MASTER EN BLOCKCHAIN
SMART CONTRACTS Y CRIPTOECONOMÍA

GUÍA DE ESTUDIO SEMANAL 3
DISEÑO Y DESARROLLO I
V1.0 05/06/2019

Profesores: Alberto Ballesteros Rodríguez, alberto.ballesterosr@uah.es

Coordinador: Jorge Vallet Fernández, jorge.vallet@uah.es

Índice de contenidos

Protocolo de Comunicación - Whisper

Whisper (shh) es un sistema de mensajería distribuida que forma parte del protocolo Ethereum, es decir, es un protocolo de comunicación para que las aplicaciones descentralizadas (DApps) se comuniquen entre sí. Se podría considerar un sistema híbrido que combina aspectos de la estructura de datos DHT (tabla hash distribuida) y del protocolo de datagramas de usuario (UDP).

Hay que tener en cuenta que la funcionalidad de este protocolo es *offchain*, por lo que no interfiere con la cadena de bloques (blockchain), simplemente es un protocolo de comunicación *peer-to-peer*.

Es comparable a la estructura DHT pero con un TTL configurable por cada una de las entradas, con acuerdos para la firma y cifrado de los valores de estas entradas. Al igual que sucede en las tablas hash distribuidas, Whisper permite indexación múltiple de las entradas, esto quiere decir que múltiples valores pueden estar sobre la misma entrada, además de que estos valores no tienen porqué ser únicos.

Uno de los objetivos más importantes de este protocolo es la comunicación a bajo nivel parcialmente asíncrona. Ha sido desarrollado desde cero para conseguir una transmisión y multidifusión eficiente. Por esto, no se utilizan sistemas de comunicaciones existentes como puede ser el protocolo TCP, ya que no está orientado a la conexión, ni únicamente para entregar datos entre peers específicos, y tampoco ha sido diseñado para maximizar anchos de banda o disminuir la latencia, ya que como se verá posteriormente es impredecible.

Whisper pretende ser un patrón para la construcción de nuevas aplicaciones descentralizadas (DApps), las cuales necesiten reconocimiento de datos a gran escala, transmisiones pequeñas con pequeñas alteraciones permitidas y donde se precisen unos criterios mínimos de privacidad. Con ésto se persigue otro de los objetivos que es reducir o retrasar el tráfico de poco o bajo valor.

Una de las características más relevantes de este protocolo es que está basado en la identidad de los nodos, es decir, cada uno de los nodos tiene un identificador para la emisión y recepción de los mensajes. Esencialmente, este protocolo de comunicación proporciona capacidades descentralizadas de mensajería *peer-to-peer* para que los nodos puedan comunicarse entre ellos.

Aunque la cadena de bloques se puede utilizar para la comunicación, es algo costoso y se requiere un consenso para el intercambio de mensajes. Whisper propone que cada nodo sea su propio minero del mensaje y no exista un consenso, y como se verá posteriormente, existen mecanismos para no inundar la red de mensajes irrelevantes.

Dark routing

Whisper opera en torno a la idea de ser configurable por el usuario con respecto a la cantidad de información que se filtra en relación con el contenido de la aplicación y, en última instancia, las actividades del usuario.

Es importante diferenciar entre cifrado y "oscuridad". Los protocolos p2p y cliente/servidor, actualmente en funcionamiento, utilizan el cifrado como una parte intrínseca del protocolo. A pesar de este cifrado, es posible ejecutar ataques a partir de la recopilación masiva de datos, haciendo posible que en estos protocolos mencionados se pueda comprometer con qué host se está realizando la comunicación.

En función de la finalidad de la aplicación, ésto puede ser considerado como suficiente para comprometer la privacidad de los participantes en la comunicación. A veces no es necesario obtener el contenido específico de una comunicación, ya que muchas veces ésta es cifrada, sino que simplemente se pueden obtener patrones como pueden ser conocer con quién se realiza la comunicación, cuándo y la frecuencia de ésta.

Existen en la actualidad sistemas destinados a la mitigación de estos ataques, a partir de la retransmisión y el cifrado de paquetes utilizando un tercer nodo. Ésto es considerado una solución válida en la mayoría de los casos, sin embargo, pueden provocar ineficiencias en el protocolo y retardos, además de posibles brechas si este tercer nodo es comprometido.

A raíz de lo anterior, entra en juego el concepto de "oscuridad". Un sistema completamente oscuro es inflexible a la filtración de información en las comunicaciones.

Whisper implementa este concepto, el cual lleva asociado un coste. Existe una compensación entre ancho de banda a consumir y latencia y el nivel de "oscuridad" que se desea aplicar a la comunicación. Se puede ofrecer un funcionamiento 100% oscuro siempre y cuando estemos dispuestos a aumentar el ancho de banda a consumir y un aumento en la latencia del mensaje a enviar.

También se puede aplicar al caso de que la red esté comprometida y siga siendo totalmente funcional. Gracias al nivel de oscuridad implementado, la comunicación puede hacerse entre los nodos que deseen realizar el intercambio de mensajes sin que ninguno de los nodos comprometidos conozca dicha comunicación.

Como ya se ha mencionado anteriormente, esta compensación es configurable por el usuario y podrá elegir entre privacidad del enrutamiento o eficiencia del enrutamiento.

A su máximo nivel de oscuridad, los nodos reciben y registran fragmentos de datos, que son reenviados con el fin de maximizar la eficiencia de transmisión de información a los *peers*.

Estos fragmentos de datos incluyen un elemento básico del protocolo conocido como *topic*, que normalmente es conocido como la clave de una tabla hash distribuida (DHT) pero que también puede ser utilizado como un *endpoint* seguro y probabilístico.

Existen dos métodos para enrutar la información, ambos resistentes frente ataques destinados a la recolección de información y que ofrecen una cantidad mínima de información acerca del enrutamiento que realizan.

El primero está basado en el protocolo de transferencia (*wire protocol*) explicado en el bloque anterior. Al igual que se utilizaba un sistema de reputación para calificar nodos en función de su utilidad, Whisper proporciona la capacidad de calificar a los *peers*. Según avance el tiempo, su conjunto de peers convergerá hacia aquellos que tiendan a entregar información útil.

Además, a medida que evolucione la red, el número de saltos entre el *peer* y los que tiendan a ser buenos transmisores de la información (que formen parte de su conjunto de *peers*) tenderá a 0.

La convergencia de *peers* también proporciona el incentivo para que los nodos entreguen información útil a los demás *peers*. El hecho de que un *peer* sea identificado como un nodo de bajo rendimiento, y por lo tanto, sea reemplazado en favor de otros genera incentivos para que los nodos cooperen y compartan la información más útil. Útil quiere decir que la información sea complicada de generar (a partir de un algoritmo PoW), que tenga un TTL bajo...

El segundo método está basado en el elemento *topic*. Las DApps informan a los nodos sobre qué *topic* es el útil. Así, los nodos comunican a cada uno de sus peers que son descubridores del *topic* en cuestión. El *topic* puede ser descrito de manera incompleta ya que puede contener uno o varios elementos de datos para reducir la información filtrada a la red e intentar evitar ataques. Ésto es algo que define el desarrollador o el usuario y queda bajo su responsabilidad.

Estos métodos se pueden desplegar en *peers*, es decir, a los *peers* más confiables se les puede proporcionar más cantidad de información y en el caso de las DApps podrían proporcionar un mayor número de elementos dentro del *topic*.

Para aumentar la robustez del protocolo, se utilizará el algoritmo de prueba de trabajo (PoW) para evitar ataques de denegación de servicio (DDoS) y evitar el spam, el cual podría provocar que un nodo al tener mucha información no útil necesite revelar más acerca de sus requisitos de aceptación de información.

Además, la utilización de este algoritmo permitirá que los *peers* establezcan criterios de aceptación de mensajes, es decir, los mensajes se procesarán (y reenviarán) sólo si el resultado de su PoW excede un umbral establecido, de lo contrario estos mensajes se descartarán.

Elementos básicos del protocolo

Envelopes

- En el caso de que Whisper sea considerado una tabla hash distribuida (DHT), estos *envelopes* se consideran elementos de la tabla.
- En el caso de que Whisper sea considerado un sistema de mensajería basado en datagramas, estos *envelopes* se convierten en paquetes que contienen los datagramas cifrados.

Siempre serán comprensibles por los nodos porque no se encuentran cifrados. Se transmiten como estructuras codificadas con el esquema RLP. La estructura coincide con la que se muestra a continuación:

```
[expiry: P, ttl: P, [topic0: B_4, topic1: B_4, ...], data: B, nonce: P]
```

`expiry`: Timestamp en Unix/Epoch de la fecha de vencimiento del *envelope*. Una vez transcurrido, el *envelope* ya no debe ser transmitido ni almacenado a no ser que se indique lo contrario. Si este valor es menor que el tiempo de vida del mensaje, se considerará inválido y se descartará además de penalizar al *peer* encargado de transmitirlo.

`ttl`: Tiempo de vida del mensaje dentro de la red, especificado en segundos. Su valor predeterminado es de 50 segundos.

`topic`: Los *topics* son un conjunto de índices, palabras clave o etiquetas codificadas dentro de una lista o como un solo elemento, que son registrados para que el receptor sea capaz de filtrar el mensaje o enrutarlo hacia su destinatario.

`data`: Campo que contiene la carga útil como una matriz de bytes sin formato, es decir, contiene los datos del mensaje además de cierta información y la firma. El cifrado de este campo es opcional.

`nonce`: Permite que el nodo emisor del mensaje demuestre que ha realizado cierta cantidad de trabajo arbitraria para la composición del mensaje. Más concretamente, se define a través de la función SHA3 del nonce concatenado con la función SHA3 del RLP del paquete que contiene el resto de parámetros (nonce excluido) cada uno con una longitud fija de 256 bits. Cuanto menor sea el valor del hash resultante de la función SHA3, mayor trabajo realizado se puede demostrar. Ésto se relaciona con lo comentado en el punto anterior acerca del algoritmo PoW.

Topic

Como ya se ha definido en el punto anterior, un *topic* es un elemento o una lista de ellos que se utilizan para codificar el mensaje y hacer posible su filtrado por las partes interesadas.

Son criptográficamente seguros, es decir, clasificaciones parciales probabilísticas del mensaje. Cada *topic* se define como un conjunto de los primeros 4 bytes de la función SHA3 de 256 bits de los datos arbitrarios proporcionados por el creador del mensaje. Por ejemplo, si la etiqueta es una clave pública de un *wallet* de Ethereum se obtendrán los 4 primeros bytes por la izquierda del hash resultante:

`d2f254c1``9efcf5974e19f2f7f6e83cba8cdc0c0e51e4c2af84493a2e3425a000`

Se han seleccionado 4 bytes porque ante un gran número de *topics* se considera el espacio mínimo para que no sucedan colisiones. Es un valor que puede sufrir modificaciones o que en el futuro puede funcionar dinámicamente en función de la cantidad de *topics* y mensajes.

Mensaje

Según el estado actual del protocolo, el mensaje se forma a partir de la concatenación de un byte utilizado para flags, seguido de cualquier dato adicional requerido por los flags y finalmente la carga útil. Esta matriz de bytes es lo que se ha definido anteriormente como `data`.

- Flag: 1 byte
- Firma: 65 bytes (Opcional)
- Carga útil: Tamaño no fijo. Datos binarios sin formato.

El bit 0 del flag indica si existe una firma. Se considerará como mensaje inválido si se indica que existe una firma pero el tamaño total es inferior a 66 bytes.

No es posible actualmente saber si el campo `data` está cifrado o no, por lo que el receptor debe conocer de antemano esta información. Por el momento se puede utilizar un *topic* establecido específicamente para esta información, y que sea filtrado por el receptor.

Una DApp no debe conocer nada acerca de aquellos *envelopes* cuyos mensajes no puede inspeccionar. Gracias a la oscuridad implementada por Whisper, es posible que los nodos intercambien *envelopes* independientemente de si son capaces o no de decodificar el mensaje.

Los nodos envían y reciben *envelopes* en todo momento. Cada nodo establece un mapa de *envelopes* donde se indexan a partir de su tiempo de expiración. También deben ser capaces de entregar mensajes a partir de *envelopes* (en caso de ser el receptor) a la aplicación.

A pesar de que este protocolo se utilice para la comunicación de mensajes de tamaño reducido, existe la posibilidad de que un nodo llene por completo su memoria. En este instante, el nodo debe descartar *envelopes* que considere menos importantes o que estime que no deben ser enrutados hacia sus *peers*. El funcionamiento ideal de un nodo es enviar *envelopes* con un bajo TTL y con altos resultados a partir del algoritmo de prueba de trabajo (PoW). Si un nodo envía *envelopes* caducados o con un TTL que ya haya sucedido será penalizado.

Es importante que un nodo conserve un conjunto de *topics* definidos por una DApp para hacer posible la recepción y filtración del mensaje a partir de estos *topics*.

Para el envío de *envelopes*, un nodo debe colocar dicho *envelope* en el conjunto de los mismos pertenecientes a dicho nodo. A medida que la red evoluciona y con el tiempo, el nodo aprenderá y será capaz de retransmitir los *envelopes* a su debido tiempo.

Composición del *envelope* a partir de la carga útil:

- El mensaje se compone a partir del `data`, el cual a su vez se compone del flag asociado a la firma, la firma de una identidad válida si el usuario la proporciona y la carga útil proporcionada por el usuario.
- Si la firma ha sido proporcionada, el `data` se cifra a partir de ésta.
- `Topics` a partir de los 4 primeros bytes como se ha especificado anteriormente.
- Establecer un `TTL` por parte del usuario.
- Establecer un tiempo de expiración, el cual debe contener una fecha en Unix/Epoch además del `TTL`.
- `Nonce` a partir de la PoW.

Casos de uso

- DApps que necesitan publicar pequeñas cantidades de información y que la publicación dure una cantidad determinada de tiempo.

- DApps que necesitan comunicarse entre sí para finalmente colaborar en una transacción.

- DApps que precisen de comunicaciones que no sean en tiempo real.

- DApps que precisen de una comunicación oscura entre dos partes que no conocen nada el uno del otro, exceptuando un hash.

Protocolo de Almacenamiento - Swarm

Swarm (bzz) es una plataforma de almacenamiento distribuido y un servicio de distribución de contenido. Es una red de almacenamiento descentralizada, distribuida y *peer-to-peer*. Los archivos en esta red se tratan mediante el hash de su contenido. Esto se desarrolla como un servicio nativo de capa base para la pila Ethereum web 3.0. Swarm está integrado con devp2p, que es la capa de red multiprotocolo de Ethereum. El objetivo es una solución de almacenamiento y servicio *peer-to-peer* que no tenga tiempo de inactividad, que sea resistente a los ataques DDoS, tolerante a fallos y resistente a la censura, así como autosostenible gracias a un sistema de incentivos integrado.

Dichos sistemas de incentivos modelan a los participantes individuales como agentes que siguen su propio interés racional; sin embargo, el comportamiento emergente de la red es enormemente más beneficioso para los participantes sin una coordinación.

A diferencia de sistemas descentralizados de almacenamiento distribuido como puede ser BitTorrent, donde el contenido se aloja en un host y se comparte desde este mismo servidor. Swarm proporciona el alojamiento de archivos en sí mismo como un servicio de almacenamiento en la nube descentralizado. Se persigue que se pueda realizar la carga de archivos y recuperarlos más tarde al igual que sucede con los actuales servicios en la nube sin depender de nuestro host.

Otra diferencia, a pesar de que se ha introducido, en comparación con estos sistemas descentralizados actuales es el sistema de incentivos que permitirá a los participantes agrupar sus recursos de almacenamiento y ancho de banda con el fin de proporcionar servicios de contenido global a todos los participantes.

Entre sus posibilidades se encuentran desde el alojamiento de aplicaciones web en tiempo real de baja latencia hasta para almacenamiento de datos que no son utilizados continuamente.

Antes de continuar es importante tener claros los siguientes conceptos o elementos que intervienen en Swarm.

Fragmento (*chunk*)

Fragmentos de datos de tamaño limitado (actualmente a 4 KB). Es la unidad básica de almacenamiento y recuperación en Swarm.

Referencia (*reference*)

Identificador único de un archivo que permite a los clientes recuperar y acceder al contenido de éste. El cifrado del archivo es opcional y en función de ésto la referencia del mismo se verá afectada. En ambos casos la referencia se serializará con 64 bytes hexadecimales.

Manifest

Estructura de datos que describe colecciones de archivos, es decir, pueden representar directorios.

Funcionamiento

El protocolo se define como una red, un servicio y un protocolo además de un modo de interacción.

Implementa un almacenamiento de fragmentos (*chunks*) dividido en contenido distribuido. Estos fragmentos son BLOB (*Binary Large OBject*) con un tamaño máximo fijo de 4 KB. El direccionamiento del contenido es determinista, concretamente, la dirección de cualquier fragmento se deriva de su contenido a partir de una función hash que como entrada utiliza este fragmento y resultará en una clave identificativa de 32 bytes.

Este hash de 32 bytes es la dirección que los clientes utilizarán para la recuperación del fragmento. La protección del fragmento está asegurada debido a que la función hash no es reversible y es libre de colisiones entre archivos con las funciones utilizadas. Un cliente puede comprobar si el fragmento ha sufrido modificaciones o está dañado a partir de realizar la función hash de éste.

Los nodos que conforman la red de Swarm dedican recursos, como pueden ser espacio de disco duro, memoria o ancho de banda, para almacenar y distribuir fragmentos. Cada uno de los nodos de esta red se identifica a partir de la función SHA3 de 256 bits de su clave pública de Ethereum. Este identificador se encuentra en el mismo espacio de claves que los resultados de la función hash de los fragmentos, conocido como *overlay network*.

Cuando se carga un fragmento, el protocolo normalmente determinará que se almacenará en los nodos más cercanos a la dirección del fragmento. Cuando se refiere a los más cercanos, viene determinado por el concepto de distancia en el espacio de direcciones, similar al utilizado en Kademlia. Este proceso de carga de los fragmentos en las direcciones, en las cuales se supone que debe estar almacenado, se denomina sincronización.

A su vez, cuando se desea realizar la descarga o recuperación de un fragmento, el nodo enviará una consulta a los nodos que más cerca estén de la dirección del fragmento solicitado, nuevamente a partir del criterio de distancia ya comentado.

Swarm utiliza un conjunto de conexiones TCP/IP en el que cada nodo tiene un conjunto de *peers* semi-permanentes. Todos los mensajes del protocolo de transferencia entre nodos son retransmitidos nodo a nodo saltando entre peers con conexión activa.

Los nodos administran activamente las conexiones con sus *peers* con el fin de mantener un conjunto particular de conexiones. que permiten la sincronización y la recuperación del contenido mediante enrutamiento basado en claves. Esto conlleva que siempre es posible enrutar eficientemente una solicitud de almacenamiento o recuperación de fragmentos a través de estas conexiones entre *peers* hacia los nodos que están más cerca de la dirección del contenido.

Los fragmentos que han sido retransmitidos localmente por un nodo se almacenan en la caché de éste para que estén disponibles la próxima vez que sean solicitados. Como consecuencia, el contenido más solicitado o más popular acaba siendo replicado de manera más redundante en la red, provocando una disminución en la latencia de las solicitudes de dicho contenido. El protocolo, para esto, incentiva al nodo a almacenar todo el contenido encontrado hasta llenar su espacio disponible, gracias a lo cual es posible evitar ataques DDoS sobre los poseedores originales del contenido.

En el caso de que exista un número de fragmentos entrantes que superen el espacio disponible en el nodo, éste borrará fragmentos más antiguos basándose en la media de utilidad esperada por dichos fragmentos. Esto es posible si el nodo se basa simplemente en la cantidad de solicitudes realizadas por fragmento.

A raíz de lo anterior, se puede llegar a la conclusión de que el contenido que ya no sea solicitado eventualmente, que no esté actualizado o que nunca haya sido popular será considerado como contenido basura y se eliminará a menos que esté protegido.

Swarm hash

Swarm permite la fragmentación de cualquier archivo legible como pueden ser fotos, vídeos, documentos de texto… A través de un fragmentador (*chunker*), el cual trocea el archivo en fragmentos de tamaño fijo. Cada uno de estos fragmentos se pasa a través de una función hash y se sincroniza con sus *peers*.

Los hash obtenidos se empaquetan en fragmentos, conocidos como fragmentos intermedios, de manera reiterada. Actualmente 128 hash constituyen un nuevo fragmento. A partir de lo anterior, la representación de los datos viene determinada por un árbol de Merkle, siendo la raíz de éste la dirección utilizada para la recuperación del archivo cargado.

Cuando se desea recuperar este archivo se busca el hash raíz, aunque puede suceder que este hash raíz sea un fragmento intermedio, lo que se interpretará como una serie de hash para dirigirse a fragmentos en un nivel inferior. Esto conlleva que se alcance el nivel de datos para que el contenido pueda ser entregado.

El hecho de que se utilice un árbol de Merkle es debido a que éste brinda una protección de integridad y es capaz de asegurar que los datos no han sido manipulados.

Otro criterio a tener en cuenta es el reducido tamaño de los fragmentos (como ya se ha comentado anteriormente, fijado a 4 KB), lo que permite recuperación de archivos en paralelo y en caso de que la red tenga fallos se desperdiciara una pequeña cantidad del tráfico.

Direccionamiento

Aunque ya se ha introducido el concepto de *manifest* al comienzo del apartado, Swarm proporciona una capa de organización del contenido a partir de estos archivos *manifest*.

Estas estructuras de datos son matrices '.json' que pueden contener otros *manifest*. Cada *manifest* se refiere a una ruta, a un tipo de contenido y a un hash que apunta al contenido.

Para una mejor comprensión del direccionamiento a partir de una estructura *manifest*, puede imaginar que un *manifest* es capaz de apuntar a un sitio web alojado en Swarm. La dirección utilizada en este caso apunta a un *manifest*, el cual apuntará a otras entradas *manifest* que componen el sitio, esto puede suceder recursivamente en función del sitio web alojado y la cantidad de conjuntos de datos de los que disponga. Se puede considerar que funcionan como *sitemaps* o tablas de enrutamiento que relacionan direcciones con contenido. Gracias a la estructura en árboles de Merkle la integridad del sitio está asegurada.

También puede comprender un *manifest* como un árbol de directorios, por lo que un directorio y un host virtual pueden verse de la misma forma. Puede imaginar un sitio web de alojamiento de archivos en la nube como puede ser Google Drive.

A pesar del potencial de Swarm hay que tener en cuenta lo siguiente, cada hash asociado a un fragmento o contenido es inmutable por lo que este contenido no se puede actualizar o editar. Además, debido a la sincronización mediante *peers*, algo que se ha cargado al protocolo no puede hacerse invisible ni ser inédito ni será posible eliminarlo.

Sin embargo, si volvemos al ejemplo del sitio web, se puede volver a crear un *manifest* con las nuevas entradas o descartar las existentes. Esto no afecta al contenido al que apuntan las entradas del manifest. Además, si no se quiere editar continuamente la dirección que apunta al contenido por las actualizaciones realizadas, existe la posibilidad de utilizar direcciones mutables basadas en un nombre que apuntarán siempre a la última versión del contenido, para ésto se utiliza el protocolo ENS (*Ethereum Name Service*), el cual se tratará posteriormente.

Teniendo esto en cuenta, se debe ser consciente del contenido que se comparte y cómo se hace.

Protocolo de Almacenamiento externo - IPFS

IPFS (InterPlanetary File System) es un protocolo de hipermedia distribuido, *peer-to-peer* y de código abierto que pretende funcionar como un sistema de archivos omnipresente para todos los dispositivos.

Intenta abordar las deficiencias del modelo cliente-servidor y la web HTTP a través de un sistema de intercambio de archivos p2p. Esencialmente, es un sistema de archivos versionado que puede almacenar archivos y rastrear versiones del mismo a lo largo del tiempo.

Funcionamiento

Al igual que protocolos analizados anteriormente, IPFS utiliza tablas hash distribuidas, además de un DAG de Merkle y una versión general del protocolo BitTorrent conocida como BitSwap.

IPFS proporciona un hash único para cada uno de los archivos. Si el contenido del archivo es modificado, el hash será diferente. IPFS puede usar el contenido del archivo para localizar su dirección en lugar de utilizar nombres de dominio como en el protocolo HTTP.

Se encarga de eliminar archivos redundantes en la red, además de que existe un historial de grabación por el que se pueden encontrar las distintas versiones de un documento.

Al igual que sucede en Ethereum con las direcciones o con los identificadores de archivo en Swarm, en IPFS sucede que es difícil recordar el identificador o hash de un archivo, por lo que existe IPNS para localizar los hash de IPFS.

Tabla hash distribuida (DHT)

Aunque es un concepto tratado a lo largo del bloque, se contempla su comportamiento en el protocolo IPFS.

La clave será el hash resultante sobre el contenido del archivo. De la siguiente forma:

```
QmSRdZBUCk2erbb4tpGfJ8f76JykSs1HWYtXPAKaiDph9m
```

Al realizar una consulta en un nodo IPFS para el contenido relacionado con dicho hash, el nodo buscará en la tabla hash distribuida qué nodos tienen el contenido solicitado.

Cuando se trata de contenidos cuyo tamaño de archivo es inferior a 1 KB se almacenan directamente en la DHT. Para contenidos superiores a este tamaño, son las referencias lo que se almacena dentro de la DHT, es decir, los peers que pueden ofrecer el contenido.

Merkle DAG

Un merkle DAG es una mezcla entre un árbol de Merkle y un Grafo Acíclico Dirigido (DAG)

El árbol de Merkle se encarga de asegurar que los bloques de datos sean correctos, no estén dañados ni alterados.

Un Merkle DAG es básicamente una estructura de datos donde los hash se usan para referenciar bloques de datos y objetos en un DAG. Por lo tanto, todo el contenido de IPFS puede identificarse de manera única, ya que cada bloque de datos tiene un hash único.

El principio central de IPFS es modelar todos los datos en un DAG de Merkle global. Este DAG proporciona a los objetos de IPFS propiedades como la autenticación, que sean permanentes en la red, que sean universales y que estén descentralizados.

Sistema de control de versiones

Gracias a la utilización de la estructura Merkle DAG, es posible construir un sistema de control de versiones distribuidas. Un ejemplo claro de esto es GitHub, que permite que distintos desarrolladores colaboren en proyectos de manera simultánea. Esto es gracias a que se permite el almacenamiento de distintas versiones de archivos usando un Merkle DAG.

IPFS utiliza un modelo similar para objetos de datos. Siempre y cuando los datos se correspondan con los datos originales y cualquier versión nueva sea accesible, cualquier versión del contenido puede ser obtenida.

Propiedades:

- Los objetos inmutables representan archivos, directorios y cambios (*commits*).
- El contenido de los objetos está referenciado mediante los hash.
- Los enlaces al contenido de otros objetos están incrustados, formando un Merkle DAG.
- Los metadatos relacionados con las versiones como pueden ser en GitHub las ramas, etiquetas… son simplemente referencias de punteros y de muy poco tamaño.
- Un cambio de versión actualiza la referencia o agrega nuevos objetos.
- La distribución de cambios de versión se basa en transferir objetos y actualizar referencias.

Intercambio de bloques - BitSwap

A diferencia del protocolo BitTorrent donde cada archivo tenía su conjunto de *peers*, formando una red p2p entre sí, IPFS es un gran conjunto de *peers* donde almacenar todos los datos. IPFS implementa una versión general del protocolo BitTorrent conocida como BitSwap.

Cuando un *peer* realiza la conexión ejecuta las siguientes instrucciones:

- `have_list`: Intercambia cuáles son los bloques que tiene.
- `want_list`: Intercambia cuáles son los bloques que busca.

El intercambio de datos en BitSwap se basa la transferencia de datos previa entre dos *peers*. Cuando los *peers* realizan este intercambio de bloques, realizan un seguimiento de la cantidad de datos que comparten y la cantidad de datos que reciben. Este intercambio genera una contabilidad que se almacena en BitSwap Ledger, asociando el hecho de crear crédito al de compartir datos y el hecho de crear deuda al de recibir datos.

Antes de realizar un intercambio de datos, los *peers* necesitan intercambiar el *ledger*, en caso de ser idéntico se realizará el intercambio. En caso de que no sea idéntico, el nodo se desconectará.

Ante la solicitud de un bloque:

- Si un *peer* tiene crédito, es decir, ha compartido más de lo recibido, se le enviará el bloque solicitado.

- Si un *peer* tiene deuda, es decir, ha compartido menos de lo recibido, será una función determinista, que trabaja en función de la cantidad de deuda acumulada por el nodo, quien decidirá si se le envía o no el bloque solicitado.

Sistema de Autocertificación de Archivos

Antes de finalizar, se tratará el último componente esencial del protocolo IPFS. El Sistema de Autocertificación de Archivos (SFS).

Es un sistema de archivos distribuido que no requiere de ningún permiso para el intercambio y transferencia de archivos. Es posible acceder de forma segura a un contenido remoto como si fuera local, esto es gracias a que la autocertificación viene dada por el servidor (que firma el nombre del archivo y lo autentica).

IPFS se basa en SFS para crear IPNS (InterPlanetary Name System), algo parecido a lo que en Ethereum es el Ethereum Name Service (ENS), el cual se tratará en el siguiente apartado del bloque.

IPNS es un SFS que utiliza criptografía de clave pública para autocertificar contenidos publicados por los usuarios de la red. Esto es que cualquier usuario que suba contenido a la red, identificará éste como único, pero no sólo es así para el contenido sino también para los nodos. Cada nodo de la red tiene un conjunto de claves públicas, privadas y un identificador de nodo que es el hash de su clave pública. Así, los nodos utilizarán sus claves privadas para firmar cualquier dato que publiquen, y la autenticidad de este contenido puede verificarse utilizando la clave pública del remitente.

Entonces, IPNS permite generar una dirección para un sistema de archivos remoto, donde el usuario puede verificar la validez de la dirección. Por lo tanto, el nombre de un sistema de archivos SFS certifica su propio servidor. El direccionamiento de sistemas de archivos remotos usa el siguiente esquema:

`/sfs/<Ubicación>:<HostID>`

Ubicación: Dirección de red del servidor.

HostID: `hash(public_key || Ubicación)`.

-

ENS (Ethereum Name Service)

ENS es el Servicio de nombres de Ethereum, un sistema de nombres distribuido, abierto y extensible basado en la cadena de bloques de Ethereum.

Se utiliza para resolver una amplia variedad de recursos, el estándar inicial de ENS define la resolución para direcciones de Ethereum pero el sistema es extensible por diseño, permitiendo que más tipos de recursos se puedan resolver en el futuro sin depender de los componentes centrales de ENS, los cuales requieren de actualizaciones.

ENS se puede usar para resolver una amplia variedad de recursos, el estándar inicial de ENS define la resolución para direcciones de Ethereum pero el sistema es extensible por diseño, lo que permite que más tipos de recursos se resuelvan en el futuro sin los componentes centrales de ENS.

El objetivo de ENS es resolver identificadores legibles por los humanos a identificadores legibles por las máquinas. Se pueden usar nombres registrados con el Servicio de Nombres de Ethereum para resolver:

- Direcciones de Ethereum
- Hash de IPFS
- Hash de Swarm

Pretende resolver el mismo problema que el Sistema de Nombres de Dominio (DNS) para Internet, aunque los detalles técnicos y de implementación son sustancialmente diferentes.

Componentes

Registry contract

Es un contrato único, el cual lista el dominio y los subdominios junto con los datos asociados a los mismos. Los datos asociados incluyen:

- Propietario del dominio o subdominio, el cual puede ser una cuenta externa o un contrato.
- Dirección del contrato *resolver* del dominio.
- TTL.

El propietario de un dominio o subdominio puede establecer el contrato *resolver* que desee utilizar. También, puede transferir la propiedad del dominio, así como cambiar la propiedad de los subdominios.

Resolver contract

Son los contratos que traducen los nombres legibles por los humanos a los identificadores legibles por las máquinas.

Cualquier contrato que implemente el estándar de resolución de ENS puede configurarse como un *resolver*. Cada tipo de registro, ya sea una dirección de Ethereum, un hash de IPFS o Swarm tiene sus propios métodos. Estos métodos deben ser implementados por el contrato resolver para ofrecer correctamente el recurso.

ENS no sólo permite traducir desde un nombre legible como:

```
mscethereum.eth
```

a un identificador legible por la máquina como puede ser:

```
0xff7a942d06ce768f303c28177715ae11401c3197
```

También permite la resolución inversa, lo que permite asociar identificadores como nombres canónicos con direcciones de Ethereum.

Al igual que DNS, utiliza un sistema de nombres jerárquicos, separados por puntos, conocidos como dominios. El propietario de un dominio obtiene el control total sobre la asignación de los subdominios.

- El dominio '.eth' es el utilizado en la red principal de Ethereum.
- El dominio '.test' es el utilizado en las redes de prueba como Ropsten o Rinkeby.

Además de los contratos *Registry* y *Resolver*, existe otro componente que es importante conocer para comprender el funcionamiento de ENS.

Registrar contract

Es el responsable de asignar nuevos nombres de subdominios a los usuarios dentro del espacio de nombres al que pertenecen. El principal *registrar* es '.eth', pero cualquiera que posea un dominio puede asignar la propiedad a un contrato y crear un *registrar* propio.

ENS es un sistema completamente descentralizado. La distribución de nuevos dominios bajo el dominio de nivel superior ".eth" (por ejemplo, 'mscethereum.eth') se maneja mediante un proceso de subasta que se ejecuta en la cadena de bloques de Ethereum, y cualquiera puede participar en el proceso de subasta para reservar un dominio para sí mismo.

Este proceso de asignación de dominios funciona mediante un modelo de subasta conocido como Subasta Vickrey basado en tres etapas.

1. Alguien inicia una subasta para un dominio y hace una oferta. A partir de aquí se inicia un temporizador de tres días para que el resto de interesados participen en la subasta. Durante estos tres días no se conocen detalles acerca de la subasta, no se conoce ni la cantidad ni el dominio sobre el que se está realizando la puja.

2. Finalizado el plazo de tres días, comienza un nuevo periodo de dos días en el cual todos los que hagan una oferta deben revelar todos los detalles de la misma o perderán su participación en la puja. Si una puja no es la ganadora se le devuelve al usuario exceptuando una comisión del 0.5%.

3. Al final de este periodo de dos días donde se revelan las ofertas, el ganador es la persona que revelase la oferta más alta, pero sólo debe pagar la cantidad que revelase el segundo mejor pujador. La cantidad está bloqueada en un contrato mientras que el ganador conserve el control del dominio.

4. El proceso se finaliza cuando el ganador envía una transacción de finalización para que se le otorgue el control del dominio y se le reembolse la cantidad adicional no utilizada para la adquisición del nombre.

Namehash

Para terminar de comprender el proceso, una característica importante de ENS es que éste no trabaja con nombres en texto plano, sino con la representación de dicho nombre en forma de hash de 32 bytes. El hecho de utilizar esta representación simplifica el procesamiento y el almacenamiento, ya que permite cadenas de texto de longitud arbitraria, las cuales en texto plano harían que el proceso fuese muy ineficiente. Además aumenta la privacidad de los nombres.

Como ENS es un sistema jerárquico de nombres, es necesario representar esa jerarquía a partir de un algoritmo que no sea directamente realizar el hash del dominio o subdominio completo. Por ejemplo: sha3(‘mscethereum.eth’) dará un resultado el cual no permite identificar el dominio o subdominio. Por ello se utiliza el algoritmo conocido como namehash.

Namehash es una función recursiva que para calcular el hash final, comienza con el componente más a la derecha y calcula repetidamente el hash, luego lo combina con el hash del nombre del componente anterior. De esta forma se preserva la jerarquía de los dominios.

```
namehash([label, …]) = keccak256(namehash(…), keccak256(label))

namehash('a.xyz') = sha3(namehash('xyz'), sha3('a'))
```

Actividades propuestas

Se recomienda leer y asimilar detenidamente este documento, consultar todos los enlaces y recursos asociados y familiarizarse con las herramientas y conceptos de cara a la sesión práctica/presencial, en la que se realizarán una serie de actividades relacionadas.

La evaluación del bloque se realizará mediante la PEC 2.

Evaluación

Este bloque se evaluará mediante la Prueba de Evaluación Continua 2 (PEC2).

Para fomentar el uso de git y el lenguaje markdown (ambas se utilizarán a lo largo de la asignatura). Se recomienda crear un archivo markdown (.md) con las respuestas a las actividades y alojarlo en GitHub o GitLab.

Markdown cheatsheet:

- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Si usted desea colgar algún pantallazo sobre alguna actividad se recomienda alojarlo en el mismo repositorio de git y hacer referencia al mismo dentro del .md.

Instrucciones de entrega:

- Dentro del archivo Prueba de Evaluación Continua 2 (PEC2) encontrará las instrucciones.

Recursos adicionales para el bloque

Whisper

- Web3js - Whisper
  - https://web3js.readthedocs.io/en/1.0/web3-shh.html
- Whisper Wiki
  - https://github.com/ethereum/wiki/wiki/Whisper
- Resumen Whisper
  - https://github.com/ethereum/go-ethereum/wiki/Whisper-Overview
- Whitepaper Whisper
  - https://github.com/ethereum/wiki/wiki/Whisper-PoC-2-Protocol-Spec

Swarm

- Swarm docs
  - https://swarm-guide.readthedocs.io/en/latest/index.html
- Repositorio Swarm
  - https://github.com/ethersphere/swarm
- Arquitectura Swarm (Avanzado)
  - https://swarm-guide.readthedocs.io/en/latest/architecture.html
- Sistema de incentivos
  - https://swarm-gateways.net/bzz:/theswarm.eth/ethersphere/orange-papers/1/sw%5E3.pdf

IPFS

- IPFS
  - https://ipfs.io/
- IPFS docs
  - https://ipfs.io/docs/
- Arquitectura IPFS
  - https://github.com/ipfs/specs/tree/master/architecture

ENS (Ethereum Name Service)

- ENS
  - https://ens.domains/

- Sistema de subasta
  - https://medium.com/the-ethereum-name-service/explaining-the-ethereum-namespace-auction-241bec6ef751

- ENS docs
  - https://docs.ens.domains/

- Repositorio ENS
  - https://github.com/ensdomains/ens

- Namehash
  - https://docs.ens.domains/contract-api-reference/name-processing#hashing-names