneda. IPFS implementa esto como la *estrategia BitSwap*, que veremos a continuación.

En el caso base, los nodos BitSwap tienen que proporcionar un valor inmediato entre ellos en forma de bloques. Esto funciona bien cuando la distribución de bloques entre nodos es complementaria: cada uno de ellos tiene lo que necesita el otro. Sin embargo lo más probable es que esto no sea así. En algunos casos los nodos tienen que *trabajar* para conseguir sus bloques. En el caso de que un nodo no tenga algo que quieran sus *peers*, buscará las pizas que estos quieren, con una prioridad menor que los que el propio nodo quiere. Esto incentiva que los nodos almacenen en caché y distribuyan piezas más raras, incluso si no están interesadas en ellas directamente.

3.4.1 Crédito BitSwap

El protocolo también debe incentivar que los nodos *siembren*[9] cuando no necesitan nada en particular, puesto que pueden tener bloques que otros nodos quieran. Los nodos BitSwap envían bloques a sus *peers* de forma optimista, esperando que la *deuda* se les pague en un futuro. Sin embargo, también hay que protegerse de los *leeches*, nodos que nunca comparten. Un sistema de créditos resuelve el problema:

[9] Del inglés *to seed*, en este caso significa distribuir información.

1. Los *peers* realizan un seguimiento de su saldo (en bytes verificados) con otros nodos.

2. Los *peers* envían bloques a otros *peers* deudores probabilísticamente, de acuerdo a una función que decrece conforme la deuda incrementa.

Si un nodo decide no enviar bloques a un *peer*, el nodo lo ignorará durante un tiempo `ignore_cooldown` (10 segundos por defecto en BitSwap). Esto evita que se juegue con la probabilidad simplemente haciendo más peticiones.

3.4.2 Estrategia BitSwap

Los *peers* BitSwap pueden adoptar diferentes estrategias, que proporcionarán resultados muy variados en el intercambio de bloques. La elección de una función debería procurar:

1. Maximizar la actividad de intercambio del nodo.

2. Evitar que los *leechers* se aprovechen del sistema y degraden el intercambio.

3. Sea efectiva y resistente a otras estrategias.

4. Ser clemente con *peers* de confianza.

Una elección de función que funciona en la práctica es una sigmoide, escalada por una *ratio de deuda*:

Definimos la *ratio de deuda*, r entre un nodo y su *peer* como:

$$r = \frac{\texttt{bytes\_enviados}}{\texttt{bytes\_recibidos} + 1}$$

Dado r, la probabilidad de enviar a un deudor es:

$$P(r) = 1 - \frac{1}{1 + exp(6 - 3r)}$$

Como se puede observar en la figura la función decrece fuertemente cuando la *ratio de deuda* del nodo aumenta. Esta radio es una medida de confianza:

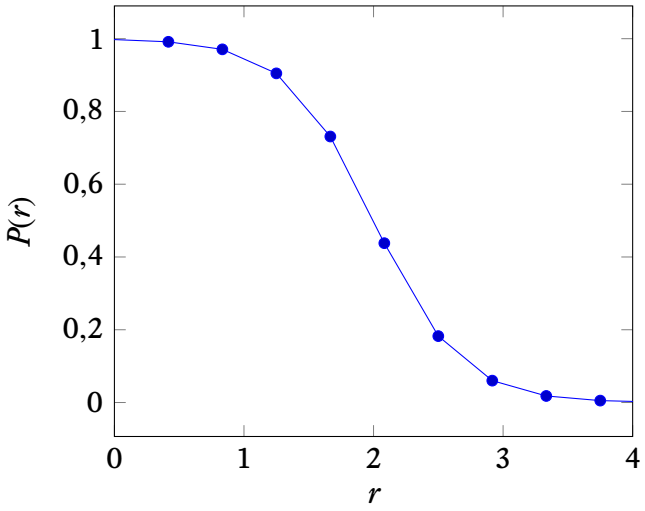

Figura 4: Representación de la función de probabilidad $P(r)$.

clemente con las deudas de los nodos con los que se que han intercambiado grandes cantidades de datos e implacable con aquellos desconocidos y que no son de confianza.

3.4.3 Libro mayor de BitSwap

Los nodos BitSwap mantienen una especie de "libro mayor"[10] (*ledger* en inglés) en el que se registran las transferencias con otros nodos. Cuando se inicia una conexión los nodos BitSwap intercambian su libro mayor. Si no concuerdan, el libro mayor se reinicia, perdiendo el crédito y la deuda. Es posible que hay nodos maliciosos que "pierdan" el libro mayor a propósito con el fin de borrar deudas. Es muy poco probable que un nodo acumule demasiada deuda como para querer perderla junto con la confianza, aunque otros nodos pueden considerarlo una mala conducta y negarse a intercambiar información.

```
type Ledger struct {
  owner NodeId
  partner NodeId
  bytes_sent int
  bytes_recv int
```

[10] Un *libro mayor* es un libro de contabilidad en que se registran las partidas importantes o globales.

```
  timestamp Timestamp
}
```

Listing 4: Implementación del libro mayor.

3.4.4 Especificación de Bitswap

Los nodos BitSwap siguien un protocolo simple.

```
// Estado adicional almacenado
type BitSwap struct {
  // Libros mayores del nodo
  ledgers map[NodeId]Ledger

  // Conexiones abiertas actualmente
  active map[NodeId]Peer

  // Checksums de bloques que necesita el nodo
  need_list []Multihash

  // Checksums de bloques que tiene el nodo
  have_list []Multihash
}

type Peer struct {
  nodeid NodeId
  // Libro mayor entre el nodo y este peer
  ledger Ledger

  // Marca de tiempo del último mensaje recibido
  last_seen Timestamp

  // Checksums de los bloques que quiere el peer
  // Incluye bloques que quieren los peers del peer
  want_list []Multihash
}

// Interfaz del protocolo
interface Peer {
  open(nodeid NodeId, ledger Ledger)
```

```
  send_want_list (want_list WantList)
  send_block (block Block) complete Bool
  close(final Bool)
}
```

Listing 5: Implementación del protocolo BitSwap

Un esquema de una conexión con un *peer* sería el siguiente:

1. Abrir: los *peers* envían `ledgers` hasta que estén de acuerdo.
2. Envío: los *peers* intercambian `wait_lists` y `blocks`.
3. Cierre: los *peers* desactivan la conexión.
4. Ignorado: (estado especial) un *peer* es ignorado (durante un tiempo específico) si la estrategia de un nodo evitar enviar información.

3.5 GDA de Merkle de objetos

IFPS implementa un GDA de Merkle, es decir, un grafo dirigido acíclico donde los enlaces entre los objetos son *hashes* criptográficos. Es una generalización de la estructura de datos de Git. Estos grafos proporcionan a IFPS una serie de propiedades muy útiles:

1. Direccionamiento de contenido: todo el contenido se identifica de forma única mediente el *checksum* de su `multihash`, incluyendo enlaces.
2. Resisencia a la manipulación: todo el contenido se verifica mediante su *checksum*. Si los datos se manipulan o corrompen, IPFS lo detecta.
3. Desduplicación: todos los objetos que tienen el mismo contenido son iguales y se almacenan una única vez.

```
type IPFSLink struct {
  // Nombre o alias del enlace
  Name string
```

```
  // Hash criptográfico del objetivo
  Hash Multihash

  // Tamaño total del objetivo
  Size int
}

type IPFSObject struct {
  // Array de enlaces
  links []IPFSLink

  // Datos opacos del contenido
  data []byte
}
```

Listing 6: Implementación de los objetos IPFS.

3.5.1 Rutas

Los objetos IPFS pueden ser recorridos con una API de rutas *string*. Las rutas duncionan como lo hacen en sistemas UNIX tradicionales y en la Web. Los enlaces del GDA de Merkle hacen que recorrerlo sea muy sencillo.

```
// Formato
/ipfs/<hash-del-objeto>/<nombre-del-objeto>

// Ejemplo
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

Listing 7: Rutas en IPFS.

3.6 Archivos

IPFS también define un conjunto de objetos para modelar un sistema de archivos de versiones sobre el GDA de Merkle. Este modelo de objetos es similar al de Git:

1. `blob`: un bloque de datos de tamaño variable.

2. `list`: una colección de `blobs` u otras listas.
3. `tree`: una colección de `blobs`, listas u otros árboles.
4. `commit`: una instantánea en el historial de versiones del árbol.

3.6.1 Objeto de archivo: `blob`

El objeto `blob` contiene una unidad de datos direccionable y representa un archivo. En IPFS los archivos pueden representarse por `lists` o `blobs`. Estos últimos no tienen enlaces.

```
{
  "data": "datos aquí",
}
```

Listing 8: Estructura JSON de un `blob`.

3.6.2 Objeto de archivo: `list`

El objeto `list` representa un archivo grande, formado a partir de varios `blobs` IPFS concatenados. Los objetos `list` contienen una secuencia ordenada de objetos `blob` o `list`.

```
{
  // Las listas pueden tener un array de objetos como
     datos
  "data": ["blob", "list", "blob"],

  // Las listas no tienen nombres en los enlaces
  "links": [
    { "hash": "nccSbCKcEWHAJ9wj0M72", "size": 38231 },
    { "hash": "2dUB2OS5O7rJBI57n7Hs", "size": 110 },
    { "hash": "BHHGh0pUmwiAyjN5mbTY", "size": 6468 },
  ]
}
```

Listing 9: Estructura JSON de un `list`.

3.6.3 Objeto de archivo: `tree`

El objeto `tree` de IPFS es similiar al de Git: representa un directorio, un *mapeo* de nombres a *hashes*. Los *hashes* hacen referencia a `blobs`, `lists`, `trees` o `commits`.

```
{
  // Los árboles pueden tener un array de objetos como
     datos
  "data": ["blob", "list", "blob"],

  // Las listas no tienen nombres en los enlaces
  "links": [
    { "hash": "nccSbCKcEWHAJ9wj0M72", "name": "test1", "
       size": 38231 },
    { "hash": "2dUB2OS5O7rJBI57n7Hs", "name": "test2", "
       size": 110 },
    { "hash": "BHHGh0pUmwiAyjN5mbTY", "name": "test3", "
       size": 6468 },
  ]
}
```

Listing 10: Estructura JSON de un `tree`.

3.6.4 Objeto de archivo: `commit`

El objeto `commit` en IPFS representa una instantánea en el historial de versiones de cualquier objeto. Es similar al de Git, pero puede hacer referencia a cualquier tipo de objeto. También enlaza al objeto autor.

```
{
  "data": {
    "type": "tree",
    "date": "2017-11-12 19:58:00",
    "message": "El mensaje del commit."
  },
  "links": [
    { "hash": "nccSbCKcEWHAJ9wj0M72", "name": "parent",
       "size": 38231 },
```

```
    { "hash": "2dUB2OS5O7rJBI57n7Hs", "name": "object",
       "size": 110 },
    { "hash": "BHHGh0pUmwiAyjN5mbTY", "name": "author",
       "size": 6468 },
  ]
}
```

Listing 11: Estructura JSON de un `commit`.

3.7 IPNS: Denominación y estado mutable

3.8 Cómo soluciona IPFS los problemas de HTTP

4 La web distribuida en la actualidad

4.1 La Wikipedia descentralizada

4.2 Neocities

4.3 La web del referéndum ilegal sobre la independencia de Cataluña (2017)

Referencias

[1] Miniwatts Marketing Group. *Internet World Stats*. 2017. URL: http://www.internetworldstats.com/stats.htm (visitado 14-10-2017).

[2] Maciej Cegłowski. *Web Design: the first 100 years*. 2014. URL: http://idlewords.com/talks/web_design_first_100_years.htm (visitado 14-10-2017).

[3] Jonathan Zittrain, Kendra Albert y Lawrence Lessig. "Perma: Scoping and Addressing the Problem of Link and Reference Rot in Legal Citations". En: *Legal Information Management* 14.2 (2014), págs. 88-99. DOI: 10.1017/S1472669614000255.

[4] John Perry Barlow. *A Declaration of the Independence of Cyberspace*. 1996. URL: https://www.eff.org/cyberspace-independence (visitado 14-10-2017).

[5] BBC. *Facebook has a billion users in a single day, says Mark Zuckerberg*. 2015. URL: http://www.bbc.com/news/world-us-canada-34082393 (visitado 16-10-2017).

[6] Sky News. *Google Outage Internet Traffic Plunges 40 percent*. 2013. URL: http://news.sky.com/story/google-outage-internet-traffic-plunges-40-10437065 (visitado 16-10-2017).

[7] Washington Post Barton Gellman y Ashkan Soltani. *NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say*. 2013. URL: https://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html (visitado 16-10-2017).

[8] Wikipedia. *Vídeos más vistos en YouTube*. 2017. URL: https://es.wikipedia.org/wiki/Anexo:Videos_m%C3%A1s_vistos_en_YouTube (visitado 17-10-2017).

[9] Amazon. *Precios de Amazon CloudFront*. 2017. URL: https://aws.amazon.com/es/cloudfront/pricing/ (visitado 17-10-2017).

[10] Kim Zetter. *Undersea Cables Cut; 14 Countries Lose Web*. 2008. URL: https://www.wired.com/2008/12/mediterranean-c/ (visitado 18-10-2017).

[11] Ingmar Baumgart y Sergio Mies. "S/Kademlia: A practicable approach towards secure key-based routing". En: 2 (ene. de 2008), págs. 1-8.

[12] Robert Nyman Alan Kligman. *WebRTC data channels*. 2013. URL: https://developer.mozilla.org/es/docs/Games/Techniques/WebRTC_data_channels (visitado 12-11-2017).

[13] Wikipedia. *Micro Transport Protocol*. 2017. URL: https://es.wikipedia.org/wiki/Micro_Transport_Protocol (visitado 12-11-2017).

[14] Wikipedia. *Interactive Connectivity Establishment*. 2017. URL: https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment (visitado 12-11-2017).

[15] Wikipedia. *HMAC*. 2017. URL: https://es.wikipedia.org/wiki/HMAC (visitado 12-11-2017).

[16] Wikipedia. *Red superpuesta*. 2017. URL: https://es.wikipedia.org/wiki/Red_superpuesta (visitado 12-11-2017).