UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA UDELAR

“DISTRIBUCIÓN DE ARCHIVOS CON IPFS EN UNA RED BLOCKCHAIN EDUCATIVA”

Ing.Reyna Der Boghosian

Tesis de Maestría presentado a la Facultad de Ingeniería de la Universidad de la República en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Informatica

Tutores

Dra.Regina Motz

Tribunal

DR. NELSON PIEDRA (REVISOR)
DR. ALFREDO VIOLA (PRESIDENTE)
DRA. LAURA GONZÁLEZ

Montevideo, Uruguay
Junio de 2022

Resumen

La permanencia de información en la web es un tema actual, aún no satisfactoriamente resuelto. Es frecuente encontrarse con sitios o servicios web que dejan de ofrecer información.
En esta tesis se analiza este problema, estudiando cómo el protocolo Interplanetary File System (IPFS) puede brindar soluciones para algunas de las causas que impiden mantener el acceso a la información en la web.
Se presentan las características de IPFS que lo posicionan como un buen candidato para oficiar de solución para este tipo de problemas, así como también mitigar el problema de almacenamiento que experimentan las redes distribuidas tales como Blockchain.
En una red Blockchain el espacio de almacenamiento es un punto critico debido a que todas las transacciones que ocurren en la Blockchain son almacenadas en cada nodo que compone la red. A medida que crece la demanda de los nodos integrantes de la Blockchain la capacidad de almacenamiento de cada nodo se ve limitada ante el inminente registro de las transacciones a diario. IPFS ofrece un mecanismo de almacenamiento distribuido en una red la cual ayuda a reducir el tamaño de los nodos que componen a una Blockchain.
En particular se analiza el impacto de trabajar con IPFS en una plataforma Blockchain educativa, desarrollada en el marco del proyecto SELI (Smart Ecosystem for Learning and Inclusion), y se compara su uso en las combinaciones de arquitecturas peer-to-peer (P2P) y cliente-servidor. Se analizan las dinámicas de manejo de archivos en un cluster-IPFS y se presenta una posible extensión de la arquitectura Blockchain de SELI para uso de IPFS totalmente distribuido, D-SELI. Se presenta además un estudio de comportamiento del uso de Meteor y de IPFS.

Índice general

Índice de figuras

Capítulo 1

Introducción

Actualmente la web se ha convertido en el repositorio de información más importante donde se recurre cotidianamente en búsqueda de información y datos, donde se realizan transacciones comerciales y hasta donde se mudan las aulas de clase para estudiar de forma colaborativa en plataformas de gestión del aprendizaje, entre muchas otras actividades. Por ejemplo, cuando una biblioteca de una institución educativa expone sus servicios en la web, lo hace generalmente a través de servicios web y plataformas digitales en las cuales los estudiantes acceden a recursos educativos teniendo la posibilidad de disponer del material necesario para cumplir con sus metas académicas. Sin embargo, debido a distintos motivos, puede ocurrir que los estudiantes se enfrenten al problema de no tener acceso a la información que se encuentra en la web.

1.1. El gran reto: La Permanencia de datos en la web

Los motivos que impiden a los usuarios de la web tener acceso a datos a pesar de tener la información de su acceso, como lo es su dirección URL, pueden deberse a cambios en las políticas de uso, problemas de locación o a fallas técnicas de la red de comunicación o del propio servidor que aloja los datos, así como también el tema de privacidad de datos debido a que muchas veces los usuarios desconocen las políticas de privacidad de datos y como se utiliza la información de los usuarios almacenadas en servidores.

Si alguna de las grandes empresas de alojamiento y/o proveedores de servicios, como Google, Microsoft, Amazon, Dropbox, Rackspace y similares, fallara repentinamente o simplemente decide dejar de brindar servicios, el impacto que esto generaría en los usuarios sería desastroso debido a que toda la información almacenada por estos ya no estará disponible, causando pérdidas importantes.

Esto ha sido la principal fuente de inspiración para el autor del protocolo Interplanetary File System (IPFS) y Filecoin [7], Juan Benet [5]. Para Benet, el

protocolo HTTP [14] presenta vulnerabilidades, como por ejemplo el problema de locación conocido como "location address".

Los problemas de locación ocurren cuando la dirección URL, o enlace http(s), no dirige hacia el sitio correcto de los datos buscados, ya sea porque los datos fueron cambiados de ubicación o porque el servidor que publica los datos en esa dirección no se encuentra en funcionamiento. Una dirección http(s) está formada por el nombre del sitio, o dirección IP donde se encuentran alojados los datos [14]. Si el sitio desaparece o cambia su dirección IP la solicitud de información mediante el enlace http(s) queda inaccesible. Si ese nodo era el único que disponibilizaba la información allí alojada o si no existe un sistema que redireccione automáticamente hacia otro sitio con la información replicada, ocurre que el usuario ve imposibilitado el acceso a la información que buscaba. Esto es lo que sucede en las arquitecturas centralizadas donde la accesibilidad a la información es un punto neurálgico, debido a que todo flujo de información pasa por un único nodo centralizador y si este falla, todo el sistema falla. Sin embargo, en sistemas distribuidos cuando falla un nodo el sistema sigue funcionando con normalidad debido a que existen otros nodos conteniendo réplicas de la información.

No es el caso de la web actual, donde el protocolo HTTP no soluciona el problema de locación. Esto es debido a que la web actual es una web centralizada. Si bien desde hace tiempo se viene trabajando en propuestas para solucionar estos problemas, entre ellos los proyectos ZeroNet [57], MaidSafe [22], Dat [52], Tor [16] cuyos objetivos son construir una web descentralizada y accessible, la propuesta del protocolo Interplanetary File System (IPFS) [4] ha cobrado recientemente una amplia aceptación. El protocolo IPFS propone que los recursos estén disponibles siempre en la web permitiendo al usuario tener acceso a la información independientemente de quien la aloje.
Por otro lado, Blockchain [62] es un buen ejemplo de una tecnología basada en una arquitectura distribuida, que ha innovado en áreas como la salud, educación, finanzas, entre otras. Uno de los problemas que tiene Blockchain es el espacio de almacenamiento que los nodos insumen. Dado que los bloques contienen una copia local de todas las transacciones de la red que integran, a medida que más bloques se unan a la Blockchain el tamaño de estos aumenta considerablemente. Para mitigar este problema, algunos sistemas basados en Blockchain utilizan los servicios descentralizados de almacenamiento que proporcionan IPFS[1], Storj[2], SWARM[3] o Sia[4]. En este trabajo decidimos trabajar sobre la plataforma IPFS por ser una de las mas populares y estables hasta el momento. IPFS es direccionable por contenido, de igual a igual, de código abierto, un sistema de archivos distribuido globalmente que se puede utilizar para almacenar y compartir un gran volumen de archivos

[1] https://ipfs.io/
[2] https://www.storj.io/
[3] https://www.ethswarm.org/
[4] https://sia.tech/

con alto rendimiento [44].

Atendiendo estas dos problemáticas, la de acceso a los recursos de la web y la de optimización del espacio de almacenamiento en plataformas Blockchain, esta tesis presenta un análisis detallado del protocolo de distribución de archivos IPFS (Interplanetary File System) para mitigar estos problemas.

Como caso de estudio se analiza a IPFS dentro del Proyecto ERANet-LAC SELI: "Smart Ecosystem for Learning and Inclusion" [66]. La plataforma SELI es una combinación de arquitecturas centralizadas y arquitecturas distribuidas. Por un lado se tiene una plataforma centralizada por la cual los estudiantes y docentes acceden a los cursos que contiene. Cuando un docente crea un curso y desea subir a la plataforma SELI el material asociado al curso, este proceso está basado en una arquitectura completamente centralizada. Por otro lado para brindar certificaciones digitales, SELI cuenta con una red privada Blockchain Ethereum [9] integrada por servidores de los países Ecuador, Finlandia y Uruguay. Esta red se dedica a crear diplomas digitales a través de *smart contracts*, lo cual le permite a los alumnos obtener una certificación cada vez que culminan un curso en la plataforma.

Con el fin de difundir recursos educativos en SELI nos hemos encontrado con el desafío de almacenar la información del sitio de la forma más óptima posible, teniendo como limitante contar con una cantidad finita de servidores a distribuir y un espacio acotado de almacenamiento. En este escenario es donde el protocolo IPFS toma protagonismo y sobre el cual experimentamos la incorporación de IPFS como protocolo de distribución de archivos.

Si bien HTTP ha ido recibiendo actualizaciones con el paso del tiempo, este es aún vulnerable ante eventuales fallas en servicios. Para resolver el problema de permanencia, IPFS cuenta con una funcionalidad conocida como PIN, el cual indica que un archivo no será eliminado cuando se lo aloja en una red compuesta por nodos IPFS, este proceso se detalla en el Capítulo 3.

1.2. Aplicaciones que usan IPFS

A continuación presentamos algunos trabajos situados en áreas diversas como ser, salud, educación e industria, que proponen una dinámica alternativa de almacenamiento con respecto a sus recursos usando IPFS como principal protocolo de almacenamiento para lograr que los sistemas se adapten de forma eficiente a la demanda digital que crece día tras día.

- **Implementation of Distributed File Storage and Access Framework**

using IPFS and Blockchain [27]. Seleccionamos este trabajo para presentar una de las principales limitantes de la tecnología Blockchain respecto al incremento de espacio de almacenamiento necesario en la cadena de bloques. Como parte de una solución efectiva, los autores proponen un framework que implementa el almacenamiento de las transacciones en una red Blockchain utilizando IPFS como protocolo de almacenamiento.
Según datos reportados en [45], en una red Bitcoin se ocupa aproximadamente 200 GB, creciendo a una tasa de 0.1GB por día lo que causa un fuerte impacto en el espacio de almacenamiento, por tal motivo el espacio de almacenamiento siempre fue un desafío a la hora de analizar la viabilidad del uso de tecnología Blockchain.
Existen técnicas propuestas por varios autores para mitigar este problema de espacio en la Blockchain, las cuales son recopiladas y presentadas en [27]. La primera que mencionan adopta el método donde solo se registran las transacciones que no han sido expiradas aún. Si bien esto reduce el tamaño del nodo, se tiene como desventaja que no hay registro de una parte del historial de las transacciones, lo cual conlleva a que no exista una trazabilidad completa de una transacción.
Otro método adoptado para mitigar el continuo crecimiento de los nodos en la Blockchain propone alojar las transacciones activas en un archivo externo y eliminar las transacciones antiguas de la cadena. Si bien esta técnica reduce el tamaño de los nodos, se elimina una gran parte del historial de la Blockchain. Si bien las técnicas mencionadas logran parcialmente reducir el tamaño de los bloques, gran parte del registro de las transacciones no quedan almacenadas, amenazando así la trazabilidad y transparencia en la red.

La propuesta de los autores en [27], consiste en almacenar las transacciones en una red Blockchain y utilizar IPFS para alojar dicho archivo. Cada bloque contiene un único hash identificado en la red IPFS el cual hace referencia al archivo imagen generado. Cuando un nodo solicita unirse a una cadena de bloques, los mineros deben verificar el formato de las transacciones del nodo, así como también que cumplan con el protocolo de consenso de la red Blockchain. Una vez que se verifican las transacciones se agregan las existentes desde el nodo previo al nuevo nodo solicitante, por lo que los mineros deben resolver una operación compleja como un cripto puzzle[39], calculando el hash del bloque que se va a agregar. Una transacción puede ser de gran tamaño más aun si cada bloque va acumulando transacciones de otros bloques, lo cual hace que ocupen espacio muy rápidamente y el nodo crezca considerablemente en cada transacción que se genera en la Blockchain.
El tamaño de un hash generado en una red que utiliza IPFS es de 46 bytes, por lo que si en vez de registrar la transacción en el nodo, solo registramos el hash que identifica al archivo en IPFS se logra reducir significativamente el tamaño de un bloque.

El hecho de que IPFS reduzca considerablemente el espacio de almacenamiento en una red Blockchain pone en manifiesto que modelos como Bitcoin, Ethereum y otros que sufren la limitante de almacenamiento, consideren incorporar IPFS como una buena opción a la hora de reducir el tamaño de los bloques. Además como presenta [27], IPFS no solo logra una buena performance en cuanto al almacenamiento sino que también, mantiene la inmutabilidad y consistencia de la información, características y principios fundamentales de una red Blockchain.

- **An InterPlanetary File System Based Picture Archiving and Communication System [70]** es un trabajo que seleccionamos para describir el uso de IPFS en el escenario de la Salud. Este trabajo propone la implementación de un framework Hopacs, cuyo objetivo es el almacenamiento y procesamiento de imágenes basada en *edge computing* [25], utilizando IPFS como principal protocolo de almacenamiento. PACS (Picture Archiving and Communication System) [11] es un sistema de almacenamiento de imágenes médicas que utiliza el protocolo DICOM [8] para transmitir y almacenar imágenes.

  DICOM es el estándar para imágenes médicas digitales, por lo que las diferentes estaciones que se comuniquen con otros sistemas PACS deben realizarlo usando el estándar DICOM.
  Hopacs propone al protocolo IPFS como principal protocolo de almacenamiento, siendo Hopacs una alternativa sobre los sistemas tradicionales PACS. La idea de este framework es utilizar la ventaja del protocolo IPFS el cual almacena cualquier tipo de imágenes o archivos y no se limita solamente a un solo tipo de archivo como lo hacen los sistemas PACS actuales.

  En [70] se analiza el desempeño de un sistema PAC tradicional con uno basado en Hopacs, mostrando que la velocidad de transmisión en Hopacs es 35 veces superior a la de un sistema PACS. Los autores señalan que de acuerdo con las estadísticas, mensualmente la cantidad de imágenes almacenadas por sistemas tradicionales PACS en una institución médica (perteneciente al contexto de estudio de los autores) es de aproximadamente 2TB, lo que implica un costo aproximado de 45000 dólares en equipos y mantenimientos de servidores PACS, que se debe invertir cada 4 meses. El artículo muestra además cómo el uso de Hopacs con IPFS permite una reducción de costos en mantenimiento de equipos así como también una mejora en el uso de banda ancha gracias a la incorporación de IPFS como protocolo de almacenamiento y su mecanismo de intercambio de información. Cuanto más nodos integren el framework descentralizado más se observa un decremento en costos de equipamiento.
  En este trabajo se muestra el aporte de IPFS como principal protocolo de almacenamiento, sus ventajas a la hora de intercambiar archivos

y la mejora de desempeño frente a otro protocolo como lo es DICOM. Proponer el uso de IPFS en un sistema descentralizado, donde se intercambia con otros nodos y se almacenan imágenes de calidad como lo son las imágenes médicas, evidencia los beneficios que presenta IPFS así como también la buena performance en este contexto. IPFS se torna una alternativa económica a la hora de administrar los recursos que se requieren para el mantenimiento de imágenes médicas.

- En el contexto de comunicación entre agentes, destacamos el trabajo **Open Peer-to-Peer Systems over Blockchain and IPFS: an Agent Oriented Framework [67]** es un trabajo del escenario de comunicación entre agentes. Lo seleccionamos para mostrar un sistema distribuido de preguntas y respuestas que fusiona la tecnología Blockchain con IPFS. El propósito del trabajo es presentar el diseño y desarrollo de un framework abierto y distribuido. La idea se basa en presentar un sistema de preguntas y respuestas, basado en agentes, quienes mantienen un intercambio de información, cumpliendo distintos roles dentro del sistema. Hay tres roles relevantes en el sistema propuesto: (i) los que auspician como consultores, son los que realizan las consultas,(ii) luego se encuentran los agentes proveedores, aquellos que responden las consultas y por último (iii) los agentes votantes quienes emiten su voto avalando la calidad de la información de las respuestas. Un agente dentro de un sistema abierto y distribuido es considerado como aquel que cumple tres condiciones:

  1. Comparte información dentro del sistema
  2. Consulta la información obtenida (puede ser un chequeo)
  3. Responde a las consultas mediante la información que almacena a nivel local.

  En los sistemas centralizados y federados, se utilizan fuentes fiables que almacenan datos confiables, de los cuales se obtiene la respuesta a las consultas, un ejemplo de ello es el sitio StackExchange [53]. Sin embargo, en un sistema distribuido se torna más complejo que un agente demuestre ser un proveedor fiable de contenido para las consultas realizadas, he aquí que se aborda una temática compleja de resolver en sistemas distribuidos como lo es la confiabilidad de la información.
  Para abordar esta problemática los autores proponen una arquitectura en la cual utilizan una red IPFS para el almacenamiento de información así como también un sistema de puntaje mediante el cual los agentes son calificados según proporcionen datos de calidad.
  Tomando como referencia las reglas en las que se basa el sitio de preguntas y respuestas StackExchange, cada respuesta tiene una reputación asociada, la cual viene dada por la contribución a la comunidad que la misma produce.
  Los autores proponen que cada agente en el sistema, sea modelado como un objeto IPFS, por lo cual los agentes consultores, o sea los que

realizan la consulta, tendrán almacenado su identificación criptográfica según se ha visto como lo modela IPFS, título y descripción de la consulta. Cualquier agente puede contestar a la pregunta realizada por los agentes consultores, pero para convertirse en un proveedor fiable del sistema, se propone que las respuestas sean valoradas por votos.

Cada agente que responde también es modelado como un objeto IPFS, almacenando su hash de identificación, el link merkle a la pregunta que responde y su respuesta. Cada agente que brinda una respuesta a una pregunta, tiene además asociado los votos que califican la calidad de la información. Estos almacenan el voto como un puntaje, el autor el cual responde y el link merkle hacia la pregunta. Si la respuesta es satisfactoria, basta con que un agente votante emite su voto, si la respuesta tiene tres agentes votantes entonces allí es donde se convierte en un proveedor confiable.

Se utiliza Blockchain Ethereum como mecanismo de coordinación entre los agentes que proveen información, los votantes y los que realizan las consultas, de esta forma se asegura la consistencia en la distribución de información y se controla por ejemplo mediante un contrato inteligente que los votantes voten la respuesta una única vez.

La propuesta de los autores [67], presenta un mecanismo de chequeo de información alternativo para un sistema distribuido, fusionando tecnologías como IPFS y Blockchain. Modela a los agentes como objetos IPFS tomando las ventajas de inmutabilidad y almacenamiento de IPFS haciendo referencia a la información en un puntero (merkle link), el cual reduce considerablemente el espacio de almacenamiento de cada agente.

Si bien la propuesta aborda un tema difícil como es comprobar la calidad de información en un sistema distribuido, el artículo presenta un débil boceto del sistema fusionando IPFS y Blockchain, siendo muy escueto a la hora de brindar detalles más certeros por ejemplo, cómo es que realiza la coordinación de los agentes utilizando Ethereum Blockchain. Es un buen ejemplo teórico que muestra el alcance al fusionar tecnologías como IPFS y Blockchain, pero no se presenta como una solución global observando que para ciertos contextos los sistemas centralizados logran un mejor desempeño.

Sin embargo, el trabajo [67] propone un sistema de intercambio de preguntas y respuestas descentralizado, motivando un abordaje de solución distribuida para SELI que puede resultar una incorporación valiosa para el proyecto, permitiendo generar un espacio de intercambio entre los usuarios de la plataforma donde se pueda consultar información así como plantear dudas sobre los cursos que se dictan en la plataforma. Es un sistema que puede enriquecer y motivar el uso de la plataforma

SELI, incentivando también a los usuarios a familiarizarse con el uso de IPFS.

- **Towards a Decentralized and Distributed Framework for Open Educational Resources based on IPFS and Blockchain [41]** es un trabajo que seleccionamos para describir uno de los usos de IPFS en el escenario de la Educación. El trabajo aborda el problema de gestió y adopción de Recursos Educativos Abiertos (REA).
  El concepto de REA, fue adoptado por la UNESCO en el año 2002 en un seminario sobre el uso de materiales didácticos en educación superior [10]. La idea es brindar REAs que sean accesibles y cumplan con las características de ser reutilizables, actualizables y distribuidos bajo alguna licencia idónea. En la actualidad, la mayoría de los recursos educativos abiertos son almacenados en sistemas centralizados, siendo estos controlados por el propietario del servidor el cual los almacena.
  Tradicionalmente para generar un REA se atraviesan dos etapas, creación y adopción. En la etapa de creación de un REA, los expertos en el contenido envían el material a la plataforma donde se almacenará y compartirá con el resto. Una vez que el recurso ya se encuentra en la plataforma, pasa a etapa de adopción en la cual se lo edita para ser examinado, se evalúa la calidad del contenido y se mejora en caso de ser necesario. Ante este flujo de información, algunos de los problemas actuales que enfrentan los sistemas de almacenamiento de REAs son: dificultad en el rastreo y seguimiento de los recursos, autenticidad del contenido, problemas con las licencias así como también atribuciones de autoría.
  Para mitigar estas dificultades, el trabajo [41] se propone implementar un framework descentralizado utilizando contratos inteligentes [54] en una red blockchain Ethereum e IPFS para el almacenamiento de los REA. La propuesta consiste en dividir en dos etapas la suba y generación de los REA. Hay dos actores principales: los proveedores de los REA y los consumidores. En la primera etapa los proveedores utilizan IPFS para almacenar los recursos, una vez almacenados estos se les devuelve un hash que los identifique en la red IPFS. Con esto se asegura la inmutabilidad del recurso solo por el hecho de pertenecer a la red IPFS. En la segunda etapa se comienza con la verificación del recurso ingresado, para ello deberán proporcionar el hash que se obtuvo en la primera fase, allí se lo verifica y en caso de ser necesaria alguna corrección lo vuelven a subir a la red IPFS. Cuando es validado el recurso, el proveedor ejecuta un contrato inteligente en la red IPFS, utilizando los criterios de la licencia Creative Common. Una vez que el material es aprobado y subido a IPFS, se genera un contrato inteligente para incorporar en la blockchain Ethereum, lo cual mediante un mecanismo de consenso cada nodo es responsable de verificar y controlar la calidad de la información a través de la ejecución de un contrato inteligente.

> Si bien aún el framework está en su etapa inicial de desarrollo y los autores no brindan detalles de la implementación ni profundizan mucho sobre el proceso de verificación de licencias, el objetivo se centra en aprovechar los beneficios de fusionar el potencial que brinda tanto IPFS como ethereum blockchain. Aprovechando la ventaja de la red Ethereum la cual a través de contratos inteligentes se pretende verificar y validar los recursos, para así mitigar la falta de control en cuanto a autoría del material, contenido, calidad y licencias.

Si relacionamos estos escenarios con el escenario que hemos planteado en SELI, destaca la utilidad de considerar el uso de IPFS como una alternativa más económica y eficiente a la hora de aprovechar los espacios de almacenamiento del sistema. En especial, en el escenario propuesto en SELI, donde se configura un IPFS cluster, lograr aprovechar mejor los recursos disponibles de las instituciones educativas evitando costear nuevos equipos, hace que IPFS se vea como una buena alternativa para mitigar este problema.

A pesar que no todos los artículos seleccionados trabajan en el contexto de educación estudiado en esta tesis, el problema es el mismo: llevar un sistema centralizado a uno distribuido, para así mejorar el desempeño, aumentar la capacidad de almacenamiento, disponibilidad de la información y mejorar el uso de banda ancha, así como también tratar de bajar los costos de mantenimiento de los equipos utilizando al máximo los recursos de cada nodo que integra el sistema.

Considerando entonces el proyecto SELI, utilizar IPFS y contratos inteligentes sobre una red Ethereum es una alternativa a tener en cuenta a la hora de validar autenticidad, calidad y licencia de un REA en una versión descentralizada de SELI (D-SELI).

Por otro lado, apostar al cambio en la dinámica de interacción que promueve IPFS, creando una web descentralizada, a través de proyectos como Filecoin, proponen un mercado de almacenamiento en una red descentralizada creando por primera vez un contrato entre el cliente y el agente. Según Benet los usuarios cuentan con recursos que hoy en día no son aprovechados, por ejemplo espacio en disco duro que se puede aprovechar ofreciéndolo para almacenamiento y así obtener un incentivo económico, tal cual propone el proyecto Filecoin.

1.3. Resultados de esta tesis

En el proyecto SELI [66], uno de los problemas a abordar es la limitante de espacio que presenta el servidor central. Si bien es una arquitectura centralizada, la problemática es la misma que la presentada en la selección de trabajos anteriores: abordar el tema del almacenamiento ante la gran demanda de recursos a ser almacenados de forma de asegurar siempre su accesibilidad.

SELI es un proyecto implementado en React y NodeJS, que utiliza Meteor [20] como framework de desarrollo. Esta tesis presenta un análisis de ventajas y desventajas de 2 escenarios de SELI: peer-to-peer y cliente-servidor. A partir del trabajo de análisis realizado en ésta tesis se propone una forma de integrar en SELI un clúster IPFS [36] y se presenta una posible extensión de la arquitectura Blockchain de SELI para uso de IPFS totalmente distribuido, que llamamos D-SELI. Se presenta además un estudio de comportamiento del uso de Meteor y de IPFS.

1.4. Organización de este documento

La organización del resto de este documento es la siguiente.

Capítulo 2 presenta los conceptos base de las arquitecturas centralizadas y distribuidas.

Capítulo 3 presenta el diseño y arquitectura del protocolo IPFS y su mecanismo de distribución de archivos.

Capítulo 4 presenta la influencia de tecnologías como Blockchain e IPFS en el contexto de educación analizando al proyecto Filecoin [7], cuyo objetivo es crear un mercado de almacenamiento en el cual los usuarios son libres de elegir dónde y cómo almacenar sus datos sin intermediarios de por medio.

Capítulo 5 presenta el análisis de IPFS en la plataforma Blockchain educativa SELI.

Capítulo 6 presenta algunas conclusiones y trabajos futuros.

Capítulo 2

Conceptos de base

Para abordar el problema de permanencia de datos, hay que tener presente cuál es el tipo de arquitectura en el cual se basa el diseño del software, si estamos ante una arquitectura centralizada o si estamos ante una arquitectura distribuida. Si un sitio web se encuentra basado en una arquitectura centralizada y deja de estar en línea inhabilita la información a los usuarios por tiempo indefinido. ¿Pero qué sucede si esta misma aplicación se diseña con una arquitectura distribuida?

En este capítulo presentamos en la Sección 2.1 los conceptos bases de las arquitecturas distribuidas y centralizadas y algunos trabajos que muestran medidas sobre las distintas arquitecturas (tiempo de distribución de archivos y performance en sistemas de streaming). Luego, en la Sección 2.2 presentamos los conceptos de base de la arquitectura distribuida de Blockchain.

2.1. Arquitecturas centralizadas vs. distribuidas

Las aplicaciones basadas en arquitecturas distribuidas han ido ganando popularidad, en especial en estos últimos años. Estas presentan grandes diferencias con las arquitecturas de software convencionales basadas en la centralización de información y no en la distribución.

Las arquitecturas centralizadas son aquellas que centralizan su información en un nodo, mientras que en las distribuidas como lo es P2P, poseen una infraestructura basada en nodos conectados entre sí, sin tener una entidad centralizadora que regule la información.

Una red P2P está compuesta por lo que se conoce como “pares”, estos componen una pareja de *hosts* que se conectan entre sí. A su vez estos pares están conectados con otros conocidos como “vecinos” o “neighbours”. A una red P2P se la visualiza con un grafo conexo donde sus nodos adyacentes son sus vecinos y cada nodo puede ser cualquier dispositivo de almacenamiento, no hay una jerarquía asignada, todos ofician de cliente y servidor a la vez.

En el caso de las arquitecturas descentralizadas, hay más de un nodo central almacenando información. La información se aloja en diferentes nodos distribuidos según zona geográfica, ejemplo de ello lo muestra la Figura 2.1.

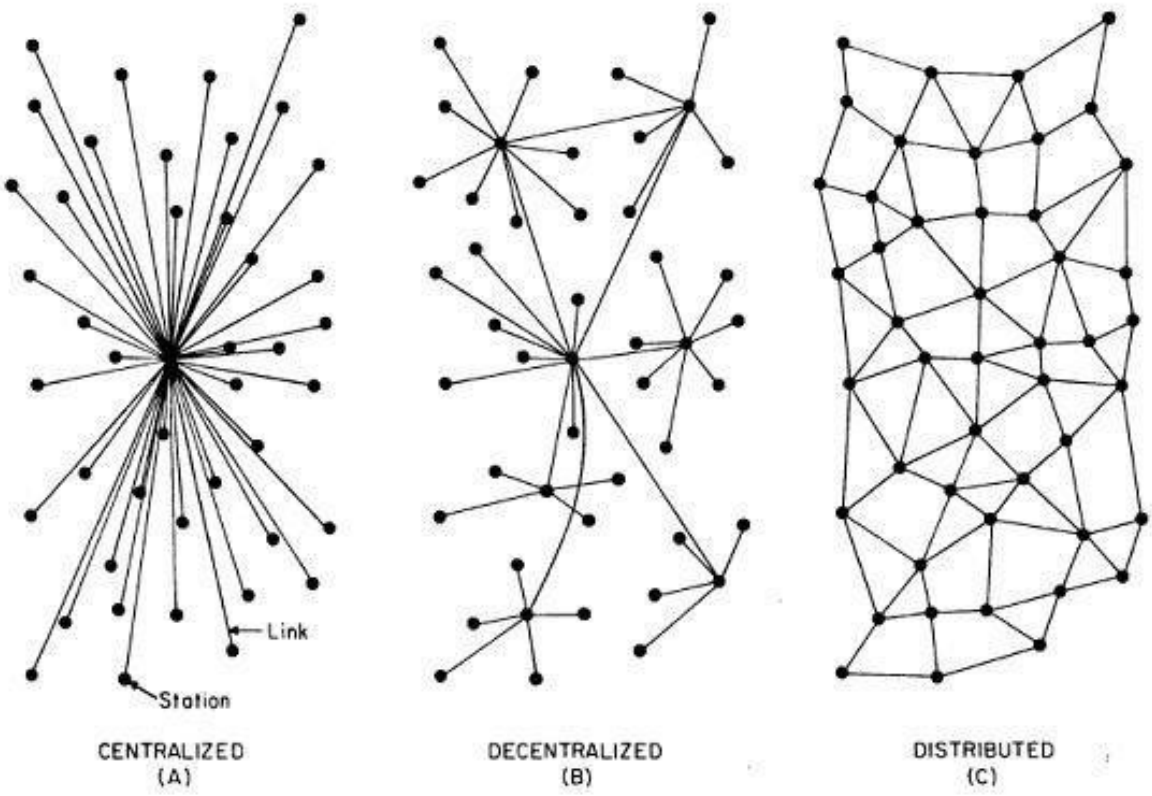

FIGURA 2.1: Ejemplos de arquitecturas para aplicaciones centralizadas, descentralizadas y distribuidas [62]

En las arquitecturas distribuidas la descentralización se da en el caso de que la información se encuentre distribuida en los múltiples nodos que conforman la red, si falla un nodo el resto responde ante la posible falla.

Las arquitecturas descentralizadas son identificadas por Raval en [63] como un campo emergente donde se investigan nuevos modelo y donde hay diferentes puntos de vista para definir el concepto de aplicación descentralizada. Para muchos desarrolladores incluido el propio autor de IPFS Juan Benet [4], para que una aplicación sea descentralizada debe cumplirse un único requisito, el de no ser "Single Point of Failure" (SPF), lo cual implica responder de forma inmediata ante la eventual falla de un nodo sin que el sistema quede inoperante.

Para visualizar las diferencias entre las arquitecturas centralizadas y distribuidas presentamos a continuación dos casos de análisis de desempeño, el primero propuesto por Kurose en [28] el cual realiza el cálculo del tiempo de distribución de un archivo entre N nodos con arquitecturas centralizads y distribuidas. El segundo, un sistema de streaming [50] basado en una arquitectura distribuida en el cual los autores comprueban que el sistema logra ahorrar un 60 % de banda ancha en comparación con un sistema centralizado como lo es por ejemplo YouTube. En ambos casos se muestra que bajo ciertas condiciones los sistemas distribuidos logran ser más eficientes frente a los

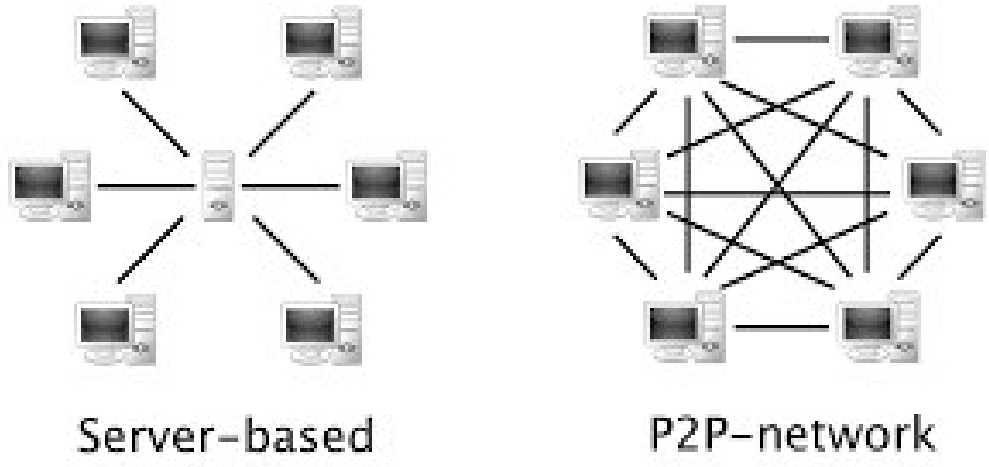

FIGURA 2.2: Arquitecturas Centralizadas y Distribuidas [64]

sistemas centralizados (Figura 2.2 presenta gráficamente ambas arquitecturas).

2.1.1. Distribución de archivos

En esta sección presentamos el análisis, realizado por Kurose en [28], del tiempo de distribución de un archivo en una arquitectura centralizada y en una distribuida.

Kurose propone en [28], un modelo cuantitativo de N pares en una red P2P, siendo F el tamaño del archivo a distribuir y velocidad (u) de carga del enlace y velocidad (d) de descarga del enlace.

En una arquitectura centralizada, cada nodo realiza una petición de descarga de archivo a un servidor central y éste devuelve una copia del mismo. Distinto es el caso en una arquitectura P2P donde cada nodo actúa como cliente - servidor distribuyendo el archivo sin la intervención de un servidor central.

En [28] se calcula el tiempo de distribución de un archivo en N nodos, para ello se divide en dos escenarios, el primero calcula Dcs (tiempo de distribución en una arquitectura cliente-servidor) y el segundo escenario calcula DP2P(tiempo de distribución en una arquitectura P2P).

En el primer escenario el servidor recibe N solicitudes de descarga de un archivo de tamaño F, por lo que debe transmitir NF bits. La velocidad de carga del servidor es us, el tiempo que demora en distribuir el archivo es NF/us. Sea dmin la mínima velocidad de descarga de un nodo, entonces el tiempo total que insume descargar F bits es F/dmin. Por lo que se obtiene Dcs= max

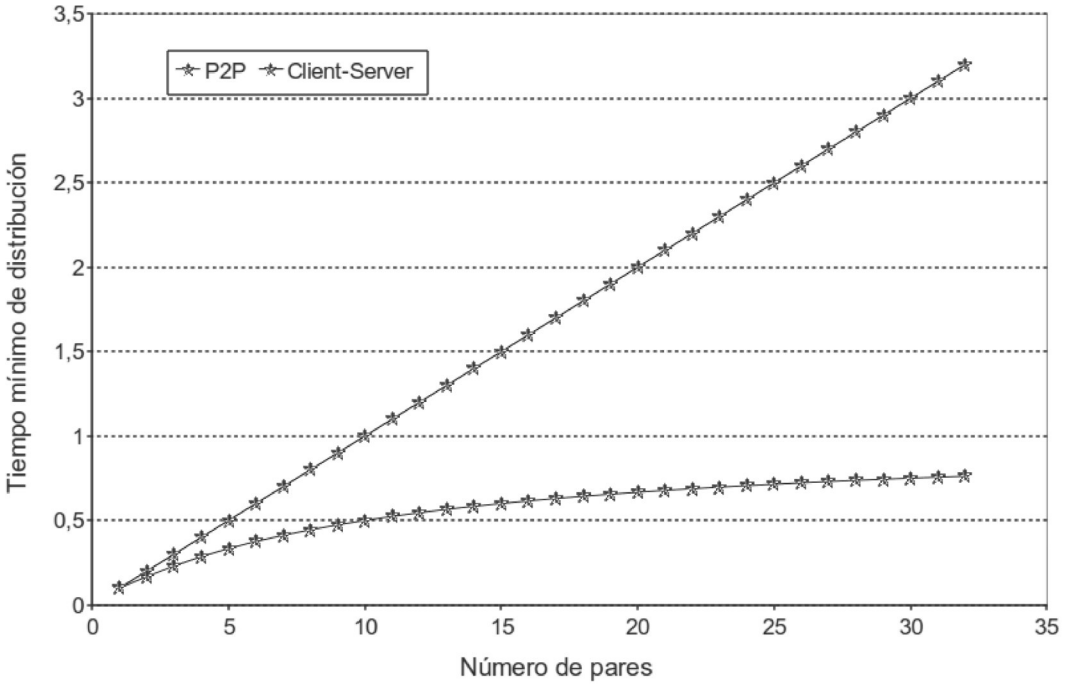

FIGURA 2.3: Comparación cliente-servidor y P2P [28]

NF/us,F/dmin , siendo NF/us el tiempo de distribución del archivo para una arquitectura cliente-servidor.

Para el segundo escenario se calcula DP2P, donde cada nodo de la red coopera en la distribución del archivo. Se considera a F/us como el tiempo que demora en descargarse por primera vez el archivo del servidor a un nodo de la red P2P. Si consideramos la capacidad de carga del sistema como un todo, este es igual a la velocidad de carga del servidor más las velocidades de carga de cada nodo que compone la red. De lo anterior se reduce a utot= u1+ u2+ ...+uN, el sistema debe distribuir el archivo en N nodos por lo que el tiempo de distribución total es igual a NF/utot.

En síntesis DP2P= max F/us,F/dmin,NF/utot, el gráfico de la Figura 2.3 compara el tiempo mínimo de distribución de un archivo en ambas arquitecturas P2P y cliente-servidor suponiendo que todos los nodos tienen la misma velocidad de carga u.

Con estas medidas Kurose muestra cómo las arquitecturas P2P en comparación con las arquitecturas cliente-servidor son más veloces en el contexto propuesto. La gráfica de la Figura 2.3 muestra una clara tendencia en el aumento del tiempo de distribución ante el inminente crecimiento de los nodos en una arquitectura cliente-servidor, mientras que lo opuesto sucede para una arquitectura P2P.

Tanto las arquitecturas distribuidas como las centralizadas pueden ser más o menos eficientes según el contexto en el cual se aplica cada una de

ellas. Hay sistemas distribuidos y centralizados que resuelven un mismo problema, en el cual generalmente hay uno que lo resuelve más eficiente que el otro. El caso de distribución de archivos es un ejemplo de ello, otro ejemplo son los sistemas de transmisión en vivo en los cuales se busca aprovechar al máximo la banda ancha de la red. En la sección siguiente se presenta un trabajo que muestra como un sistema P2P-VDN de distribución de videos basado en una arquitectura P2P es más performante para ciertos escenarios frente a una plataforma de streaming (transmisión) centralizada como lo es YouTube. [50]

2.1.2. Performance en sistemas de streaming: YouTube vs P2P-VDN

YouTube [69] es una de las plataformas más populares de streaming de vídeos basada en un modelo cliente-servidor centralizado. A comienzos del año 2008 la plataforma ya contaba con alrededor de 79 millones de usuarios [68], de los cuales tres billones de videos habían sido vistos para ese entonces. Al ser una plataforma centralizada, todos los videos pasan por un nodo central, haciendo que ese nodo de YouTube tenga el control y manejo de los videos que se almacenan en su plataforma. Al aumentar la cantidad de usuarios, que se unen en grandes números a diario, hace a la plataforma YouTube poco escalable para ofrecer un buen servicio de streaming [56].

Uno de los puntos débiles de los sistemas de streaming centralizados, es la limitada capacidad de banda ancha de los servidores, viéndose afectada la calidad de servicio debido a que los servidores no cuentan con la la capacidad para atender las infinitas solicitudes de los usuarios ante la demanda requerida por mirar un video en transmisión real en un instante dado.

Por otro lado, P2P-VDN es un sistema de distribución de videos basado en una arquitectura P2P. La principal ventaja que presenta el sistema P2P-VDN es la performance de banda ancha frente a sistemas de streaming centralizados como YouTube. En [50] los autores demuestran un 60 % de eficacia en el uso de banda ancha utilizando el sistema de streaming P2P-VDN. El diseño de P2P-VDN se basa en tres puntos claves, dispersión de datos, reabastecimiento de datos y en el protocolo de transmisión de diversidad de ruta.

Cuando un video es publicado por un usuario en P2P-VDN se utiliza la técnica de dispersión de datos la cual se basa en particionar los videos en múltiples piezas (chunks) k y distribuirlas entre los nodos de la red P2P. La diferencia con publicar un video en YouTube es que allí se almacena el video en un servidor centralizado, mientras que el P2P-VDN se dispersan las múltiples piezas del video entre los nodos que componen el sistema. Estas k piezas son codificadas utilizando la técnica RCN (random network coding) la cual las distribuye de forma aleatoria entre los usuarios de P2P-VCN. Cuando un usuario solicita el vídeo dentro de P2P-VDN, se le consulta al nodo más cercano al usuario utilizando el protocolo Chord [65] para comenzar con el

proceso de recuperar las piezas distribuidas para así obtener el video original y poder descargarlo o mirarlo desde la plataforma.

Debido a que en una red P2P los nodos entran y salen constantemente de la red, los autores en [50] observan que durante un determinado período de tiempo, no existían suficientes nodos que contengan las piezas del video en cuestión provocando que el usuario no pueda completar su solicitud. Esta situación da paso al segundo punto clave de P2P-VDN, el reabastecimiento de datos. Cuando un nodo decide abandonar la red P2P, este les envía las piezas almacenadas a otros nodos para que la información no se pierda si el nodo se va de la red. Sin embargo en un sistema P2P de videos las piezas a transferir tienden a ser de gran tamaño lo que hace que muchas veces se vean interrumpidas o los nodos abandonan la red antes de que esta transferencia se complete. Por tal motivo en P2P-VDN los autores en [50] proponen mitigar esta situación de forma que la información no se pierda mediante un mecanismo de abastecimiento de datos.

En [50] se asume que en una red P2P mientras un nodo se va otro nodo se une en un corto periodo de tiempo con espacio suficiente para almacenar los datos del nodo que abandona la red, asumiendo el rol del nodo que deja la red. La robustez del sistema consta en la probabilidad de recuperación de datos sea cada vez mayor.

Los autores en [50] plantean dos proposiciones en las cuales se basa la eficiencia del sistema P2P-VDN, la primera afirma que si el reabastecimiento de datos depende de que un nodo nuevo en la red copia datos desde dos nodos aleatorios, entonces el promedio de los datos abastecidos no podrán ser recuperados en un O(n2) siendo n el número de nodos participantes durante el proceso de abastecimiento de datos.
La segunda proposición es que si el reabastecimiento de datos sigue la técnica RNC utilizando dos nodos para generar los datos,entonces el promedio de números de respaldo es de O(2[̂]n), siendo n los nodos participantes en el proceso de respaldo.

Estas dos proposiciones muestran que el mecanismo de abastecimiento de datos es más performante que la técnica aleatoria cuando se trata de redundancia y respaldo de datos.

Por último P2P-VDN utilizado en sistemas centralizados, donde las solicitudes entre el servidor y el usuario pueden experimentar grandes congestiones en la red, produce que el streaming tienda a perder la calidad. Para sobreponer esto P2P-VDN utiliza el protocolo “recuperación de ruta”, la técnica se basa en que distintas partes del video se descargan de diferentes nodos en simultáneo en distintas rutas.

Con el fin de probar que la arquitectura propuesta para el sistema P2P-VDN tiene una buena performance, en [50] se realiza una simulación entre

sistemas de streaming, comparando un Non-NC y uno RNC. Los resultados obtenidos muestran que en P2P-VDN se aprovecha el uso de ancho de banda de hasta un 60 % con respecto al esquema tradicional.

Los dos casos anteriores presentados respecto a la distribución de archivos basados en modelos P2P frente a un modelo centralizado, muestran que en ciertos escenarios resulta más performante aplicar arquitecturas P2P, pues logran aprovechar mejor los recursos aplicando técnicas más eficientes que los sistemas tradicionales centralizados.

En la siguiente sección presentamos las características principales de una arquitectura distribuida como lo es Blockchain.

2.2. Blockchain: Arquitectura distribuida

Desde el 2008 con el auge de Bitcoin [45], blockchain ha recibido una gran atención, ganando popularidad como la tecnología que ha hecho funcionar a Bitcoin. Sin embargo, blockchain es también ampliamente usado en los contextos educativos. Esta sección presenta los conceptos de la tecnología blockchain necesarios para entender su aplicación en la plataforma de educativa SELI presentada en el Capítulo 4.

Blockchain es una tecnología basada en una arquitectura P2P, que puede ser pública, privada o de consorcio [71]. Una red blockchain se compone por un conjunto de nodos interconectados entre sí, viéndose como una base de datos distribuida en la cual cada nodo contiene una copia exacta de todas las transacciones que se realizan en la red.

A cada nodo en una red blockchain se lo denomina bloque. Toda blockchain contiene un bloque inicial, el cual debe ser el mismo para todos los bloques que conforman la red. Al bloque inicial se lo conoce también bajo el nombre “bloque génesis” o “bloque cero”, todos los nodos que quieran unirse a la red deben unirse al mismo bloque génesis.

Generalmente a una red blockchain a menudo se la visualiza como una pila (stack) de bloques donde cada bloque está enlazado al anterior [2]. La identificación de cada bloque se almacena en un campo dentro del mismo, a su vez cada bloque almacena la referencia al bloque que lo antecede. Para establecer el enlace con el bloque predecesor, se examina el encabezado del bloque al que se quiere enlazar, este contiene los campos que lo identifican, entre estos se encuentra la dirección (o "hash") del bloque previo, conocido como bloque padre, por lo que el bloque se enlaza al hash que es identificado en el campo “previous block hash”. De esta forma queda enlazado el nodo en la red.

La Figura 2.4 muestra la arquitectura de una cadena de bloques que integran una blockchain, esta se compone de un cabezal (block header) y un

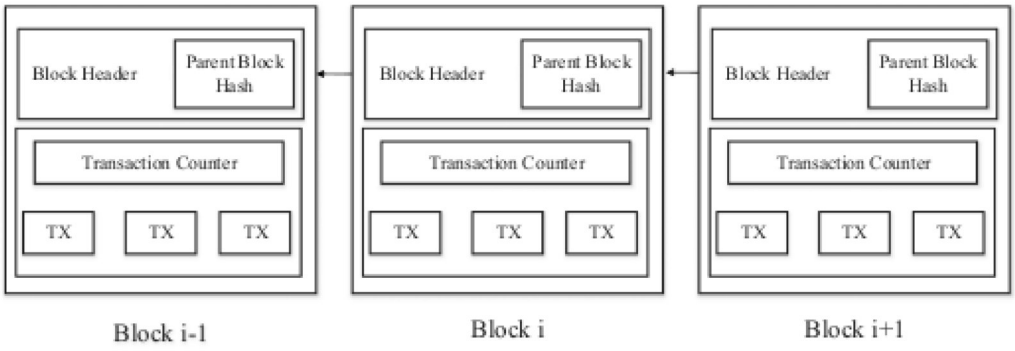

FIGURA 2.4: Cadena de bloques en una red Blockchain [71]

cuerpo (block body). En el block header se almacenan el identificador del bloque, el identificador del bloque anterior, fecha en la cual las transacciones del bloque han sido validadas (timestamp), mientras que los datos de todas las transacciones registradas en la blockchain se alojan en el cuerpo del bloque.

Las transacciones son validadas por bloques conocidos como *mineros*, cada vez que un bloque desea unirse, los mineros son los encargados de crear el bloque, enlazarlo a su predecesor y a su vez validar las transacciones en toda la red.

Una red blockchain se caracteriza por ser inmutable, de consenso porque los bloques adoptan un protocolo de acuerdo común al unirse a la blockchain y transparente dado que las transacciones deben validarse por todos y cada bloque almacena una copia local de las transacciones aprobadas [71].

Cada transacción es almacenada en una estructura conocida como "Merkle Tree" [44]. Los Merkle Tree son utilizados para verificar la integridad de los datos que se almacenan, de ahí el concepto que la red blockchain es inmutable. Ningún bloque puede alterar las transacciones luego de que estas hayan sido registradas en la blockchain.

Capítulo 3

Interplanetary File System (IPFS)

Interplanetary File System (IPFS) es un protocolo de intercambio de archivos, basado en una arquitectura distribuida peer-to-peer, que permite tener un servicio de almacenamiento de datos descentralizado. Su creador Juan Benet impulsó este proyecto de libre distribución de archivos cuyos motivos principales son la libertad de expresión sin censura, la importancia de transformar la web en una web más descentralizada así como también preservar el historial de información de los usuarios mediante un mecanismo que garantice la permanencia de recursos en la web. En el seminario llevado a cabo en la Fundación Long Now en San Francisco,(2018) [48], Benet enumera los problemas aún sin resolver en la web, como ser el mal funcionamiento en *IoT* y *mobile*, la falta de conectividad ante la falla de un nodo, la censura por parte de las empresas de software, las personas que están limitadas al acceso a internet y la vulnerabilidad a ataques informáticos que presenta la web.

Para mitigar estos problemas mencionados anteriormente Benet propone IPFS como una alternativa que desafía los problemas de la web actual haciendo de esta un sitio robusto, fiable, descentralizado y libre de políticas de censura de información. IPFS es un proyecto que continúa en desarrollo y como todo proyecto de software libre invita a todos los desarrolladores e investigadores a colaborar en él para así crear un sistema eficiente y óptimo que cambie la dinámica de almacenamiento de información que existe hoy en día.

En la Figura 3.1 se muestran dos sistemas, por un lado un sistema centralizador como lo es google drive, donde los usuarios almacenan y descargan sus archivos en un servidor de Google. Por otro lado, se observa la distribución de archivos en un sistema P2P donde cada usuario oficia de servidor sin ningún tipo de nodo centralizador. Si Google dejase de funcionar en un lapso de tiempo, el sistema centralizador dejaría sin acceso a la información a sus usuarios, tal como sucedió en el episodio relatado en [43] dejando a usuarios sin tener acceso a sus cuentas así como a empresas, instituciones entre otros sin poder operar hasta que el servicio pueda restablecerse.

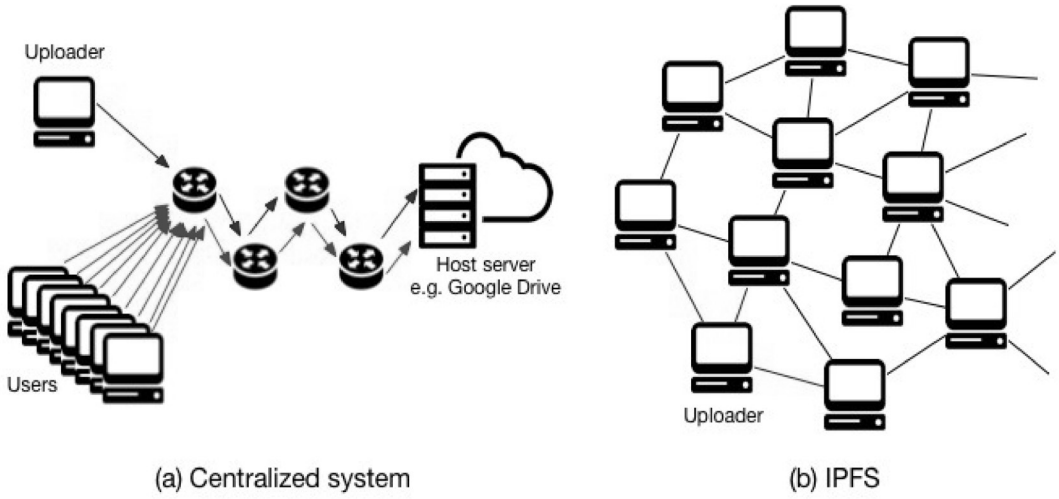

FIGURA 3.1: IPFS vs sistema centralizado [19]

3.1. Diseño IPFS

IPFS propone una alternativa a los problemas que acarrean los sistemas centralizados, buscando conectar todos los dispositivos a un sistema de archivos. Ningún nodo que compone una red IPFS tiene privilegios sobre otros nodos, todos los nodos comparten y distribuyen datos entre sí, no existe un nodo centralizador de información. Los datos son almacenados en una estructura conocida como "Merkle Trees" [44] en el cual se pueden construir sistemas versionados, blockchains e incluso webs permanentes. IPFS utiliza la combinación de BitTorrent, Github y Kademlia [58].

IPFS se basa en el principio de búsqueda por contenido de archivo, no por su nombre, título o locación [58]. Cuando un archivo es alojado en una red IPFS se divide en bloques de 256 KB cada uno de estos se someten a un proceso donde son identificados según el contenido çontent identification"(CID), básicamente el CID es un hash el cual se obtiene basado en el contenido del archivo únicamente. Este CID es único e inmutable asegurando que cada bloque no podrá ser modificado, al basarse en el contenido del archivo si este se modifica se tendrá un nuevo CID asignado. Esta primera etapa en la cual se somete un archivo cuando se lo agrega en una red IPFS se conoce como "chunking". En una primera instancia cada archivo mayor a 256 KB se fracciona en bloques, de allí su nombre.

Como mencionamos IPFS se basa en el contenido del archivo y no en su locación, cuando agregamos un archivo a una red IPFS nos devuelve un hash como muestra el ejemplo en la Figura 3.2.

Existen múltiples algoritmos de encriptación sha1, sha2-256, sha3-256, sha3-512, shake-256, keccak-512, blake2b-160, entre otros. Por defecto IPFS

```
$ ipfs add mytextfile.txt
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy mytextfile.txt
```

FIGURA 3.2: Archivo IPFS

FIGURA 3.3: Multihash

utiliza el algoritmo sha2-256, pero puede soportar otros algoritmos que quieran ser utilizados. Para poder tener esta opción se creó la estructura MULTIHASH [60] el cual se encuentra compuesto por:

- El primer campo es el código del algoritmo que se utiliza para la encriptación, en el caso del algoritmo sha2-256 el código es 18.
- Luego sigue el largo del hash, en caso de sha2-256 es 256 bits lo cual tiene un equivalente de 32 bytes.
- Por último el valor del hash actual.

El CID se representa como un string que encapsula la combinación de un formato Multihash y base encoding[34]. En la primera versión de IPFS se utilizó base58btc y formato Multihash, siendo una representacion de ello: "QmY7Yh4UquoXHLPFo2XbhXkhBvFoPwmQUSa92pxnxjQuPU", generalmente la primera versión comienza con Qm, haciendo referencia a una version 0 de CID, luego evolucionó a versión 1 comenzando con el prefijo "bafy". Por el momento IPFS solo tiene dos versiones de CID, versión 0 y 1.

3.2. MERKLE DAG

Las estructuras MERKLE [44] son estructuras de datos de tipo árbol, donde cada nodo que tiene hijos está etiquetado con el hash producto de la concatenación de los hijos de este y las hojas son los bloques de contenido. De esta forma se tiene múltiples datos producto de pertenencia a un mismo hash.

En la Figura 3.4 se muestra un ejemplo de árbol merkle, el cual básicamente es un árbol binario donde cada hoja es identificada por una función de hash y el nodo que apunta a estas hojas es identificado por una función de hash resultado de la combinación de las funciones hash de ambos nodos hijos y así sucesivamente se va formando el árbol hasta llegar a la raíz la cual es la que identifica a toda la estructura. A modo de ejemplo en el caso de la Figura 3.4 el nodo que se identifica como Hash 0, resulta de la combinación

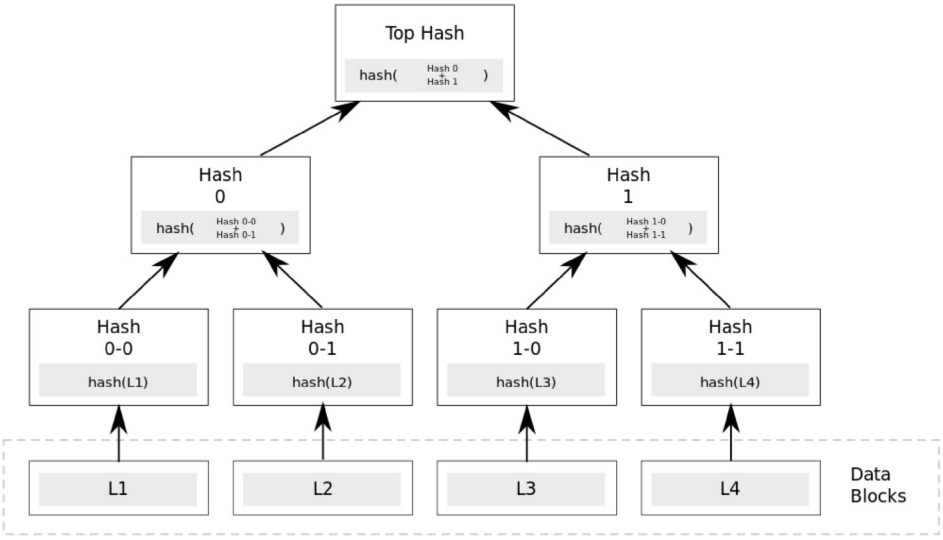

FIGURA 3.4: Estructura de datos Merkle Tree [34]

del nodo Hash 0-0 y Hash 0-1. Lo mismo sucede con el nodo Hash 1, resultante de la combinación de Hash 1-0 y Hash 1-1, luego Top-Hash combina Hash 0 y Hash 1.

Este tipo de estructura de datos cumple con ciertas características que hacen que sean fiables a la hora de utilizarlas.

Una de las principales características es la seguridad que este tipo de estructuras brindan. Debido a que cada nodo que compone un Merkle Tree tiene un *hash* que lo identifica de forma única, si hay alguna alteración en algún bloque de datos se debería actualizar toda la estructura del árbol incluso la raíz, generando un identificador completamente distinto, esto conlleva a que la estructura de datos quede invalidada. Por esta razón al realizar cualquier cambio en el contenido de un bloque de datos, se vuelve a generar un identificador diferente construyendo otra versión del archivo.

Los *hashes* que identifican a los nodos son los que determinan a toda la estructura de un Merkle tree, resulta casi imposible alterar los datos sin que de inmediato se produzca un efecto en cadena que termina por modificar la propia raíz que se utiliza para verificar la integridad de todo el bloque. Estas estructuras de almacenamiento son muy utilizadas en sistemas como Git, IPFS, BitTorrent y Blockchain entre otros.

3.3. Características del intercambio de archivos en IPFS

IPFS está diseñado para ser multitransporte, es decir, se adapta a cualquier protocolo de transporte de una red [6] o sistema de direccionamiento por ejemplo a TCP, UDP, uTP, entre otros. Si bien en términos generales TCP/IP [24] son protocolos confiables por las cuales las aplicaciones los utilizan para garantizar al usuario que los datos naveguen por la red y lleguen todos de forma ordenada y segura, no todas lo utilizan alegando que muchas veces TCP agrega sobrecarga, por lo que se decide utilizar protocolos menos confiables pero más simples como por ejemplo UDP

Libp2p [61] es un sistema modular formado por un conjunto de protocolos y librerías el cual provee una interfaz que se adapta a cualquier protocolo de transporte. A pesar de que libp2p fue diseñado durante la implementación de IPFS, se puede adaptar a cualquier proyecto por su diseño modular y componible. En una red P2P un nodo acepta más de una conexión a la vez, por ejemplo puede aceptar conexiones TCP, en simultáneo con conexiones websocket de otros nodos que se ejecutan en un navegador web. En libp2p el responsable de administrar los transportes se denomina conmutador (en inglés: switch, de ahora en adelante nos referimos como switch), quien se encarga de coordinar la negociación del protocolo, la multiplexación de flujos y el establecimiento de comunicaciones seguras. El switch proporciona un único "punto de entrada"para escuchar y realizar conexiones entre otros nodos, lo cual hace que soporte múltiples protocolos de transporte.

Confiabilidad

IPFS proporciona confiabilidad a pesar que las capas inferiores a este no lo hagan. Utilizando protocolos tales como uTP y SCTP. El protocolo uTP [1] tiene algunas diferencias con el de TCP, fue escogido por sobre TCP por la misma razón que BitTorrent lo hizo en su momento. TCP tiene la característica de dar a sus pares el mismo ancho de banda a todos por igual, esto conlleva a ciertos inconvenientes. En el caso de BitTorrent el usuario debe limitar el ancho de banda configurando el límite de carga y descarga para no utilizar toda la banda ancha, teniendo así que conocer su propia conexión a internet. Esta situación muchas veces lleva a realizar configuraciones poco óptimas, este es el principal motivo por lo cual BitTorrent creó uTP siendo que detecta automáticamente el espacio libre de banda ancha y lo ajusta de forma automática. IPFS adaptó este protocolo como confiabilidad de conexión entre sus pares. Además uTP tiene un mecanismo de control de congestión para evitar la pérdida de paquetes como suele suceder con UDP que no lo tiene. El control de congestión lo realiza a través de LEDBAT[26], un algoritmo de control de congestión que busca utilizar el ancho de banda disponible en una conexión de extremo a extremo entre dos nodos mientras limita el consiguiente aumento de la demora en la cola de dicha conexión.

Conectividad

Libp2p resuelve entre otras funcionalidades la conectividad entre los nodos en una red P2P. La idea es adaptar técnicas ya utilizadas en otros sistemas de red, y poder resolver los conflictos que se presentan cuando dos nodos quieren establecer una conexión. En un sistema central generalmente eso lo resuelve el centralizador aplicando distintos protocolos que logran resolver la conectividad, en el caso de P2P al no tener un punto central cada nodo debe tratar de establecer conexión adaptándose a la red en cuestión.

IPFS utiliza la técnica de ICE-NAT [23] para realizar la conexión en una red P2P. NAT es una técnica que mapea direcciones privadas IP a una única dirección IP pública. Si bien NAT suele ser transparente para las conexiones salientes, la escucha de las conexiones entrantes requiere cierta configuración. El enrutador (router de ahora en más) escucha en una única dirección IP pública, a través de una tabla de mapeo alojada en el router enruta a direcciones IP privadas de una red interna que pueden manejar la solicitud. Para atender solicitudes, el router debe estar configurado para enviar cierto tráfico a una máquina específica, generalmente asignando uno o más puertos TCP o UDP desde la IP pública a una interna. Si bien es posible configurar routers de forma manual, no todos los que quieran ejecutar una aplicación P2P u otro servicio de red podrán hacerlo. Las aplicaciones que utilicen libp2p se ejecutan no solamente en centros de datos o servidores con direcciones IP públicas estables sino en cualquier dispositivo. Para que suceda la conexión, libp2p se enfoca en técnicas como router automation, hole punching, autoNAT y circuit Delay. Configuración automática de router (router automation), algunos routers permiten que la tabla de enrutamiento se configure automáticamente como UPnP o nat-pmp (Port Mapping Protocol). Libp2p configura automáticamente una asignación de puertos que le permite escuchar el tráfico entrante.

Hole Punching es una técnica que se utiliza para conectar con aquellos nodos de la red que de cierta forma son inaccesibles por otros. La idea es que cuando un nodo solicita una conexión con otro, se tenga una lista de nodos con los cuales se ha conectado el nodo solicitante.
De esta manera los nodos se informan entre sí de sus direcciones observadas, esto se lo conoce como STUN (en inglés: Session Traversal Utilities for NAT).

En libp2p se utiliza el protocolo de identificación, que permite que un nodo le pida a otro cierta información de identificación. Al enviar su clave pública, el nodo que se identifica incluye el listado de direcciones que ha observado para el nodo que hace escucha la conexión entrante. Este mecanismo de descubrimiento externo cumple la misma función que STUN, pero sin la necesidad de un conjunto de "servidores STUN". El protocolo de identificación permite que algunos nodos en una red P2P se comuniquen a través de NAT que, de otro modo, serían inaccesibles.

Otra técnica que aplica libp2p para establecer conexión es la llamada auto-NAT, la misma aplica la idea de conocer la lista de nodos que establecen conexión con el nodo solicitante e intercambiar información. De esta forma otros nodos colaboran intentando establecer conexión a las direcciones observadas. Si tiene éxito, podemos confiar en que otros nodos puedan conectarse y comenzar a anunciar la dirección de escucha.

Por último, libp2p proporciona un protocolo de circuito de retardo (delay circuit) que permite a los nodos comunicarse indirectamente a través de un nodo intermedio. La funcionalidad pretende ser igual a TURN [40] (Traversal Using Relays around NAT) aplicada en otros sistemas de red. TURN es un protocolo de red que utiliza la técnica NAT, en determinadas ocasiones puede que un nodo sea inalcanzable incluso aun así aplicamos NAT por lo que se necesita utilizar los servicios de un nodo intermedio, el protocolo TURN fue diseñado para lograr realizar la comunicación.

Autenticidad

Cada nodo en IPFS tiene una identificación, por un lado tiene una clave privada y por otro lado una clave pública generada por un hash criptográfico, creada con el cifrado estático de S / Kademlia [3]. Los nodos guardan sus claves públicas y privadas cifradas bajo una frase con contraseña. Cuando un nodo quiere conectarse a la red con otro nodo, primero se realiza un intercambio de claves públicas y se verifica si la función Hash que encripta la clave pública del nodo al cual se quiere realizar la conexión, genera como resultado la identificación del nodo que solicito dicha conexión entonces se procede a realizar dicha conexión, caso contrario se cancela.

La estructura de datos que envuelve la descripción de la identidad del nodo en IPFS, el identificador del nodo es un multihash descrito en la sección anterior, las claves tanto pública como privada que son las que se utilizan para identificar al nodo cuando realiza un intercambio con otro nodo.

```
type NodeId Multihash
type Multihash []byte
type PublicKey []byte
type PrivateKey []byte


type Node struct {
    NodeId NodeID
    PubKey PublicKey
    PriKey PrivateKey
}

S/Kademlia based IPFS identity generation:
```

```
difficulty = <integer parameter>
n = Node{}
do {
    n.PubKey, n.PrivKey = PKI.genKeyPair()
    n.NodeId = hash(n.PubKey)
    p = count_preceding_zero_bits(hash(n.NodeId))
} while (p < difficulty)
```

3.4. Ruteo

IPFS utiliza lo que se conoce en una red peer-to-peer como DHT [32] (distributed hash table), para conocer los pares que almacenan los bloques de información según contenido.

Cada nodo en una red peer-to-peer contiene una base de datos distribuida que almacena clave-valor. Esto significa que cada nodo IPFS contiene una base de datos distribuida donde almacena la IP del nodo "vecino" y los bloques que contiene.

En IPFS la librería encargada de descubrir los pares cercanos y generar la tabla DHT es libp2p [61], la particularidad de libp2p es que permite las conexiones multiplexadas. Estas conexiones permiten establecer una sola conexión entre dos pares y comunicar cualquier servicio en forma remota a través de esta conexión, en vez de abrir una conexión cada vez que un par quiera enviar datos al otro. De esta forma establecer conexiones múltiplexas entre los pares se hace más eficiente, una vez que se establece una conexión entre los pares se puede realizar cualquier intercambio entre ellos sin tener que abrir conexiones nuevas cada vez que se quiera enviar algo, haciendo de esta forma un mecanismo eficiente y un buen uso del ancho de banda de la red.

Cada par que compone una red IPFS almacena en su tabla DHT el identificador del nodo y el bloque de datos que almacena. Si el tamaño del bloque de datos es menor que 1 KB lo almacena directamente en la tabla, caso contrario almacena el identificador del nodo y una referencia, que son los identificadores de los nodos que tienen el bloque de contenido.

A continuación se muestra la interfaz de ruteo en IPFS en la cual se definen las funciones principales para lograr una conexión a una red IPFS tales como, encontrar pares cercanos y manipular la tabla distribuida de nodos.

```
type IPFSRouting interface {
    FindPeer(node NodeId)
    // gets a particular peer's network address
    SetValue(key []bytes, value []bytes)
    // stores a small metadata value in DHT
```

```
    GetValue(key []bytes)
    // retrieves small metadata value from DHT
    ProvideValue(key Multihash)
    // announces this node can serve a large value
    FindValuePeers(key Multihash, min int)
    // gets a number of peers serving a large value
}
```

3.4.1. INTERCAMBIO DE BLOQUES - PROTOCOLO BITSWAP

En IPFS el intercambio de información se da a través del intercambio de bloques de contenido. Para realizar dicho intercambio se utiliza el mismo protocolo que utiliza BitTorrent: BitSwap [30].
Los nodos IPFS tienen una lista denominada "want_list", esta lista contiene los bloques que el nodo desea adquirir. Por otro lado los nodos ofrecen los bloques que almacenan en una lista bajo el nombre "have_list".

Estrategia del protocolo BitSwap en IPFS

Para diseñar una buena estrategia de intercambio de información entre nodos IPFS, los autores del protocolo se basan en los siguientes principios:

1. Generar estrategias para maximizar la performance del nodo durante todo el intercambio.
2. Evitar nodos malintencionados.
3. Ser eficaz y resistente a otras estrategias desconocidas.
4. Ser flexible a la hora de confiar en los nodos de la red.

Cada nodo IPFS tiene un "libro de registro" (ledger) [4] el cual va anotando las transferencias que realiza con otros nodos. De esta forma se intenta evitar la manipulación de información, todo intercambio de bloques queda registrado y cada nodo tiene su propio historial de información.
Cuando los pares establecen conexión entre sí, intercambian su historial el cual de ahora en más llamaremos ledger. Cada ledger está definido por la siguiente estructura:

```
type Ledger struct {
    Owner NodeId
    Partner NodeId
    bytes_sent int
    bytes_recv int
    timestamp Timestamp
}
```

En IPFS los nodos tienen la opción de elegir si desean guardar su historial de información o no. Son libres de eliminar aquellos ledgers donde tal vez ya los nodos con los cuales intercambiaron bloques no existen, o no han recibido ningún intercambio en un periodo considerable de tiempo. Por tal motivo se tiene un registro de tiempo, que indica la última conexión que se mantuvo con el nodo en cuestión.

La especificación del protocolo BitSwap para IPFS está definida por la interfaz que se presenta a continuación:

```
   type BitSwap struct {
        ledgers map[NodeId]Ledger
        // Ledgers known to this node, inc inactive
        active map[NodeId]Peer
        // currently open connections to other nodes
        need_list []Multihash
        // checksums of blocks this node needs
        have_list []Multihash
        // checksums of blocks this node has
    }

type Peer struct {
    nodeid NodeId
        ledger Ledger
        // Ledger between the node and this peer
        last_seen Timestamp
        // timestamp of last received message
        want_list []Multihash
        // checksums of all blocks wanted by peer
        // includes blocks wanted by peer's peers
    }


interface Peer {
    open (nodeid :NodeId, ledger :Ledger);
    send_want_list (want_list :WantList);
    send_block (block :Block) -> (complete :Bool);
    close (final :Bool);
}
```

En la interfaz Peer se describen las cuatro operaciones básicas de la dinámica de intercambio de bloques entre los nodos de IPFS. Se establece la conexión con la operación open() en la cual se indica el nodo al cual establecer conexión y se intercambian los ledgers hasta llegar a un acuerdo, una vez establecida dicha conexión se envían entre sí la lista de los bloques que desean obtener *want_list* de cada nodo. Si tienen los bloques que desea cada uno comienza el intercambio con la función send block, si pasa cierto periodo de

tiempo (por defecto viene configurado hasta 30 segundos) en los cuales no hay ningún intercambio se envía una señal *Ignored* para evitar que el nodo envíe algún bloque en ese tiempo.
Si los pares durante ese periodo de tiempo intercambiaron bloques, se modifica la lista *want_list* y esos bloques pasan a formar parte de la lista *have_list* de cada uno.

Luego se cierra la conexión en caso de que no desean intercambiar más bloques, o se espera un tiempo por default de 30 segundos y si no se recibe nada durante ese periodo se ignora y se cierra la conexión.

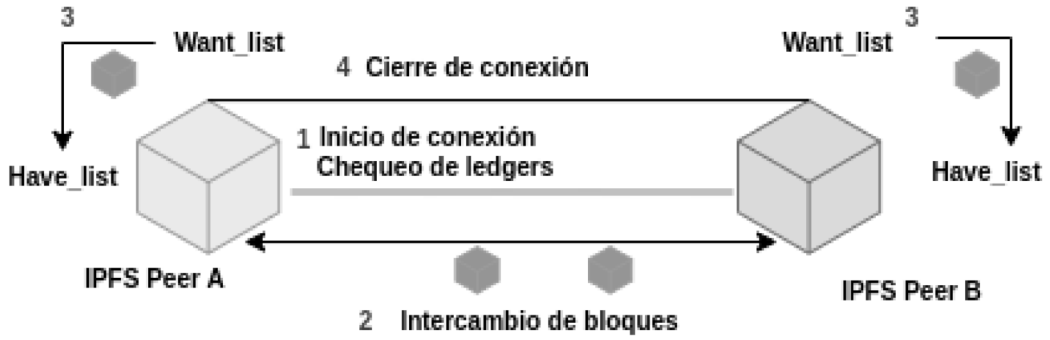

FIGURA 3.5: Conexión IPFS entre dos nodos.

En la Figura 3.5 se muestra el proceso en orden numérico según los sucesos que acontecen, primero el nodo A solicita la conexión al nodo B, luego una vez establecida la conexión se comienza con el intercambio de bloques, como se muestra en la Figura 9 con las listas *want_list* y *have_list*. Por último, se cierra la conexión entre ambos bloques.

Etapas de un archivo IPFS

Se pueden establecer cuatro etapas por las cuales un archivo es sometido durante el proceso de subida a una red IPFS.

- La primera etapa es establecer los chunks de los bloques pertenecientes al archivo. Estos son de 256 Kb de máximo tamaño.
- La segunda etapa corresponde a la asignación a cada bloque de contenido el CID correspondiente.Los bloques que pertenecen a un mismo archivo son vinculados en una estructura MERKLE y puestos en la lista *have_list* según el nodo IPFS que lo contenga.
- La tercera etapa consiste en establecer los protocolos de conexión de nodos, búsqueda de pares que contengan los bloques de contenido deseados así como también encontrar los pares de nodos IPFS más cercanos con los cuales se pueda intercambiar contenido.
- La cuarta etapa, llamada etapa Fetch, es donde interviene el protocolo Bitswap, encargado del intercambio de bloques entre los nodos una vez que estos establecen la conexión para intercambiar datos.

3.5. OBJETOS IPFS

3.5.1. Objeto Merkle en IPFS

En el caso de IPFS, al subir un archivo a la red primero se lo divide en bloques de 256 KB. Por cada bloque se genera un hash único que identifica al bloque según su contenido como anteriormente hemos hecho referencia se generaron 4 bloques, cada uno del tamaño que se describe en el campo size, junto con la función de hash que identifica a cada bloque.
Un archivo de gran tamaño puede contener múltiples bloques como se muestra en la Figura 3.6, cada bloque es identificado por una función de hash basada en el contenido del mismo. Por ejemplo el archivo de la Figura 3.6 muestra que para un mismo archivo se generaron 4 bloques según tamaño se describe en el campo size y el hash que identifica a cada bloque.

```
ipfs ls -v QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9
Hash                                           Size   Name
QmPHPs1P3JaWi53q5qqiNauPhiTqa3S1mbszcVPHKGNWRh 262158
QmPCuqUTNb21VDqtp5b8VsNzKEMtUsZCCVsEUBrjhERRSR 262158
QmS7zrNSHEt5GpcaKrwdbnv1nckBreUxWnLaV4qivjaNr3 262158
QmQQhY1syuqo9Sq6wLFAupHBEeqfB8jNnzYUSgZGARJrYa 76151
```

FIGURA 3.6: Archivo en una red IPFS

IPFS utiliza la estructura Merkle DAG para alojar los datos, hay dos requerimientos que debe de cumplirse, el primero es que tenga un identificador CID basado en el contenido del archivo y el segundo es que estén encapsulados en un formato el cual IPFS lo define como *IPFS Object* [4]. Hay dos tipos de objetos que modelan la estructura de datos de IPFS, por un lado se tiene una estructura para representar a IPFSLink y otra para IPFSObject. Los IPFSLink son los enlaces entre dos objetos en la estructura Merkle que se generan cuando un archivo es agregado a IPFS. Un enlace IPFS está representado por el hash del objeto destino.

```
type IPFSLink struct {
    Name string
    data []byte
}

type IPFSObject struct {
    links []IPFSLink
    data []byte
}
```

Los IPFSObject almacenan un segmento de enlace que contiene todos los enlaces a otros nodos y los datos que contiene el bloque.

En la Figura 3.7 se muestra de qué manera IPFS almacena los datos en una estructura de tipo Merkle, donde la raíz es el CID del archivo original enlazado a los cuatro bloques que representan al archivo con sus respectivos CID, cada bloque de datos es encriptado por la función criptográfica SHA-256.

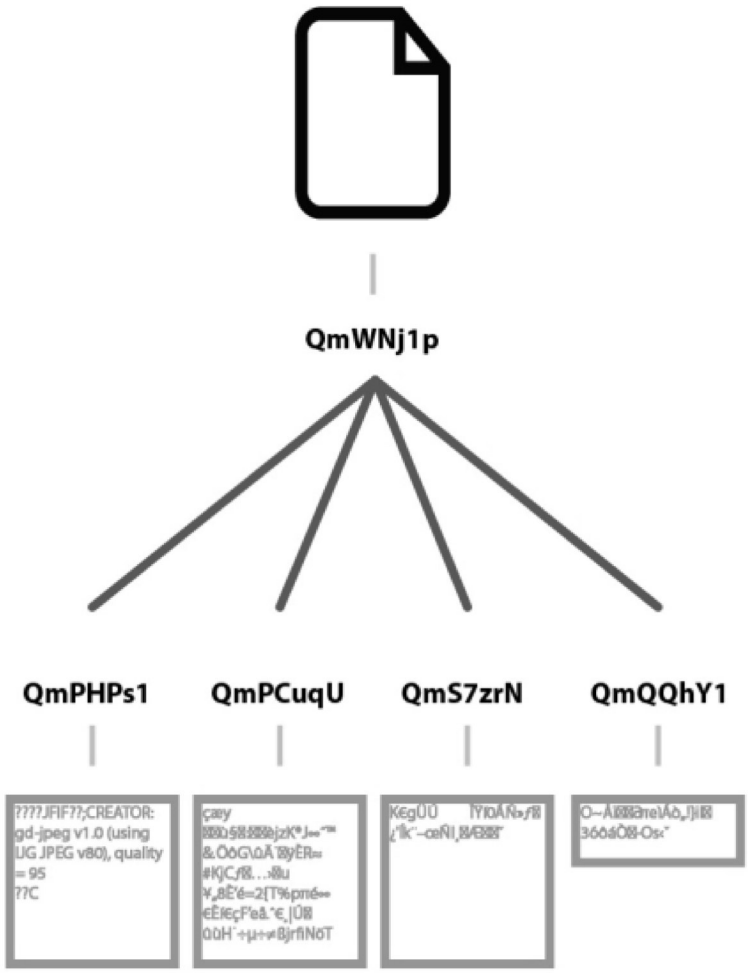

FIGURA 3.7: Estructura Merkle para un archivo IPFS [37]

El protocolo IPFS se compone de un conjunto de protocolos a los cuales se les delega diferentes responsabilidades. Estas responsabilidades se agrupan en funcionalidades tales como identidad, red, ruteo, intercambio de bloques, estructura de archivos, objetos IPFS y nombres de dominio IPNS. Si bien no son independientes entre sí, veremos por separado el diseño de cada una así como el rol que cumplen para que el protocolo IPFS pueda llevar a cabo su correcto desempeño.

En la actualidad cuando se quiere acceder a algún archivo en la web, le indicamos al navegador la dirección del sitio a donde queremos acceder, por ejemplo http:/digitalfiles.com/archivo.html.

En IPFS la forma de acceso se da través del link con el siguiente formato:

/ipfs/<hash-identificador>/ruta hacia el archivo

Ejemplo en IPFS en referencia al archivo alojado en digitalfiles.com:

```
/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/digitalfile/archivo.html
```

El prefijo /ipfs/ indica el montaje en el cual IPFS accede al sistema para alojar a los archivos, luego le sigue el hash que es el identificador del objeto el cual queremos acceder, seguido del nombre del archivo en cuestión.

Los nodos en IPFS deben guardar cierto espacio para alojar los documentos que comparten localmente en IPFS, puede ser un espacio en disco o en memoria RAM según así lo disponga el nodo. Dicha funcionalidad da la posibilidad a que durante el proceso de intercambio de bloques, se puedan descargar y almacenar de forma local o en memoria RAM de forma temporaria.

Se puede indicar que los objetos en IPFS estén almacenados de forma permanente, estos se denominan *pinning objects*. De esta manera se asegura la permanencia de datos, dado que en una red IPFS podemos indicar que los objetos se almacenan de forma permanente y no temporal por un determinado periodo.

3.5.2. Tipos de Archivos IPFS

Los archivos en IPFS se manipulan de forma similar a como lo hace Git [4]. Existen cuatro tipos de archivos que IPFS maneja en su estructura: listas, árboles, commit y blob.

IPFS Lista

Los archivos de tipo lista representan a una lista encadenada de archivos tipo blob a su vez esta estructura puede contener otras listas encadenadas dentro de la misma.Por ejemplo:

```
{
    "data": ["blob", "list", "blob"],
```

```
    "links": [
    { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x", "size": 189458 },
    { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5", "size": 19441 },
    { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z", "size": 5286 }
    ]
}
```

La estructura de archivo IPFS de tipo "lista" que se muestra en el ejemplo, representa una lista encadenada de archivo blob, lista y archivos blobs. En el campo "data" contiene la información de los elementos que integran la lista mientras que el campo "link" contiene el detalle de los elementos como el CID que los representa y el tamaño del elemento.

IPFS Arbol

A diferencia de las listas en IPFS, los árboles representan directorios que pueden contener todo tipo de datos, desde listas, archivos blob, árboles y commits. La estructura es la misma que las listas IPFS con la particularidad que en el campo "links" además de detallar el CID y el size del archivo contiene el campo "name", este indica el nombre del archivo.

IPFS Commit

En IPFS se puede representar el historial de cada archivo con una estructura de datos de tipo "commit", donde se detalla la fecha de creación así como el mensaje que es asignado en ese momento. Esto refleja un concepto similar al mecanismo que se sigue en Git cuando realizamos cambios en un proyecto y lo dejamos por sentado con un mensaje indicando que se realizó. Continuando con la estructura que se ha tomado de ejemplo en esta sección, en el campo "data" que se agrega el tipo de archivo, la fecha y el mensaje. De esta forma se tiene una captura del estado del archivo en este momento.

```
$ ipfs file-cat <archivo-hash> --json

{
    "data": {
    "type": "tree",
    "date": "2019-09-20 12:44:06Z",
    "message": "Este es un archivo que representa el historial de cambios."
},
"links": [
    { "hash": "<archivo1-hash>", "name": "parent", "size": 25309 },
    { "hash": "<archivo2-hash>", "name": "object", "size": 5198 },
    { "hash": "<archivo3-hash>", "name": "author", "size": 109 }
    ]
}
```

IPFS Blob

Dentro del contexto IPFS, los datos que se almacenan en los bloques son del tipo Blob, esto significa que no tienen una estructura asociada, el tipo Blob contiene una unidad de datos direccionable y representa a un archivo. Como se describe en la presente sección los archivos IPFS se pueden representar como listas, arboles, commits o Blobs siendo estos últimos los bloques de datos que almacenan los datos de los usuarios, sin estar enlazados. En el ejemplo siguiente se muestra la estructura del tipo de archivo Blob, el cual contiene el campo datos sin enlaces a ningún otro archivo IPFS.

```
{
    "data": "datos que son almacenados por los usuarios",
    // blobs no contienen enlaces.
}
```

3.5.3. Descubriendo nodos: Algoritmo Kademlia

En la Sección 3.1 hemos detallado como dos nodos en IPFS se conectan entre sí, cada nodo en IPFS contiene una tabla DHT distribuida donde almacena la identificación de los nodos y su dirección IP.

Cuando se inicializa el servicio IPFS los nodos tratan de descubrir a sus pares más cercanos, el algoritmo Kademlia[42] es con el que se basa IPFS para realizar dicha acción y así ir construyendo la tabla de pares que cada nodo almacena para intercambiar datos.

Cada nodo es identificado de forma única en la red peer-to-peer, para construir la tabla distribuida, IPFS se basa en la obtención de peers según el algoritmo Kademlia. Este fundamenta tres principios básicos:

1. El espacio de direcciones de los pares se encuentra identificado entre el rango $0.,2^{256-1}$.
2. Se basa en una métrica para ordenar los pares según el espacio de direcciones, de forma de visualizarlos desde la dirección más pequeña hasta la más grande. En IPFS se calcula en base a SHA256(PeerID) e interpreta el rango de direcciones de un entero entre el $0.,2^{256-1}$.
3. Una proyección que toma una clave de registro y calcula una posición en el espacio de direcciones donde el par o pares más idóneos para almacenar el registro deberían estar cerca. IPFS utiliza SHA256 (clave de registro).

Tener este espacio de direcciones y una métrica de ordenación entre pares nos permite buscar en la red como si fuese una lista ordenada. En particular, podemos convertir el sistema en algo así como una lista de omisión donde

un par conoce a otros pares que están a distancias de alrededor de 1,2,4,8 ... de él. Esto permite buscar en la lista en un tiempo logarítmico del tamaño de la red (N pares), O (log (N)).

3.6. Arquitectura IPFS

En esta sección presentamos la arquitectura e interfaz principal de IPFS, mostrando cómo los diferentes componentes que integran el protocolo interactúan entre sí, su instalación, qué tipo de servicios expone IPFS y de qué forma.

En la Figura 3.8 se muestra un diagrama completo de la arquitectura del protocolo IPFS.

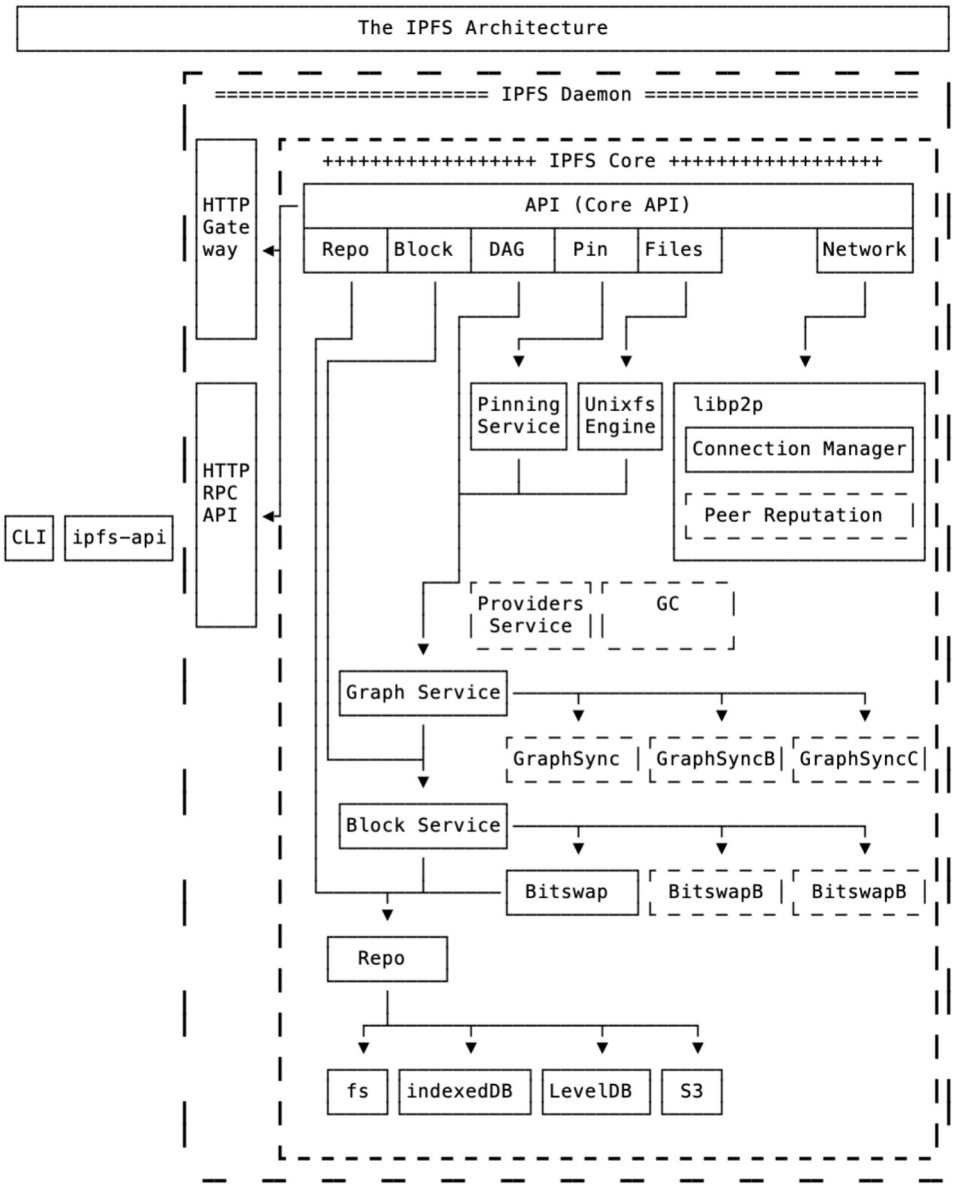

FIGURA 3.8: Arquitectura IPFS [13]

Instalación IPFS

IPFS está formado por un conjunto de paquetes, protocolos y configuraciones de archivos que permiten el intercambio de datos estableciendo conexiones entre nodos. El lenguaje de programación elegido para el desarrollo del protocolo IPFS es Go [17],es necesario tenerlo instalado para que IPFS funcione.

Para realizar la instalación de IPFS los usuarios pueden elegir qué paquetes instalarán así como también la versión de IPFS a utilizar. En el caso de ser un desarrollador de IPFS se puede descargar el proyecto [31] para así comenzar a utilizarlo o convertirse en un colaborador IPFS. Se recomienda instalar la versión de escritorio ya que es la más adecuada para el usuario que desea utilizarlo con el propósito de ser un nodo IPFS de intercambio y no un potencial desarrollador.

IPFS versión web

La versión web de IPFS es la más amigable para aquel usuario que desea utilizar el servicio. Cuenta con una interfaz gráfica según se muestra en la Figura 3.9, la cual le permite navegar por la web descentralizada, subir archivos, realizar descargas de otros nodos así como también cuantificar cuántos nodos IPFS están conectados. En la actualidad existen versiones para los sistemas operativos windows, ubuntu y macOS.

Para realizar la instalación del software se accede a través de la página oficial del sitio [29] donde se encuentran las versiones para los sistemas operativos según preferencia del usuario.
Una vez instalada la aplicación se accede a través del puerto 5001, en la cual desde la interfaz web se administra toda la información tal como se muestra en la Figura 3.9, accediendo a la red de datos descentralizada.

Existe otra alternativa de instalación de IPFS que es a través de comando de línea. Este tipo de instalaciones se recomienda realizarla para usuarios que cuenten con un cierto expertise técnico.

Comandos IPFS

Cuando se instala IPFS en un nodo, se ejecuta por defecto en el puerto 4001 el proceso el cual permite el descubrimiento de los nodos, el intercambio de bloques, el acceso a la información entre otros. Existen una serie de comandos que se ejecutan desde la consola IPFS, el objetivo es poder interactuar con el protocolo de forma alternativa a como se hace en la interfaz web de la API. En la Figura 3.10 se muestra la consola IPFS desde la cual se interactúa con el protocolo, la lista de comandos que provee IPFS se listan a continuación.

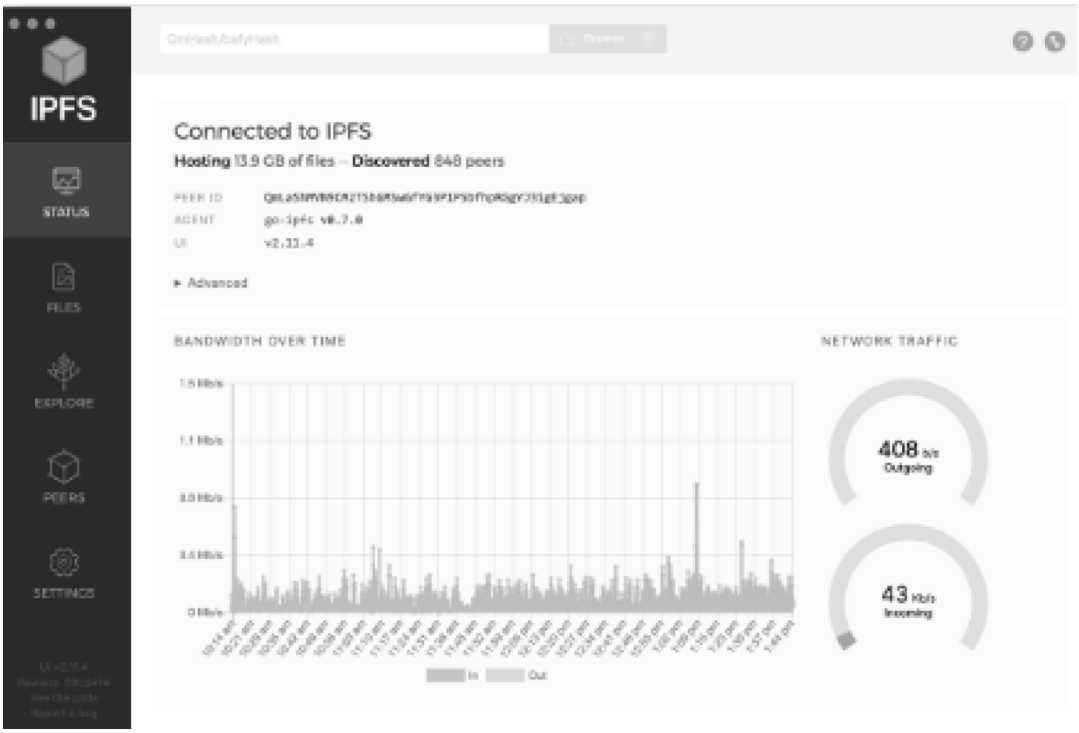

FIGURA 3.9: Interfaz grafica IPFS

1. IPFS init - El comando ipfs init se utiliza para inicializar el repositorio por primera vez.
2. IPFS daemon - Se utiliza para inicializar el proceso IPFS mediante el cual el nodo se une a la red IPFS.
3. IPFS config - Se ejecuta para que el usuario pueda configurar el protocolo, por ejemplo el número de puerto donde recibe las peticiones, la lista de nodos a los cuales conectarse al unirse a la red, entre otras funcionalidades.
4. IPFS add - Comando que se ejecuta cuando se quiere agregar un archivo a la red IPFS la cual está conectado el nodo.
5. IPFS ls - Lista todos los CID de los archivos que almacena el nodo.
6. IPFS cat - Edita el archivo.
7. IPFS swarm peers - Lista todos los nodos IPFS a los cuales está conectado.

3.6.1. Conclusión sobre el uso de IPFS

Con los conceptos base de arquitecturas distribuidas, los autores de IPFS tomaron como base lo implementado en [50] para la distribución de los bloques de contenido, de esta forma se evidencia que se pueden implementar

```
$ ipfs init
initializing ipfs node at /Users/jbenet/.go-ipfs
generating 2048-bit RSA keypair...done
peer identity: Qmcpo2iLBikrdf1d6QU6vXuNb6P7hwrbNPW9kLAH8eG67z
to get started, enter:

  ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/
```

FIGURA 3.10: Consola IPFS

sistemas basados en peer-to-peer que logran mejorar el uso de banda ancha frente a un sistema centralizado.

En esta tesis se trabaja con IPFS como protocolo para la distribución de archivos en una plataforma educativa que hace uso de Blockchain. En el siguiente capítulo se describen algunas aplicaciones de plataformas educactivas con Blockchain y con IPFS, y se introduce la plataforma educativa SELI sobre la cual trabajamos.

FIGURA 3.11: Inicio del servicio IPFS en un nodo [35]

Capítulo 4

Aplicaciones de Blockchain y de IPFS en Plataformas Educativas

En este capítulo se presentan algunos ejemplos de aplicaciones de Blockchain y de IPFS en escenarios educativos con el fin de comprender cuál es el aporte que agregan estas tecnologías a las plataformas educativas.

4.1. Blockchain en la educación

Las plataformas educativas son un claro ejemplo de arquitecturas centralizadas, donde toda la información la dispone la institución, pero si ocurren fallos por ejemplo, los usuarios pierden el acceso a los recursos encontrándose perjudicados en el normal desarrollo de sus estudios. Una alternativa es utilizar plataformas educativas basadas en una arquitectura descentralizada como lo son las redes peer-to-peer. En los últimos años han surgido desarrollos utilizando Blockchain que han tenido buena repercusión como alternativa a los procesos burocráticos existentes en educación. En esta sección se mencionan algunos beneficios de aplicar Blockchain en el área educativa, con el fin de comprender cuál es el aporte que agrega implementar Blockchain en plataformas educativas. A continuación describimos sucintamente algunos ejemplos:

Certificaciones

Las certificaciones son un área destacada para la utilización de tecnologías como Blockchain. Al ser una red inmutable, se puede confiar en generar certificaciones y diplomas emitidos por la institución en curso [49]. El proceso para otorgar un diploma en una institución educativa, suele ser un trámite burocrático y que por lo general lleva tiempo. Generalmente debe de ser firmado y validado por varias personas que avalan el diploma que se otorgará al alumno. En una red Blockchain, esto se soluciona generando un diploma digital y almacenando en la red la cual es inmutable. Blockcerts [63] es una plataforma basada en tecnología Blockchain que se dedica a validar diplomas, ahorrando trámites y procesos burocráticos por parte de las instituciones educativas.De esta manera las universidades pueden emitir títulos

a través de Blockcerts que podrían ser verificados al instante cuando los estudiantes los utilicen para solicitar trabajo o continuar sus estudios en otras instituciones.

Historial del alumno

se puede almacenar todo el registro académico de los alumnos de forma segura debido a las características de inmutabilidad de Blockchain. De esta forma cuando el estudiante culmine y quiera obtener su diploma, la información ya se encuentra registrada y validada en la red.

Insignias

Un estudiante a lo largo de su carrera u formación no necesariamente académica, puede aprender distintas habilidades que necesitan ser validadas en su currículum, la cual puede ser de relevancia para los empleadores. Estas habilidades no son fáciles de verificar. Pero una persona puede conseguir un tercer experto para verificar esa habilidad y otorgar un certificado o insignia. Si estos se almacenan en una red Blockchain, demuestran que una persona realmente tiene las habilidades en cuestión. Servicios como Open Badges [15] son un primer paso en esta dirección.

Reducción de costos

Las instituciones se ven beneficiadas a la hora de invertir en infraestructura, pues al basarse en arquitecturas peer-to-peer, cada usuario puede almacenar información y auspiciar de cliente-servidor para otros que integren la red. Esto implica un costo menor al que se tiene con una arquitectura centralizada puesto que ya no interviene un servidor central de almacenamiento.

4.2. IPFS en plataformas educativas

Caso ODEM

ODEM [51] es una empresa que ofrece una plataforma educativa basada en Ethereum [9], el objetivo de ODEM es ofrecer su plataforma para que los usuarios logren alcanzar sus metas educativas, donde el proceso de aprendizaje lo realiza cada uno a su ritmo. Si bien ODEM no es una aplicación descentralizada pues contiene una parte centralizada como el registro de usuarios y otras funcionalidades según [51], cuenta con una red Blockchain para diplomas una vez que el estudiante finalice sus estudios. Para crear un proceso eficiente, los diplomas en sí no se almacenan en la Blockchain. En cambio, un diploma es almacenado en una red IPFS, lo que se almacena en la Blockchain es el CID que identifica al diploma. IPFS agrega valor a la plataforma educativa de ODEM, logrando reducir el espacio de almacenamiento en la Blockchain.

FILECOIN: Red de almacenamiento descentralizado.

Filecoin [7] es una red descentralizada de almacenamiento o DSN (decentralized storage network) basado en blockchain que utiliza protocolo de consenso "Prueba de Espacio Tiempo" o (Proof-of-Spacetime o PoST). La iniciativa la lleva a cabo Protocol Labs, los mismos encargados del desarrollo de IPFS.

El objetivo principal de Filecoin es crear una red descentralizada que sea auditable, públicamente verificable y que incentive mediante un token del mismo nombre Filecoin (FIL) a almacenar datos de quienes así lo soliciten. En la Figura 4.1 se muestra la interacción entre los actores que lo integran, los dos actores principales son los clientes que solicitan el espacio y los mineros que ofrecen almacenar datos. Los mineros son incentivados mediante FIL como token de pago, para así crear los bloques con el espacio de almacenamiento solicitado.

Cuando un cliente desea solicitar espacio en la red descentralizada genera un nuevo pedido, este pedido es alojado en el Mercado de almacenamiento (Storage Market) para que luego un minero sea quien se encargue de almacenar los datos que el cliente solicita. Cuando llega una solicitud por parte de un cliente, esta es registrada (Orderbook), el minero que tenga la capacidad de alojamiento que se solicita, firma un contrato (deal) con el cliente y la envían a la Blockchain para registrar la transacción.

Los mineros garantizan su almacenamiento a la red depositando una garantía en FIL, esta se deposita por el tiempo destinado a proporcionar el servicio, y se devuelve si el minero presenta fallas a la hora de almacenar los datos. Cuanto más espacio ofrezcan almacenar mejor será la recompensa por parte de los clientes que obtendrán los mineros.

Filecoin incentiva a que exista otra dinámica alternativa donde los participantes de la red son libres de integrar, almacenar y acceder a información sin preocuparse si hay una entidad controlando el acceso. Lo que le da valor a Filecoin es que genera un nuevo mercado competitivo para alojar datos, con la diferencia de que no existe un ente centralizador que controle, el cliente elige según su conveniencia con quien alojar los datos obteniendo una remuneración en FIL como garantía que se cumplirá en el tiempo acordado.

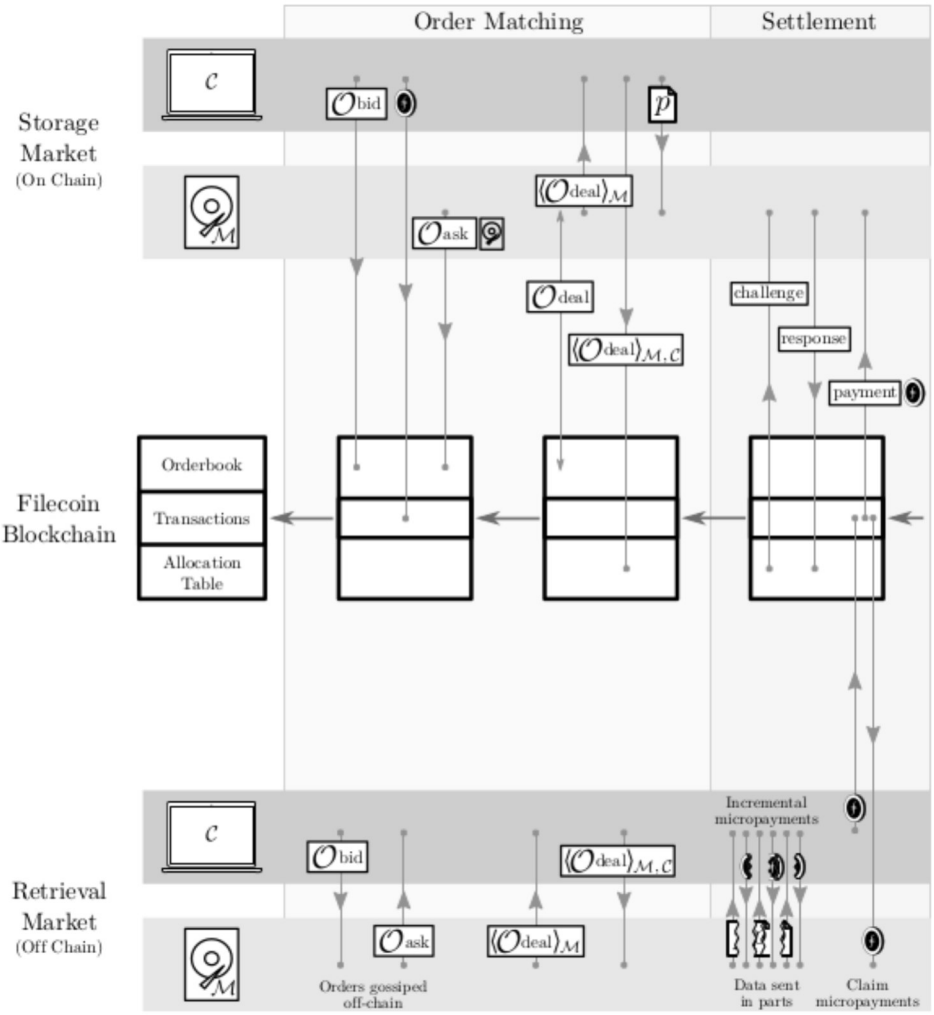

FIGURA 4.1: Interacción de los agentes que participan en el proyecto Filecoin [7]

Capítulo 5

Caso de estudio: Proyecto SELI

El proyecto SELI [66] tiene como objetivo servir de plataforma web, en donde los estudiantes puedan tener acceso a materiales y cursos dictados por docentes de instituciones educativas alrededor de distintos países.

El objetivo principal del proyecto nace en la difusión de recursos educativos que sean accesibles a través de esta plataforma y al mismo tiempo el estudiante encuentre el material de apoyo necesario para su proceso propio de aprendizaje. El sitio ofrece múltiples cursos de interés donde cada curso tiene una dinámica particular propuesta por el docente a cargo.

Con el fin de difundir recursos educativos nos hemos encontrado con el desafío de almacenar los recursos educativos que ofrece de la forma más óptima posible, teniendo como limitante contar con una cantidad finita de servidores a distribuir y un espacio acotado de almacenamiento.
He aquí donde el protocolo IPFS comienza a tomar protagonismo, en las siguientes secciones detallamos la experiencia que se tuvo en la incorporación de IPFS como protocolo de distribución de archivos dentro de la plataforma SELI.

5.1. Estructura del proyecto SELI

La plataforma SELI es un híbrido compuesto por arquitecturas centralizadas y arquitecturas distribuidas. Por un lado se tiene una plataforma centralizada por la cual los estudiantes y docentes acceden a ella y por otro lado para brindar certificaciones digitales, cuenta con una red privada Blockchain integrada por servidores en los países Ecuador, Finlandia y Uruguay.

Cuando un tutor crea un curso y desea subir a la plataforma SELI el material asociado al curso, este proceso está basado en una arquitectura completamente centralizada como se muestra en la Figura 5.1. Básicamente toda la información se almacena en un único servidor, esto conlleva a realizar múltiples peticiones que el servidor debe atender cada vez que los usuarios soliciten información o desean descargar material. Por lo anteriormente expuesto,

se presenta en el sitio ciertos tiempos de retraso que el usuario ha estado experimentado en los últimos tiempos.

Al ser una arquitectura completamente centralizada una de las principales desventajas es la responsabilidad de mantener la información activa en el sitio. Si por ejemplo en la actualidad, el servidor principal dejase de funcionar al estar pensado todo el sitio como un sistema centralizado de información, tiene como consecuencia dejar a todos los usuarios sin acceso al sitio y a los recursos por un lapso de tiempo.

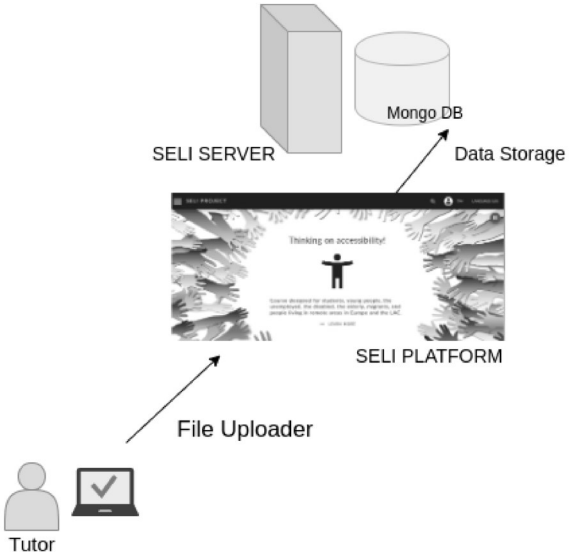

FIGURA 5.1: Arquitectura centralizada SELI.

El mayor de los desafíos consistió en la incorporación de IPFS (orientado para aplicaciones descentralizadas) a un proyecto centralizado como lo es SELI. Esto implica también la realización de evaluaciones de performance y de mejoras en el proceso de carga de archivos hacia la plataforma en diversos escenarios que se detallan a continuación.

5.1.1. Caso 1 - Escenario Inicial

SELI es un proyecto implementado en React y NodeJS, utiliza Meteor [20] como framework de desarrollo. Meteor es un framework open source implementado en JavaScript, cuyo objetivo es facilitar la gestión entre el backend y frontend del proyecto.
El framework separa en módulos las diferentes componentes del proyecto, nos centraremos especialmente en el módulo "Cursos", allí se encuentran todas las acciones referentes a la gestión de cursos dentro de la plataforma SELI.

IPFS EN SELI

IPFS aún está lejos de ser un protocolo estable y compatible con las múltiples tecnologías que existen hoy en día. IPFS es un proyecto open source por lo que se está abierto a encontrar vulnerabilidades donde diferentes desarrolladores se topan a la hora de utilizarlo en sus proyectos.
Cuando se inició la búsqueda para utilizar una librería IPFS dentro de SELI, nos topamos con la librería ipfs-http-client [38] la cual hacemos referencia en la Figura 17.

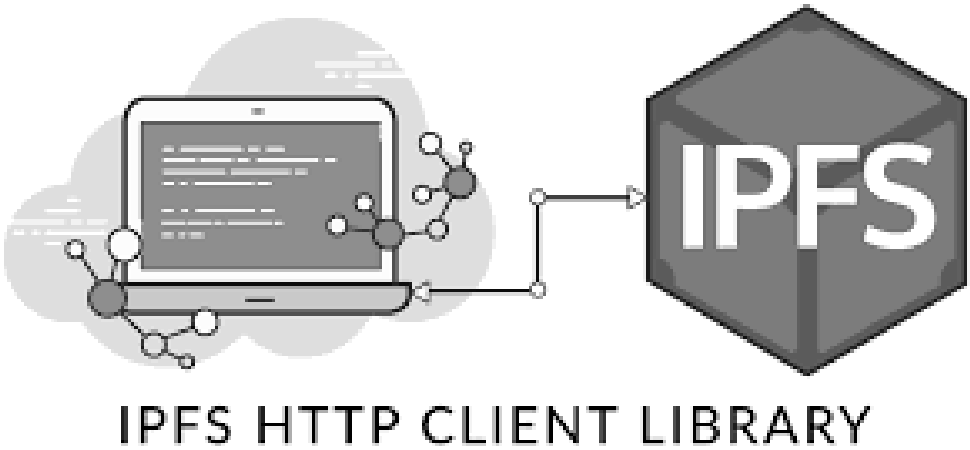

FIGURA 5.2: Libreria ipfs-Http-Client [38]

Esta librería desarrollada por Alan Shaw [18] uno de los co-desarrolladores del protocolo IPFS, está diseñada para poder interactuar con el protocolo

IPFS dentro de cualquier proyecto web.
Una vez incluida la librería dentro del proyecto comenzamos a buscar ejemplos de cómo interactuar con ella. En principio realizamos una prueba mínima, esta consiste en guardar en la red IPFS la imagen del logo del curso que un tutor diseña.
En la Figura 5.3 se muestra como es la arquitectura de la plataforma una vez que se interactúa con IPFS.
Pasamos de tener una plataforma basada en centralización como muestra la Figura 5.1 a una arquitectura híbrida, o sea compuesta por una parte centralizada y otra distribuida tal como se muestra en la Figura 5.3.

Cada nodo que compone una red IPFS almacena sus archivos en su localhost o eventualmente donde le indique al archivo de configuración del protocolo. Se puede almacenar en redes que son públicas y ofrecen espacio mínimo de almacenamiento, una infraestructura muy popular en el mundo de redes P2P es la empresa Infura [21].

Para un primer escenario y como primer acercamiento de esta tecnología en SELI, decidimos configurar el protocolo IPFS utilizando el servicio de Infura, cuyo espacio de almacenamiento gratuito alcanza los 2GB y luego si ese almacenamiento no es suficiente debemos considerar en contratar el servicio que ofrece la empresa.
Al utilizar una infraestructura pública como lo es Infura debemos tener presentes los pros y contras. En principio la principal razón fue que el servicio nos proporciona un almacenamiento gratuito de 2 GB en un servidor externo cuyo mantenimiento lo realiza la propia empresa Infura, delegando responsabilidades y costos de mantenimiento a la misma. La desventaja es que 2 GB de almacenamiento no es suficiente para la cantidad de recursos que contiene el sitio, por eso nos restringimos sólo a utilizar los logos de imágenes de los cursos. Esto nos habilita a ver la dinámica con el protocolo, por ejemplo, si la performance es mejor que con el escenario inicial.

Cuando un archivo utiliza IPFS como protocolo de distribución, lo primero que este realiza es asociar un hash a ese archivo, este hash es único y está basado solamente en el contenido del archivo. Si el contenido del archivo cambia o se modifica, este hash es otro diferente haciendo referencia a un archivo distinto. Esto es el fundamento principal de IPFS [33], siendo que se basa en el contenido del archivo y no en el nombre o tamaño.

En la Figura 5.3 se muestra la interacción del usuario al subir una imagen en el sitio, en el primer paso el tutor crea el curso y carga el logo que lleva ese curso. Luego dentro del sistema comienza la interacción con el protocolo IPFS mediante la librería ipfs-http-client la cual se configuró para que utilice Infura como servidor de almacenamiento, así se muestra en la interacción SELI - IPFS red pública.

El archivo se almacena dentro de la red pública IPFS y este le devuelve el

hash que asocia al archivo. Una vez que se recibe el hash asociado al archivo que se almacenó, se inserta un registro en la base de datos con los datos hash IPFS, fecha almacenamiento, usuario y curso asociado.

De esta forma se pasa de almacenar una imagen y su registro asociado, a sólo almacenar una tupla en la base de datos con el hash de tipos string devuelto por el protocolo IPFS. Ahora cuando se quiere acceder a la imagen o al archivo almacenado en esta red pública, se accede de la forma http.ipfs.infura.io/hashArchivo.

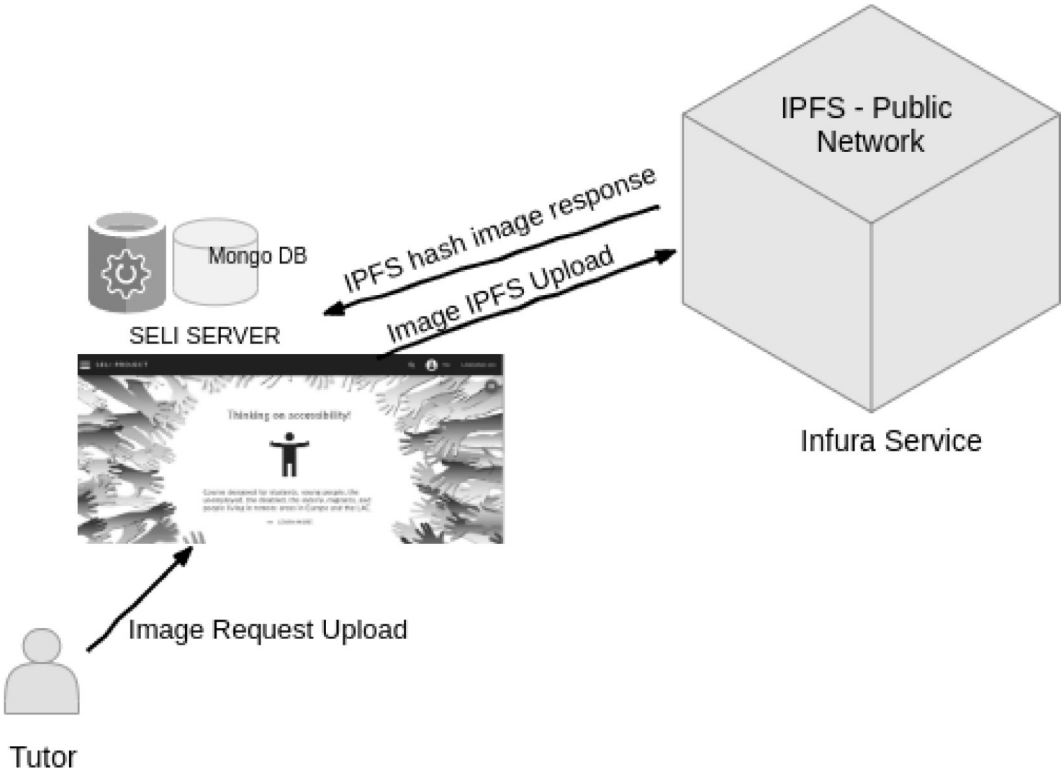

FIGURA 5.3: Arquitectura hibrida IPFS-SELI.

5.1.2. Dificultades encontradas

IPFS es una tecnología que está en vías de desarrollo y aún queda un camino por recorrer para que alcance la robustez necesaria para acoplarse bien a las tecnologías que están en uso hoy en día. Este factor es determinante a la hora de utilizar las librerías que ofrece IPFS para su inserción en un proyecto y SELI no fue la excepción.
Las librerías que IPFS ha desarrollado son varias, pero solo ipfs-http-client es la que se encuentra completa en cuanto a los servicios que expone, por ejemplo es la única librería que maneja archivos de browser, cuestión que en este caso es esencial si el objetivo que se tiene es que el usuario pueda cargar archivos a través de un sitio web.
El resto de las librerías que se probaron no eran de nuestra utilidad debido a

que ninguna tiene implementada aun la funcionalidad de poder trabajar con archivos blob [47].
Esto implica un gran impedimento pues solo dependemos de una librería para poder adaptarla a nuestro proyecto web. En la Figura 5.4 se detallan las librerías que fueron probadas para el proyecto SELI.

Librerias	Soporta archivos blob	Compatible con Meteor
ipfs-http-client	Si	x
Ipfs-mini [5]	No	Si
Pinata[6]	No	Si
SilentCicero/ipfs-mini[7]	No	x

FIGURA 5.4: Librerías IPFS utilizadas en IPFS-SELI.

Realizamos un proyecto web demo utilizando las mismas tecnologías que utiliza SELI que son React y Node JS con excepción de Meteor que no lo utilizamos dentro del proyecto demo [12].
El objetivo es interactuar con la librería y adaptarla en un principio a un proyecto web simple. Se simuló la carga de imágenes a través de un sitio web, se configuró la librería utilizando el servicio Infura y obtuvimos la dinámica esperada.
Se pudo obtener el hash del archivo y guardarlo en la red IPFS pública. El delay de respuesta por parte del servidor Infura se mantuvo estable alrededor de unos dos segundos por cada imagen que se cargó al servidor.

A partir de este caso exitoso y luego de familiarizarnos con la librería, pasamos a incorporar la misma en SELI. El hecho de haber realizado con éxito las pruebas preliminares nos dio una base de conocimiento para así facilitar la ardua tarea de incorporar dicha librería en un proyecto de mayor complejidad como lo es SELI.

Como se mencionó al principio SELI utiliza el framework Meteor, si bien en un principio no pareció un impedimento, realizamos los mismos pasos para incluir la librería pero en el momento de probar el mismo caso de uso que realizamos para el proyecto web base, en SELI no funcionó.
El primer error con el cual nos topamos “ipfsClient is not a function at ipfs.js”, nos indica que la función principal de la librería no es reconocida dentro del proyecto. Para poder continuar realizamos una búsqueda tratando de dar con la solución, teniendo en cuenta el contexto en el cual estamos realizando

las pruebas Meteor + IPFS.
Luego de varios intentos fallidos al respecto encontramos la solución [58], que nos permitió agregar la librería sin problemas. Para ello tuvimos que modificar el archivo package.json [59] de la librería ipfs-http-client, este archivo hace referencia para instalar las dependencias necesarias en la librería. En este caso el problema que detectamos fue un tema en cómo Meteor incluye las librerías para que el navegador pueda compilarlas. En la Figura 5.5 se muestra la secuencia de registro de una librería en un proyecto Meteor, npm es un sistema de gestor para Node JS donde se puede registrar las librerías a utilizar.

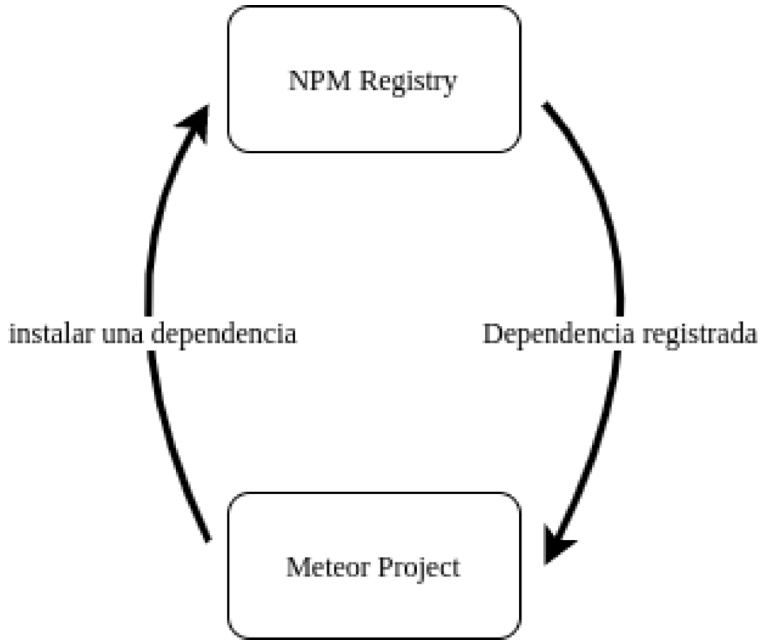

FIGURA 5.5: Secuencia de registro de una librería en un proyecto utilizando Meteor.

Una vez reconocidas las funcionalidades de la librerías ipfs-http-client, creamos una función dentro del proyecto SELI que simula el escenario propuesto de la Figura 5.1.
Agregamos una función del lado cliente que agregue el archivo a subir a la red pública IPFS, para ello insertamos el siguiente código:

```
{
const IpfsHttpClient = require('ipfs-http-client') // llamado a la librería
const ipfs = IpfsHttpClient({ host: 'ipfs.infura.io', port: '5001', protocol: 'http
// configuración del servicio
}

{
  onSubmit=(event)=>{
```

```
    event.preventDefault()

   ipfs.add(this.state.buffer,(error,result) => {
      console.log('IPFS result', result)
      const pic = result[0].hash
      this.setState({urlPic:pic})
      if (error){
        console.error(error)
        return
       }})}
}
```

IPFS interactúa con el servicio Infura y este devuelve el hash asociada a dicho archivo. Este código es el mismo utilizado en la demo que construimos para interiorizarnos con la librería. El mismo código que funciona en un proyecto web base con las mismas tecnologías React + Node JS, no funcionó para el proyecto SELI. Detectamos entonces que el problema seguía persistiendo con el framework Meteor, ahora al cargar un archivo en la red IPFS nos devuelve el error "cannot read property xxx" [47]. Más raro aún nos pareció el hecho de que probamos las librerías que muestra la Figura 5.4 y con todas ellas el mismo código no presenta error, pero el problema reside en que solo soporta texto y no archivos por lo que no nos fue útil para el proyecto.

Luego de buscar información que nos sea de utilidad y no encontrar una solución adecuada al problema en cuestión, decidimos comunicarnos con los desarrolladores de la librería exponiendo el contexto de pruebas que estábamos realizando. No obtuvimos respuesta.
La falta de información acerca de Meteor + IPFS sumado a la escasa documentación de las librerías IPFS en sí junto con Meteor, hacen ardua la tarea para encontrar una posible solución.
Por lo que nos hizo replantearnos un nuevo escenario, una nueva posible alternativa para aplicar IPFS al proyecto SELI con el propósito de optimizar el almacenamiento de archivos.

5.1.3. Caso 2 - Escenario Red privada IPFS

Actualmente en SELI existen tres servidores de instituciones educativas interconectados entre sí que integran la red blockchain. Los países que integran la red Ethereum [9] Blockchain en SELI son Ecuador, Finlandia y Uruguay. Esta red se dedica a crear diplomas digitales a través de *smart contracts*, lo cual le permite a los alumnos obtener una certificación cada vez que culminan un curso en la plataforma.

Para sacar mayor provecho al uso de esta red de servidores introducimos el concepto de crear una red IPFS-Cluster con el objetivo de descentralizar el almacenamiento de archivos en SELI.
IPFS-Cluster tiene como objetivo principal ser una red descentralizada de distribución de archivos de forma que cada nodo que compone el cluster contenga una réplica de los demás nodos.
Una red cluster IPFS es una aplicación distribuida que funciona como un orquestador en una red IPFS, cuya función es alojar de forma inteligente los ítem de los pares que conforman la red. Cada nodo que forma una red cluster debe correr el demonio IPFS y esta instalación se realiza aparte del cluster. Por tal motivo decimos que el cluster es un orquestador dentro de una red IPFS.

El cluster IPFS está compuesto por tres servicios ipfs-cluster-service, ipfs-cluster-ctl e ipfs-cluster-follow. Estos servicios se instalan por separado, ipfs-cluster-service es el principal siendo este el que realiza las configuraciones dentro del cluster.
Luego tenemos el servicio ipfs-cluster-ctl que es el que se comunica con otro peer del cluster y además se encarga de replicar los archivos que son agregados a esta red.
Por último se tiene el servicio ipfs-cluster-follow el cual oficia de sustituto de ipfs-cluster-service, siendo una combinación de ambos servicios ipfs-cluster-service e ipfs-cluster-ctl.

En la Figura 5.6, se muestra la arquitectura que engloba al cluster IPFS, cada nodo tiene por separado los dos servicios IPFS por un lado y el servicio IPFS-cluster por otro.

Subir un archivo a un IPFS-Cluster

En una red IPFS-Cluster todos los nodos son iguales, no existe ningún nodo que tenga autoridad o privilegio alguno. Al ser una aplicación descentralizada carece de autoridad centralizadora por lo que todos los nodos dentro del cluster contiene una réplica del otro. De esta forma, si existiese alguna falla en algún nodo, la información siempre estará respaldada.
Cuando se inicia un cluster por primera vez se debe configurar el modo en el cual se realiza el consenso entre los pares. Existen dos modos de consenso en

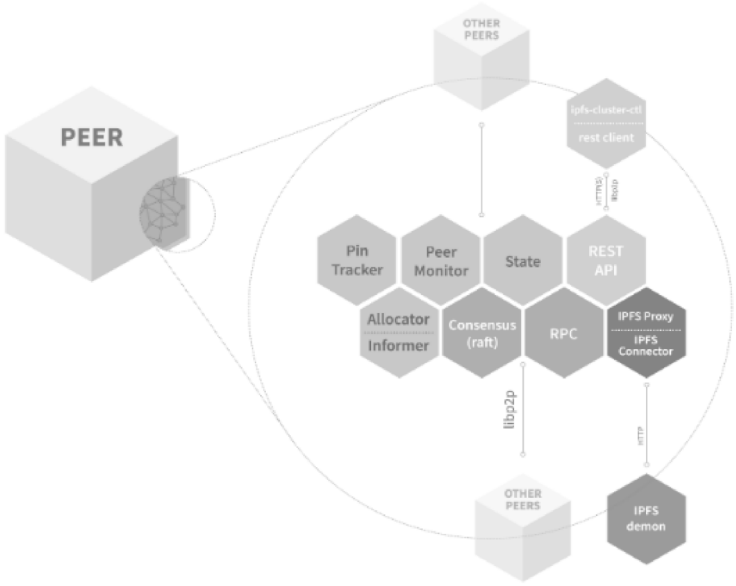

FIGURA 5.6: Arquitectura IPFS-cluster[36]

un IPFS-Cluster, raft y crdt.

Modo CRDT

Basado en "Conflict-Free Replicated Datatypes", este modo es el que viene por defecto en la configuración. Está diseñado para que los pares que integran el cluster puedan unirse o abandonar el cluster de una forma ágil. Generalmente se configura este modo cuando estamos ante una red más inestable.

Modo RAFT

Basado en "log-based consensus algorithm", se configura este modo cuando tenemos una red de nodos estable, en la cual es raro que un par abandone la red. Para el caso de SELI es la modalidad más adecuada para configurar dado que es una red estable donde no hay una variación significativa en la cantidad de pares a unirse o abandonar la red.

Dinámica de carga de archivos en un cluster IPFS

En un servidor perteneciente a un cluster se tiene por un lado el servicio IPFS y por el otro lado el IPFS-Cluster service.
Cuando se agrega un archivo a un cluster, se utiliza el servicio ipfs-cluster-ctl, este interactúa con el nodo IPFS quien almacena en forma local el archivo en cuestión. A nivel del cluster se replica el hash IPFS que devuelve el servicio y este hash se replica en todos los nodos pertenecientes al cluster.
En la Figura 5.7 se visualiza el escenario de una red IPFS-Cluster dentro del proyecto SELI. Se muestran los servidores interconectados entre sí, en donde en cada servidor está instalado el nodo IPFS y el cluster. Cuando un nodo se une a la red cluster debe correr el servicio y una clave que se determina al unirse al cluster, a partir de allí todo archivo que se suba a ese servidor se replicará dentro del cluster.
Es una solución válida donde se almacena en forma inteligente los archivos,

creando un balanceo de carga óptimo.

A modo de ejemplo, si subimos un archivo al nodo SELI, este se sube al cluster, a su vez el cluster lo replica en el nodo de Uruguay, Ecuador, Finlandia. Lo que se replica es el hash IPFS el archivo queda localmente alojado en la red local IPFS del nodo SELI.
Es importante destacar que no hay duplicación de archivos, que cada vez que un archivo se sube a una red IPFS se genera un hash según contenido del archivo, y solo el hash es el que se replica en el resto de los nodos.
Es una buena solución para la distribución de archivos, favorece la distribución inteligente entre servidores y el balance de carga.

Esta mejora en la eficiencia, se logra mejorando el balance de la carga a la que están sometidos tanto los servidores que alojan los contenidos como los enlaces que interconectan las distintas secciones de la red, eliminando posibles cuellos de botella y sirviendo los datos en función de la cercanía geográfica del usuario final.

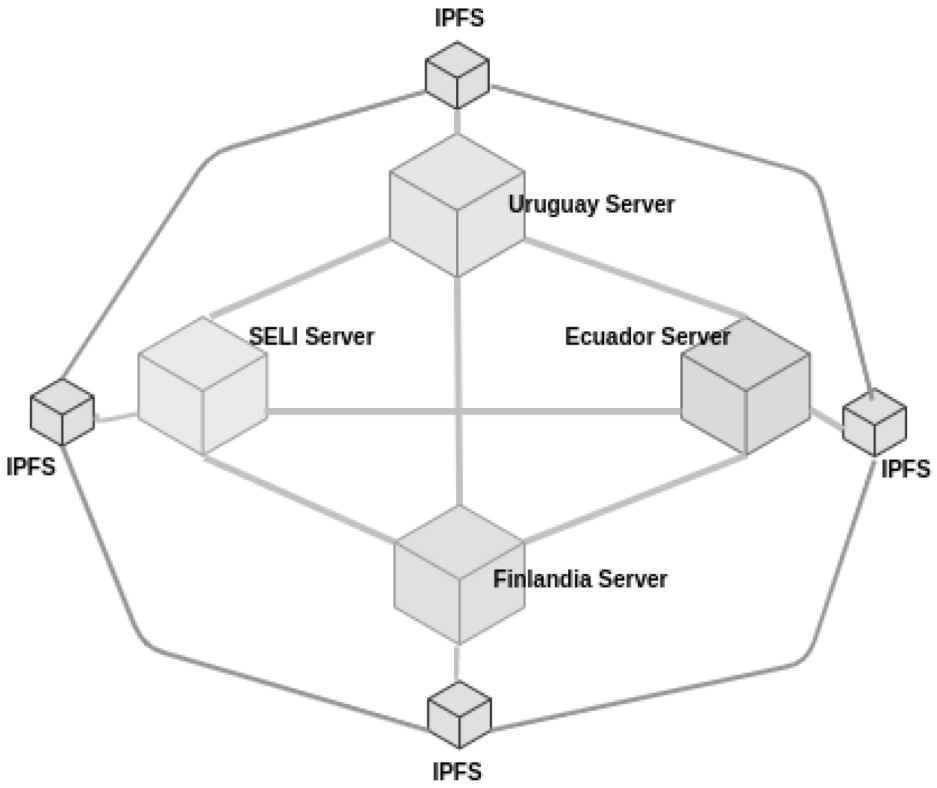

FIGURA 5.7: Red Cluster en SELI.

En el caso SELI, el sitio está alojado en un solo servidor por lo que los archivos físicamente quedan alojados en los servidores locales, pero favorece a la distribución y uso de los servidores restantes que integran la red. De forma que se realiza un backup de datos cada vez que un archivo se sube al cluster.

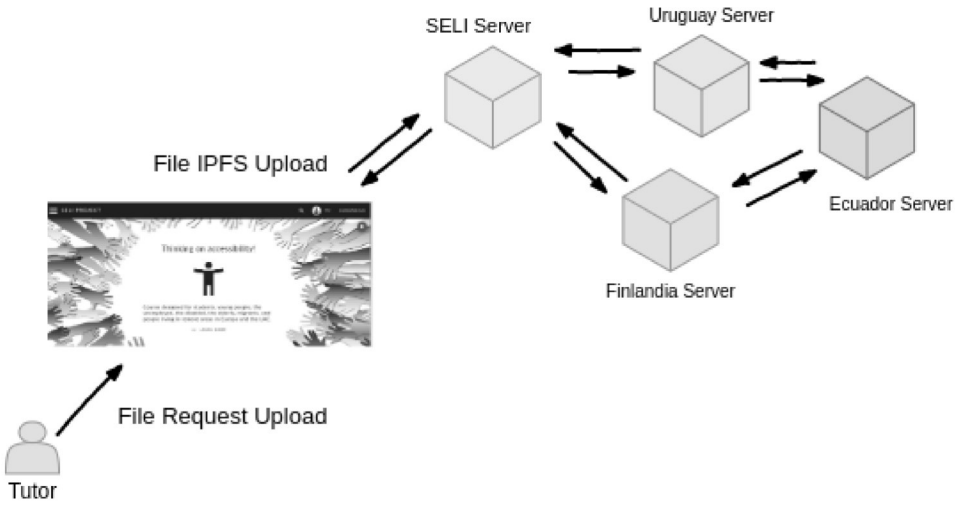

FIGURA 5.8: Caso SELI (autoría propia)

5.1.4. Caso 3 - Escenario diseño de arquitectura: D-SELI

Los dos escenarios anteriores proponen una mejora en la realidad actual del proyecto. En el escenario inicial realizamos un análisis de la arquitectura del proyecto en el cual encontramos una fuerte incompatibilidad a la hora de incluirle IPFS.
A raíz del caso inicial se propone una solución alternativa para incluir una red privada basada en IPFS para el almacenamiento de recursos educativos en SELI. Ninguna solución propuesta anteriormente logra aprovechar al máximo el potencial que brinda el protocolo IPFS, para eso se necesita hacer un cambio radical desde los cimientos de la aplicación.
En esta sección se propone como trabajo a futuro desarrollar una nueva aplicación basada en una arquitectura descentralizada fusionando dos tecnologías: Blockchain e IPFS.

D-SELI

D-SELI es la propuesta de versión descentralizada del proyecto SELI. Se propone cambiar el paradigma con el cual se trabaja en la versión SELI pero manteniendo la misma filosofía que impulsó al proyecto desde sus comienzos. La propuesta D-SELI consiste en utilizar tecnología Blockchain, e IPFS como el protocolo que maneja el almacenamiento de los recursos en la aplicación. La idea es que cada usuario de la aplicación D-SELI sea un nodo IPFS dentro de una red blockchain, donde los recursos educativos se almacenarán en la red IPFS compuesta por todos los usuarios D-SELI. Cuando un usuario se une a D-SELI, participa o dicta cursos de su propia formación o interés por lo que compartir recursos educativos es una funcionalidad esencial dentro de la plataforma. Por lo que es acertado una librería personal del usuario donde almacenará y dispondrá de sus archivos los cuales a su vez son compartidos y accesibles para todos aquellos que así deseen.

En la Figura 5.9 se muestra como es la arquitectura, no existe un servidor central que almacena información alguna, cada usuario de D-SELI auspicia de cliente-servidor.

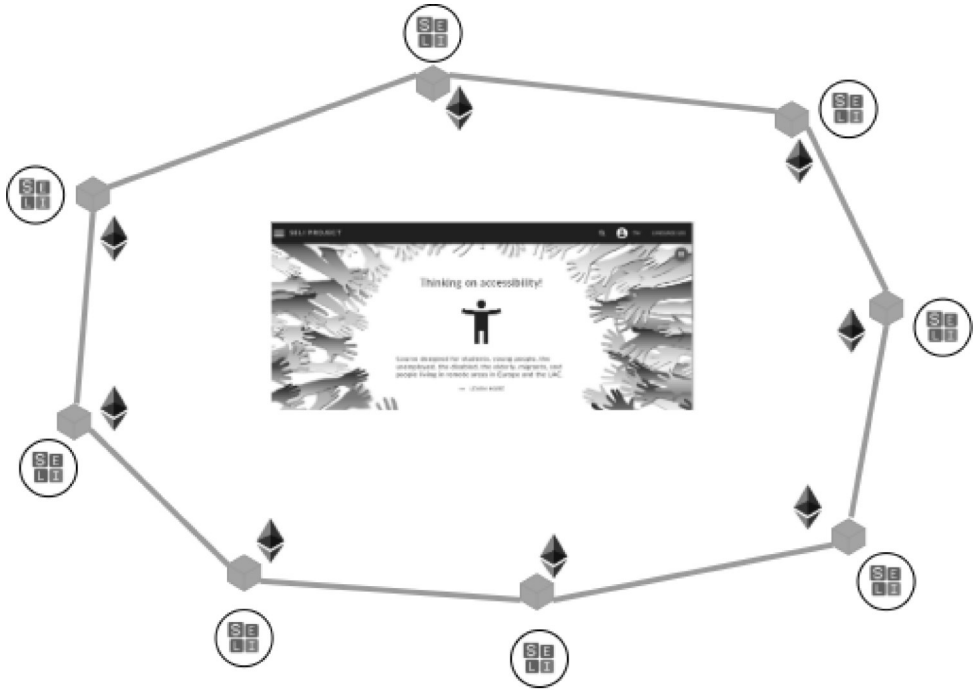

FIGURA 5.9: Arquitectura D-SELI (autoria propia)

D-SELI Coin

En la nueva versión descentralizada D-SELI, los usuarios tienen la opción de optar por la posibilidad de ser remunerados por almacenar datos de otros usuarios que así lo requieran. De esta manera se busca incentivar y explotar al máximo la filosofía de intercambio de archivos que busca incentivar a los usuarios para que pongan a disposición de otros sus recursos y a la vez estos sean remunerados. Esta idea se basa en el proyecto FILECOIN el cual se detalló en la sección 4.2 Los usuarios que almacenen datos de otros, serán remunerados con una criptomoneda propia de la plataforma D-SELI, la llamamos D-SELI Coin.
La idea de generar una criptomoneda propia de la plataforma es incentivar el uso de IPFS y de esta forma los usuarios serán beneficiados ofreciendo recursos para compartir almacenamiento a cambio de un resarcimiento que le permita obtener beneficios. Estos beneficios varían desde obtener reconocimiento dentro de la plataforma, como por ejemplo poder valorar positivamente aquellos recursos que le hayan sido de utilidad para su proceso de aprendizaje así como también aquellos no tan relevantes. De esta forma permite guiar a otros usuarios que compartan los mismos temas de interés.

5.2. Estudio de comportamiento: Meteor e IPFS

Si bien la idea de incorporar IPFS dentro del proyecto SELI en principio fue para estudiar el funcionamiento del protocolo dentro de un sistema con alta demanda por parte de los usuarios, en lo que refiere a carga y descarga de recursos educativos, este no fue su único fin.
Ante el inminente crecimiento de usuarios dentro de la plataforma y debido al incremento de volumen de recursos educativos, el sistema comenzó a experimentar fallas en el almacenamiento de datos sobretodo en la carga de archivos de los cuales los usuarios comenzaron a notar grandes demoras y fallas en la conexión al intentar acceder así como también al cargar los recursos en la plataforma.
Se desea comprender el rendimiento del framework Meteor en el cual se implementó SELI, se decide realizar pruebas de conexión y carga cuyo objetivo es obtener métricas que permitan estudiar el comportamiento del framework. El mismo escenario se plantea para el caso de IPFS en SELI, para el cual se realizaron la mismas pruebas cuyo objetivo es ver el desempeño de ambos tanto del framework Meteor como del protocolo IPFS.

Para realizar las pruebas se utilizó la herramienta Apache Jmeter[55], la cual proporciona una interfaz gráfica amigable para el usuario, esta permite establecer múltiples escenarios de prueba, a su vez brinda una opción para obtener métricas relevantes, útiles a la hora de analizar rendimiento y comportamiento.
Los casos que se presentan a continuación se realizaron en una notebook HP con procesador Intel Core i7 vPro velocidad de procesamiento de 2.6 Ghz, memoria RAM 16 GB y disco sólido de tamaño 1TB.

5.2.1. Meteor

Meteor es un framework basado en lenguaje Javascript, el mismo facilita la interoperabilidad cliente-servidor. El framework se basa en una arquitectura centralizada, la cual es diseñada para desarrollar aplicaciones web y móviles. Con el fin de analizar el comportamiento de Meteor frente a múltiples conexiones en simultáneo, se plantean diferentes casos de uso en donde la carga útil de las peticiones de los usuarios se incrementa para cada contexto de prueba.

Escenarios de prueba

Para realizar la carga de archivos en la plataforma SELI, se utiliza la librería Meteor:ostrio[12], esta librería presenta una limitante a la hora del almacenamiento de archivos, pues no permite almacenar archivos de gran tamaño, tiene una limitante que solo permite almacenar archivos de no más de 10MB de tamaño.

Antes de comenzar con el análisis de los resultados obtenidos, se definen los conceptos de las métricas más relevantes obtenidas con Apache JMeter.

1. Media - Es aquel número que divide el cincuenta porciento de las peticiones completas en un tiempo menor a esa medida y el otro cincuenta porciento de aquellas peticiones que se completan en un tiempo mayor a la media.
2. Desviación (capacidad de procesamiento) - es un ratio o proporción que representa la cantidad de trabajo completo por un periodo definido de tiempo.
3. 90 % line - Representa el noventa porciento de las peticiones que se consumaron por debajo del tiempo indicado, medido en milisegundos.
4. 95 % line - Representa el noventa y cinco porciento de las peticiones que se consumaron por debajo del tiempo indicado, medido en milisegundos.
5. Máximo - Tiempo máximo que toma completar una solicitud medido en milisegundos.
6. Mínimo - Tiempo mínimo que toma completar una solicitud medido en milisegundos.
7. Latencia - Tiempo de respuesta por cada nueva conexión, medido en milisegundos.

Primer Caso

El primer caso de prueba cuenta con una muestra de 10000 usuarios conectados en simultáneo, que realizan la carga de un archivo de tamaño 20 KB. Esta prueba se realiza a nivel local sin tener que conectarse a un servidor externo.

El resultado obtenido se muestra en las Figuras 5.10 y 5.11, la Figura 5.10 representa el total de peticiones realizadas con éxito mientras que la Figura 5.11 muestra el rango de tiempo de respuesta en el cual son atendidas las peticiones. Para una prueba de carga de archivo pequeño de 20 Kb se obtiene un 100 por ciento de peticiones exitosas con 10000 usuarios concurrentes, la misma se realizó con un incremento de hasta 50000 usuarios concurrentes y mismo tamaño de archivo obteniendo el mismo resultado de éxito.

En la Figura 5.11 se muestran los tiempos de respuesta de la aplicación en milisegundos (ms), en color verde se atendieron un promedio de 4500 peticiones en un rango de tiempo menor de 500 ms, entre 500 ms y 1500 ms son atendidas un promedio de 4900 y el resto se ubica en un tiempo mayor a 1500 ms.

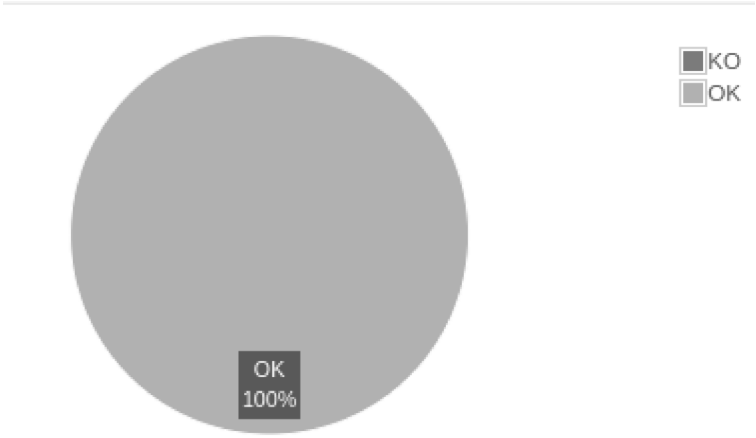

FIGURA 5.10: Respuestas exitosas en servidor local (autoría propia)

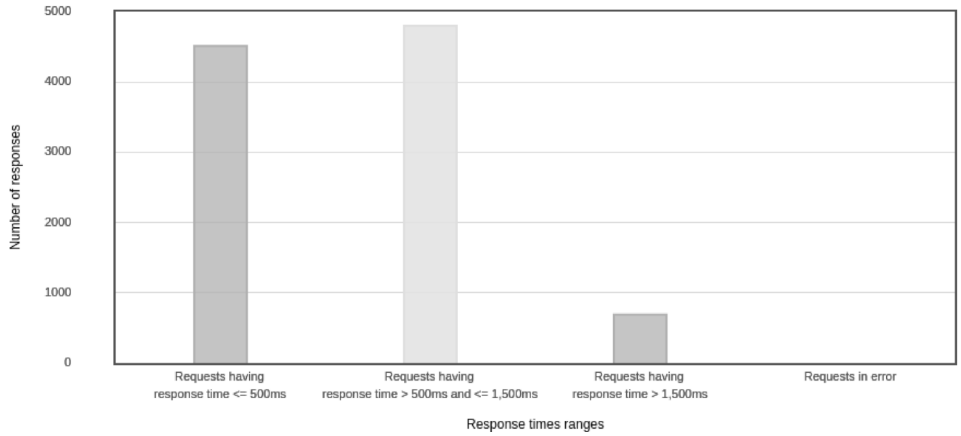

FIGURA 5.11: Tiempo de respuesta (autoría propia)

En la Figura 5.12 se muestran las métricas que se obtuvieron fruto de la prueba propuesta. Como no se registraron errores en el contexto propuesto el porcentaje indica un 0 %, el promedio en milisegundos(ms) de respuesta desde que se envía el primer byte de datos hasta que se complete toda la operación por parte del servidor es de 695.81 ms. El mínimo tiempo empleado en completar una petición al servidor fue de 2 ms y el máximo tiempo en el cual se tardó fueron 1667 ms.
El rendimiento para esta prueba resultó ser de 293.12 transacciones por segundo en el cual el servidor logra procesar la información recibida y responder a esta. La media indica que el cincuenta porciento de las peticiones se completan en un tiempo menor de 724 milisegundos, mientras que el otro cincuenta porciento lo logra hacer en un tiempo mayor al indicado por la media.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**0**	**0.00%**	**695.81**	**2**	**1667**	**724.00**	**1458.00**	**1538.00**	**1620.99**	**293.12**	**2795.21**	**5052.29**
HTTP Request	10000	0	0.00%	695.81	2	1667	724.00	1458.00	1538.00	1620.99	293.12	2795.21	5052.29

FIGURA 5.12: Estadísticas obtenidas al finalizar la prueba (autoría propia)

Segundo caso

El segundo caso de prueba se incrementa el tamaño del archivo, en este caso el tamaño es de 2MB y las pruebas se realizaron en base a la misma cantidad de usuarios que en el primer caso o sea 10000 usuarios concurrentes. Las respuestas obtenidas presentan una notoria diferencia con respecto al primer caso en el cual el archivo es de un tamaño mucho menor, en la Figura 5.13 se muestra el resultado obtenido con la herramienta Apache Jmeter luego de ejecutar la segunda prueba. Se obtuvo un total de 52.87 % de éxito frente a un 47.13 % de respuestas fallidas.

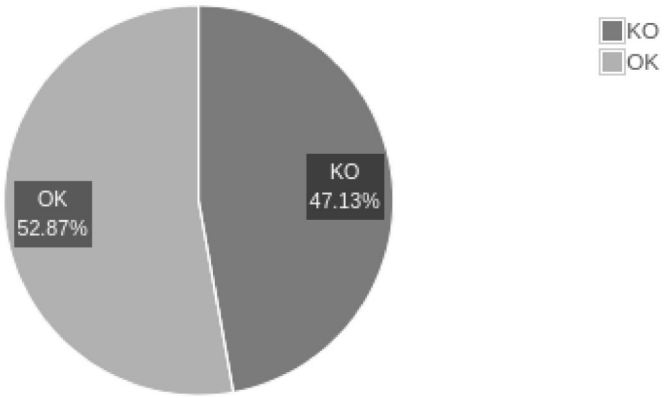

FIGURA 5.13: Peticiones exitosas y no exitosas durante las pruebas (autoría propia)

En la Figura 5.14 se muestran las métricas que se obtuvieron fruto de la prueba propuesta. Se registraron errores cuyo porcentaje es de 47.13 %, el promedio en milisegundos de respuesta desde que se envía el primer byte de datos hasta que se complete toda la operación por parte del servidor es de 2187 ms. El mínimo tiempo empleado en completar una petición al servidor fue de 5 ms y el máximo tiempo en el cual se tardó fueron 1127276 ms el cual es equivalente a un total de 18 minutos.

El rendimiento para esta prueba resultó ser de 7.91 transacciones por segundos en el cual el servidor logra procesar la información recibida y responder a esta.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**4713**	**47.13%**	**384838.00**	**5**	**1127276**	**2187.00**	**1029556.90**	**1030994.00**	**1034286.97**	**7.91**	**50.34**	**8624.85**
HTTP Request	10000	4713	47.13%	384838.00	5	1127276	2187.00	1029556.90	1030994.00	1034286.97	7.91	50.34	8624.85

FIGURA 5.14: Métricas tomadas durante la prueba.

Se evidencia notoriamente que al aumentar el tamaño del archivo a procesar los tiempos de respuesta son más lentos con respecto a un archivo de menor tamaño. Las razones pueden variar por múltiples razones, se analizan algunos errores reportados por la herramienta Jmeter.

Los tipos de errores detectados por Jmeter se describen en la Figura 5.15, en la mayoría de los casos el encabezado del error es "Non HTTP", cuando el servidor lo envía como respuesta generalmente se debe a que ya no recibe peticiones en el puerto donde está instalada la aplicación o no responde la conexión a internet. En este caso cuando hay alta demanda de usuarios conectados en simultáneo el servidor se sobrecarga y no acepta más conexiones haciendo que el puerto en el cual se instaló la aplicación no recibe más peticiones.

Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset by peer (Write failed)	3616	76.72%	36.16%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection timed out (Write failed)	615	13.05%	6.15%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset	400	8.49%	4.00%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Broken pipe (Write failed)	52	1.10%	0.52%
503/Service Unavailable	24	0.51%	0.24%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection timed out (Read failed)	6	0.13%	0.06%

FIGURA 5.15: Fallas en la ejecución de las peticiones durante las pruebas (autoría propia)

La Figura 5.16 muestra el tiempo de duración en milisegundos en el cual el servidor completa las peticiones agrupándolas en rangos de tiempo. Se detectaron cuatro rangos, el primer rango se encuentra entre 0 y 500 ms, son aquellas peticiones en las cuales se completa exitosamente la operación. Luego se agrupan en aquellas que abarcan entre 500 y 1500 ms y por último las que ocupan más de 1500 ms. Se observa una gran cantidad de peticiones no realizadas que son aquellas en las cuales el servidor responde con un error como los que se identifican en las Figura 5.15.

Las gráficas reflejan un cambio en la performance del sistema al procesar un archivo de 20 Kb y un archivo de 2 MB.

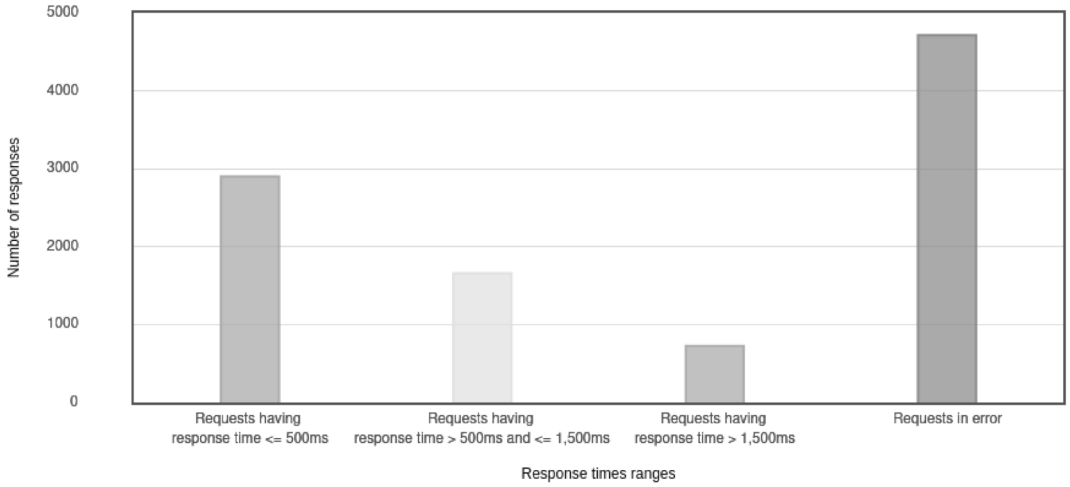

FIGURA 5.16: Tiempos de respuestas durante las pruebas.

Tercer caso

Como tercer caso de prueba se aumentó aún más el tamaño del archivo para ver el comportamiento de la librería Meteor ostrio. El tamaño del archivo es de 5 MB, se mantiene la misma cantidad de usuarios concurrentes (10000) que los casos anteriores, el porcentaje de peticiones exitosas y no exitosas se muestra en la Figura 5.17 siendo notorio el alto porcentaje de peticiones fallidas frente a las exitosas.

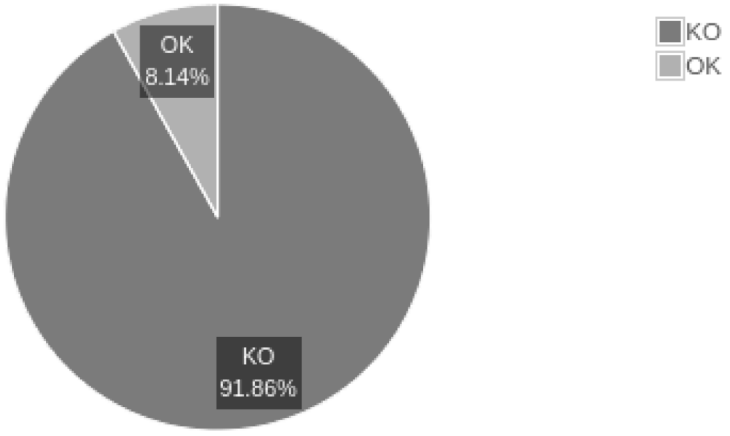

FIGURA 5.17: Respuestas exitosas y no exitosas obtenidas.

Con un archivo de tamaño 5MB como carga útil, se obtuvieron las métricas que se muestran en la Figura 5.18. El porcentaje de peticiones fallidas es de un 91.86 por ciento, siendo el tiempo mínimo de respuesta en atender una petición de 25 ms lo cual implica un promedio de cinco veces más que con una carga de 2 MB. La petición que tomó más tiempo en ser completada tuvo una duración de 959508 ms (16 min aprox), las peticiones efectuadas por segundo miden 8.8 una diferencia mínima respecto a lo obtenido con un archivo de carga de 2 MB.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**9186**	**91.86%**	**820955.31**	**25**	**959508**	**938462.50**	**944064.60**	**944877.75**	**946045.00**	**8.88**	**30.17**	**3716.34**
HTTP Request	10000	9186	91.86%	820955.31	25	959508	938462.50	944064.60	944877.75	946045.00	8.88	30.17	3716.34

FIGURA 5.18: Estadísticas (autoría propia)

Las peticiones fallidas son más de un 90 por ciento lo cual nos indica que cuanto más grande es la carga útil en las peticiones más fallas ocurren.Las causas reportadas por la herramienta que oficia de guía son las presentadas en la Figura 5.19, los errores son los mismos obtenidos en pruebas anteriores por lo que nos da una pauta de que las causas no varían demasiado.
Casi el 80 por ciento de las peticiones fallidas reportaron un error de escritura en el socket de conexión, este tipo de errores ocurren cuando la conexión se cierra por alguna de las partes cliente o servidor. El tiempo en el cual se escriben los datos no es suficiente para la carga útil total que tiene la petición, por lo cual se cierra la conexión quedando incompleta la operación.

Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset by peer (Write failed)	7999	87.08%	79.99%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection timed out (Write failed)	1003	10.92%	10.03%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Broken pipe (Write failed)	184	2.00%	1.84%

FIGURA 5.19: Tipos de errores reportados durante la prueba (autoría propia)

Los tiempos de respuesta de las peticiones que resultaron exitosas (aproximadamente 9 %) oscilan entre 500 ms y 1500 ms, teniendo un porcentaje elevado las que superan los 1500 ms.

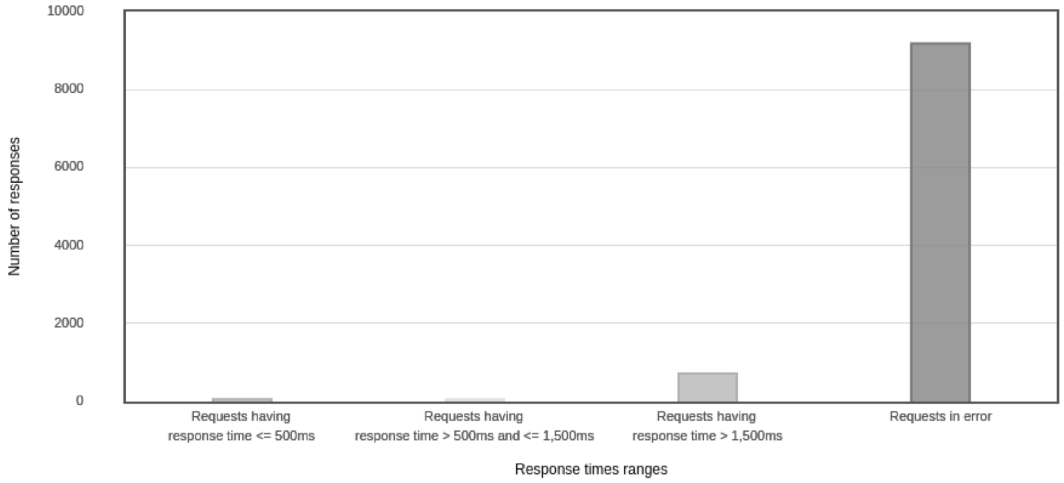

FIGURA 5.20: Tipos de errores reportados durante la prueba (autoría propia)

Las métricas obtenidas se simulan en un escenario local, las mismas dan una idea acerca del comportamiento de la librería con archivos de diferente tamaño utilizada en el proyecto SELI.
A continuación se presentan los resultados obtenidos en IPFS planteando distintos escenarios , si bien el objetivo no es realizar un análisis de performance, se pretende mostrar resultados cuantitativos que visualicen tanto en Meteor como en IPFS la carga de archivos.

5.2.2. IPFS

Con el propósito de obtener resultados para los casos de uso con IPFS, la simulación se realiza utilizando la misma librería http-lib-ipfs que se utiliza en el proyecto SELI. Todos los casos de uso que se presentan a continuación utilizan los mismos archivos que en las pruebas ejecutadas con la aplicación basada en el framework Meteor.
La diferencia con las pruebas en Meteor es que los archivos se suben a la

red pública IPFS la cual utiliza los servicios de Infura como infraestructura principal.

Primer caso

Para el caso uno se probaron las conexiones con 10000 usuarios en simultáneo con un archivo de 20 Kb. Los resultados obtenidos fueron exitosos dado que no se reporta ningún error y las peticiones lograron ser un 100 por ciento óptimas.

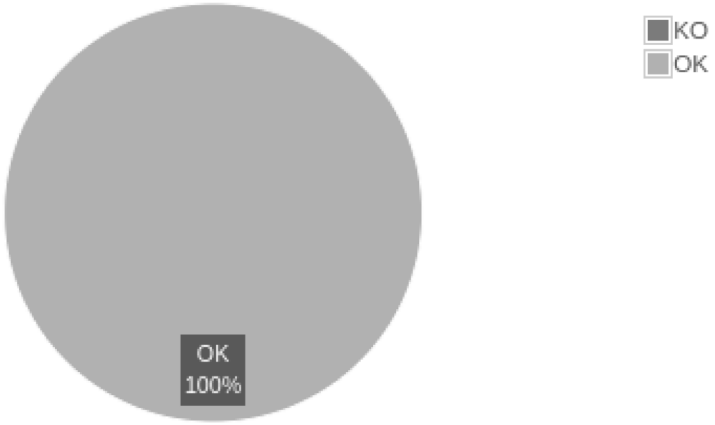

FIGURA 5.21: Peticiones al servidor Infura.

La Figura 5.22 muestra las métricas obtenidas luego de ejecutar la carga de un archivo 20 KB con 10000 usuarios concurrentes utilizando IPFS, el tiempo mínimo en atender una petición fueron 583 ms lo que equivale a 0.583 segundos, mientras que el tiempo máximo en el cual se logra completar una petición 11555 milisegundos equivalente a 11.555 segundos.
El servidor Infura logra resolver 178 peticiones por segundo, lo cual si lo comparamos con los resultados obtenidos al ejecutar el mismo caso de uso utilizando Meteor se obtuvieron 293.12 peticiones resueltas por segundo, si bien el resultado con Meteor muestra una mejora de casi un 60 % respecto al obtenido con IPFS, cabe destacar el contexto en el cual se ejecutaron ambos casos. Las pruebas en METEOR son realizadas en un contexto local sin injerencia de demoras o problemas de carga con la red. Mientras que con IPFS la situación es diferente puesto que como utiliza el servicio Infura de red pública las demoras o latencias en red pueden ser influyentes a la hora de obtener resultados.

El tiempo estimado en atender las peticiones se encuentra dividido en dos grupos, el primer grupo agrupa un estimado 5900 peticiones en un rango entre 500 ms y 1500 ms. El resto se completaron en un estimado de tiempo mayor a 1500 ms.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**0**	**0.00%**	**1982.13**	**583**	**11555**	**1366.00**	**4036.30**	**6166.00**	**7730.98**	**178.37**	**2755.34**	**31.70**
HTTP Request	10000	0	0.00%	1982.13	583	11555	1366.00	4036.30	6166.00	7730.98	178.37	2755.34	31.70

FIGURA 5.22: Métricas obtenidas durante la prueba (autoría propia)

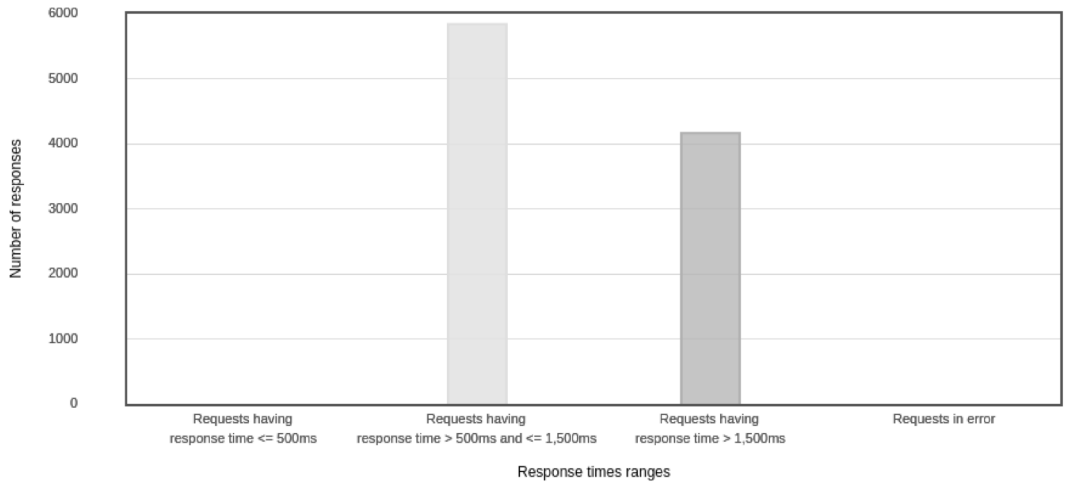

FIGURA 5.23: Tiempos de respuesta obtenidos (autoría propia)

Segundo caso

Como segundo caso de prueba se aumenta el tamaño de archivo a 2 MB tal cual se hizo con las pruebas en Meteor. En principio se comenzó con 10000 usuarios concurrentes, pero el servidor el cual recibe las peticiones IPFS no soporto las conexiones simultáneas con la carga útil de 2MB, por lo que las respuestas obtenidas resultaron en su totalidad todas fallidas. En consecuencia se fue particionando la cantidad de usuarios en grupos de menor tamaño hasta lograr alguna respuesta exitosa. Lo máximo obtenido que se pudo probar fueron 400 usuarios conectados con un archivo de 2 MB. Los resultados obtenidos se analizan a continuación, siendo el número de porcentaje de error de un 6.75 porciento como se muestra en la Figura 5.24.

Las métricas obtenidas se muestran en la Figura 5.25, el mínimo de tiempo en el cual se completa una petición es de 25717 milisegundos equivalentes a 25.717 segundos o sea 0.42 minutos. El máximo tiempo en completar una petición se logra realizar en 714522 milisegundos equivalentes a 11.9 minutos.

Se logra un rendimiento de 0.56 operaciones por segundo, marcando una fuerte diferencia entre lo medido en el primer caso con una carga útil de 20Kb que logra procesar 178.37 transacciones por segundo.

Las transacciones se procesan en un tiempo mayor a 1500 milisegundos según reporte de la Figura 5.26. El resto de las transacciones fallidas, el 13.25 porciento se debieron en su mayoría a errores de reseteo de conexión el cual se dio a lo largo de todas las pruebas hechas con Meteor e IPFS de las cuales

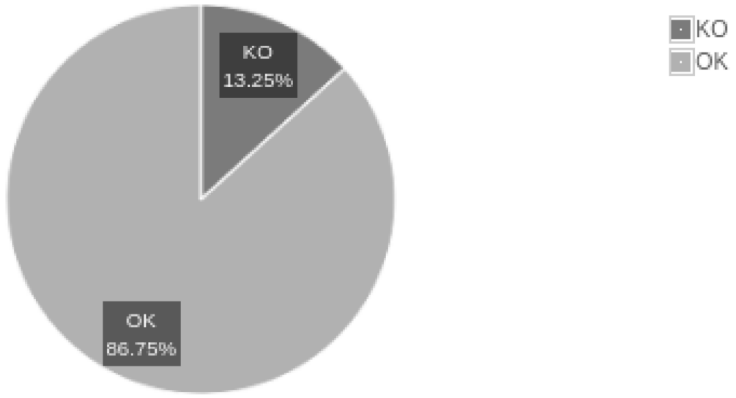

FIGURA 5.24: Porcentaje de peticiones exitosas y no exitosas durante la prueba(autoría propia)

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**400**	**53**	**13.25%**	**290264.64**	**25717**	**714522**	**302763.50**	**352680.90**	**359116.25**	**714110.23**	**0.56**	**997.03**	**0.09**
HTTP Request	400	53	13.25%	290264.64	25717	714522	302763.50	352680.90	359116.25	714110.23	0.56	997.03	0.09

FIGURA 5.25: Métricas (autoría propia)

mostraron fallas. El 1.89 porciento del total de errores es del tipo “Premature End of Content-Length” lo cual implica que el servidor espera determinada cantidad de bytes y recibe menos de lo esperado.

Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: javax.net.ssl.SSLException/Non HTTP response message: Connection reset	34	64.15%	8.50%
Non HTTP response code: javax.net.ssl.SSLException/Non HTTP response message: Socket closed	8	15.09%	2.00%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 2,101,546; received: 1,298,349)	1	1.89%	0.25%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 2,101,546; received: 1,826,559)	1	1.89%	0.25%

FIGURA 5.26: Tiempo de respuestas obtenidas.

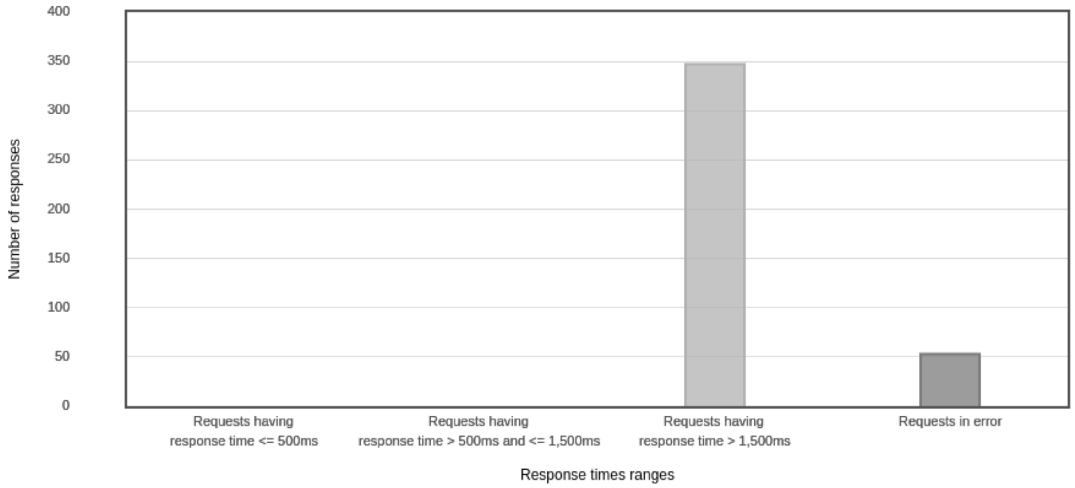

FIGURA 5.27: Errores reportados durante la prueba.

Tercer caso

Para el tercer caso, se aumentó el tamaño de archivo a 5 MB tal cual el tercer caso que se ejecutó con Meteor. El máximo número de usuarios concurrentes con el que se pudo ejecutar con una carga útil de 5MB fueron de 150, según Figura 5.28 un 64 porciento son peticiones completas mientras que un porcentaje de 36 resultaron ser peticiones fallidas.

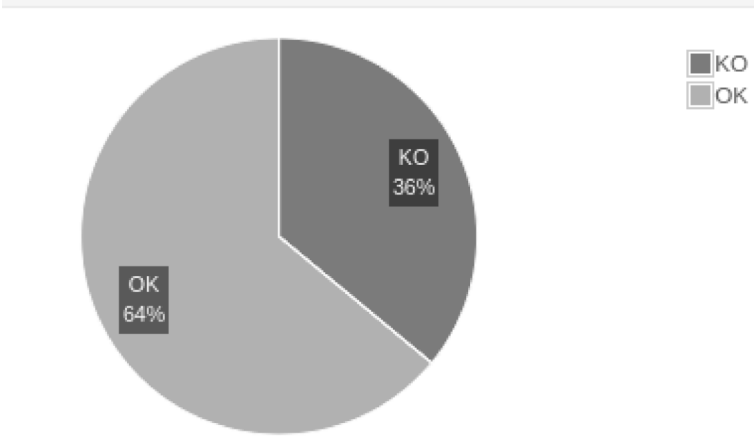

FIGURA 5.28: Errores reportados durante la prueba.

Las métricas obtenidas durante la prueba con una carga útil de 5MB para un total de 150 usuarios concurrentes, son las que muestra la Figura 5.29, donde el tiempo mínimo en completar una petición se da en 133440 milisegundos equivalentes a 2.224 minutos y el máximo tiempo empleado es de 657741 milisegundos equivalentes a 10.96 minutos.
Con respecto al segundo caso de prueba el cual la carga útil es de 2MB los máximos y mínimos obtenidos no marcan una notoria diferencia a pesar de los 3 MB de carga útil adicionales que se sumaron en este tercer caso de prueba. La cantidad de transacciones por segundo que se logra procesar mide 0.23

operaciones por segundo, casi la mitad de lo que fue medido en el caso de prueba dos.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**150**	**54**	**36.00%**	**330421.64**	**133440**	**657741**	**329829.00**	**399381.00**	**657242.40**	**657670.62**	**0.23**	**750.77**	**0.03**
HTTP Request	150	54	36.00%	330421.64	133440	657741	329829.00	399381.00	657242.40	657670.62	0.23	750.77	0.03

FIGURA 5.29: Métricas obtenidas durante la prueba. (autoría propia)

El 36 por ciento de los errores producidos durante la prueba son del tipo "Connection Reset", error común que se ha presentado en todos los casos donde se produjeron errores, durante las pruebas ejecutadas.

Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: javax.net.ssl.SSLException/Non HTTP response message: Connection reset	39	72.22%	26.00%
Non HTTP response code: javax.net.ssl.SSLException/Non HTTP response message: Socket closed	11	20.37%	7.33%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 5,266,467; received: 4,800,004)	1	1.85%	0.67%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 5,266,467; received: 4,368,131)	1	1.85%	0.67%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 5,266,467; received: 5,179,856)	1	1.85%	0.67%
Non HTTP response code: org.apache.http.ConnectionClosedException/Non HTTP response message: Premature end of Content-Length delimited message body (expected: 5,266,467; received: 5,078,925)	1	1.85%	0.67%

FIGURA 5.30: Errores reportados durante la prueba.

Las peticiones se logran atender en un tiempo mayor a 1500 ms tal cual se reportó en el segundo caso de prueba con un archivo 2 MB.

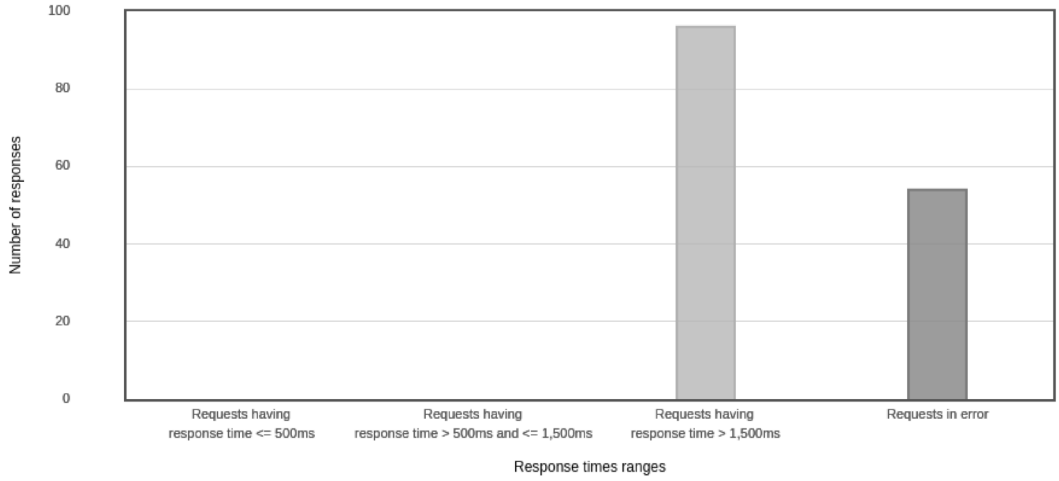

FIGURA 5.31: Tiempo de respuestas.

5.2.3. Comparación Meteor IPFS archivo de 500 Kb

En esta sección se analizan los resultados obtenidos con una carga útil de tamaño intermedio 500Kb. Las pruebas anteriores son realizadas con archivos desde 20Kb hasta 5MB, las métricas analizadas mostraron resultados vulnerables en especial cuando se trabaja con archivos a 5MB. A medida que aumenta la carga el porcentaje de peticiones fallidas aumenta considerablemente en comparación con aquellas pruebas realizadas con carga útil mínima como 20Kb.
Para un archivo de 500Kb las peticiones exitosas fueron de un 100 por ciento tanto con Meteor como con IPFS. A lo largo de todas las pruebas ejecutadas los resultados obtenidos se vieron afectados cuando se aumenta considerablemente la carga útil de las peticiones, por tal motivo para poder ejecutar algunas pruebas con la misma carga útil tanto en Meteor como en IPFS, se debió reducir en las pruebas ejecutadas en IPFS el número de usuarios concurrentes.
En el presente caso el máximo número de usuarios concurrentes soportados por el servidor público Infura de IPFS para una carga útil de 500Kb es de 1500 usuarios.

En la Figura 5.32 se muestran los resultados exitosos en ambos casos tanto con Meteor como con IPFS, el resultado fue de un 100 por ciento con una carga útil de 500Kb con 1500 usuarios concurrentes. El tiempo de respuesta obtenido durante la ejecución presenta tres grupos de rangos de tiempo bien definidos en Meteor, como muestra la Figura 5.33, en Meteor la mayoría de las peticiones logran resolverse en un rango de 500 milisegundos, le sigue las peticiones que varían entre 500 y 1500 milisegundos y en menor número aquellas que se resuelven en un periodo mayor a 1500 milisegundos. Por otro lado con IPFS se evidencia un solo rango de tiempo mayor a 1500 milisegundos para el cual las peticiones se procesan.

Las métricas obtenidas en Meteor para el presente caso de uso, muestran en la Figura 5.34 el tiempo mínimo en completar una petición se da en 6 milisegundos mientras el tiempo máximo que le toma en resolver una petición

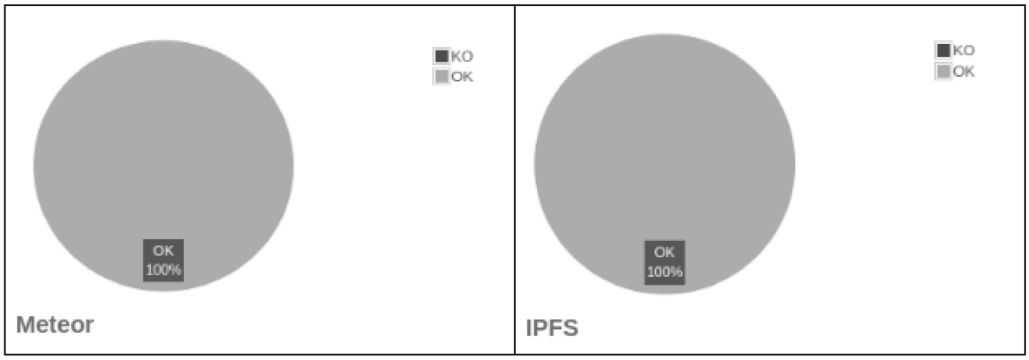

FIGURA 5.32: Peticiones exitosas y no exitosas en Meteor (izq) e IPFS (der). (autoría propia)

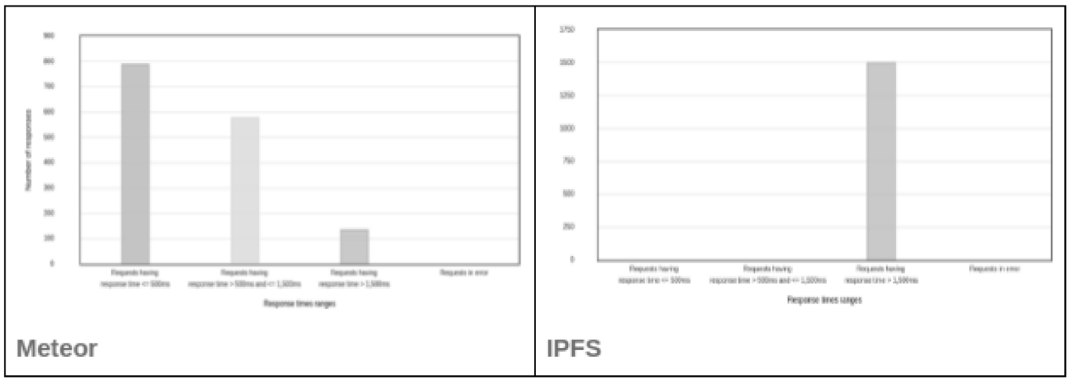

FIGURA 5.33: Tiempo de respuestas obtenidas con Meteor (izq) e IPFS (der). (autoría propia)

es de 1798 milisegundos, lo equivale a 1.798 segundos.

La cantidad de transacciones por segundo que se logra atender son de 132.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1500**	**0**	**0.00%**	**574.91**	**6**	**1798**	**471.50**	**1470.90**	**1598.90**	**1728.00**	**132.33**	**1261.95**	**66227.23**
HTTP Request	1500	0	0.00%	574.91	6	1798	471.50	1470.90	1598.90	1728.00	132.33	1261.95	66227.23

FIGURA 5.34: Métricas obtenidas en la prueba con Meteor. (autoría propia)

En IPFS los resultados medidos no fueron tan óptimos como lo fue con Meteor, si bien en Meteor las pruebas se dieron a nivel local, hay que tener en cuenta que para correr el caso de uso en IPFS se ejecuta en una red pública IPFS cuyo servidor posee una limitante o para evitar ataques o posible sobrecarga de peticiones.

El mínimo de tiempo obtenido en resolver una petición se da en 3549 milisegundos, lo que es un equivalente de 3.549 segundos casi el triple de tiempo del máximo obtenido en Meteor.
El máximo medido es de 119031 milisegundos equivalentes a 1.98 minutos que le toma procesar una petición.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1500**	**0**	**0.00%**	**54061.16**	**3549**	**119031**	**54561.00**	**79360.90**	**88731.40**	**104143.21**	**9.81**	**4912.26**	**1.74**
HTTP Request	1500	0	0.00%	54061.16	3549	119031	54561.00	79360.90	88731.40	104143.21	9.81	4912.26	1.74

FIGURA 5.35: Métricas obtenidas en la prueba con Meteor. (autoría propia)

Conclusiones

Los resultados obtenidos en las pruebas ejecutadas con Meteor e IPFS muestran diferencias a medida que la carga útil de las peticiones aumenta. Generalmente se observa que cuanto más grandes son los archivos a procesar mayor es el tiempo de respuesta por parte del servidor. Si bien en la mayoría de los casos se obtuvieron mejores resultados con Meteor, se debe de tener en cuenta que se ejecutaron en un nodo local sin tener que acudir a alojar a un servidor externo, con esto le quita latencia, errores de conexión etc. En IPFS no es posible correr en un nodo local dado que la red de prueba elegida es pública conectada con otros nodos IPFS, si bien estas diferencias no son tan relevantes, es idóneo tenerlo en cuenta como posible variable externa al realizar las pruebas. El objetivo de la presente sección no es realizar ningún tipo de comparación de performance entre Meteor e IPFS, sino se centra en observar gráficamente el comportamiento al variar escenarios de carga en ambas.
Los errores que se presentan en su gran mayoría son errores de conexión o

tiempo en el cual se tarda en escribir los datos al socket de conexión, esta situación resulta de utilidad, se puede realizar configuraciones pertinentes que mitiguen en su mayoría esta situación.

Capítulo 6

Conclusiones y trabajos futuros

6.1. Conclusiones del trabajo

En esta tesis se ha presentado el estudio del protocolo IPFS como una alternativa para la distribución de archivos en la web, en especial se presenta la incorporación de IPFS dentro del proyecto SELI.
Si bien no resultó exitosa su incorporación debido a incompatibilidades con el framework METEOR sobre el cuál está implementada la plataforma de SELI actualmente, se obtuvieron resultados que permitieron analizar el impacto de utilizar IPFS como protocolo alternativo para la distribución de archivos dentro de distintos escenarios para futuros desarrollos de la plataforma SELI u otras similares.

En la sección 5.2 se proponen diferentes escenarios utilizando distintos tamaños de archivos cuyo objetivo es medir el tiempo de carga de estos archivos utilizando METEOR e IPFS. Se obtuvieron resultados variables, observando que IPFS con archivos de gran tamaño presenta una demora considerable en su distribución dentro de la plataforma de prueba expuesta en la sección 5.2. Este punto es uno de los obstáculos que aún IPFS tiene como desafío mejorar.

En el caso de SELI, la utilidad de IPFS dentro de la plataforma no logró verse reflejada debido a los obstáculos mencionados en la sección 5, pero al menos se identificaron los problemas actuales que presenta SELI e IPFS y los posibles escenarios de mejora para el almacenamiento. Si bien IPFS puede ser una alternativa para solucionar el problema de almacenamiento de los recursos educativos en los servidores de SELI, se debe considerar las limitaciones actuales a las cuales IPFS se enfrenta, tal como el comportamiento de IPFS ante archivos de tamaño mayor a 2 MB según lo presentado en la sección 5.2, de manera que los intercambios en la plataforma sean de forma ágil y dinámica.

6.2. Algunas conclusiones generales

Una de las principales conclusiones obtenidas es la corroboración de la falta de madurez de la comunidad de IPFS, especialmente notorio en

la escasa documentación a nivel técnico existente, lo que hizo realmente difícil entender el mecanismo del protocolo.
Al tener tan pocas guías de desarrollo o manuales, se recurrió a foros de discusión sobre IPFS como principal fuente de información y ayuda, así como también a artículos académicos publicados por investigadores en el área, conferencias dictadas por Benet y presentaciones realizadas por parte del equipo de Protocol Labs en jornadas tecnológicas que organizan empresas que ofician de sponsors del proyecto.

La página oficial del sitio de IPFS [29] es una fuente de información con una documentación muy básica, generalmente la mejor fuente encontrada han sido sitios de github que alojan proyectos basados en el protocolo IPFS en los cuales desarrolladores de la comunidad IPFS evacuan dudas y optimizan el proyecto haciendo que IPFS se adecue a las herramientas web actuales para ser utilizado en proyectos globales.
Actualmente las librerías implementadas para integrar IPFS en proyectos, no están lo suficientemente maduras a nivel técnico, la comunidad IPFS aún está muy lejos de ser un referente y abundan desarrolladores con muy poca experiencia en el tema como para ayudar a contestar dudas del núcleo duro del protocolo.
Los autores de este protocolo intentan difundir el emprendimiento a través de la realización de capacitaciones en diferentes universidades, de estos se logró obtener una base como ayuda para realizar el trabajo.

Las arquitecturas de software mencionadas en esta tesis presentan sus ventajas y desventajas, dependiendo del contexto en el cual se apliquen algunas son más apropiadas que otras. Tal es el caso de Blockchain e IPFS que pueden tener un muy buen alcance, pero aún así queda un largo camino por recorrer.

El trabajo ***The Main Limitations of Applying Blockchain Technology in the Field of Education*** **[46]**, pone en manifiesto las debilidades que presenta la tecnología Blockchain en el campo de la educación. Considerando el éxito obtenido por Blockchain en el contexto financiero gracias al buen funcionamiento del Bitcoin [45], se analiza el uso de esta tecnología en otros contextos como ser por ejemplo el rubro educativo.
Para que una red Blockchain se mantenga funcional se necesita una gran cantidad de nodos participantes, el gran éxito del Bitcoin se debe a la motivación que los participantes mantienen según el valor de la criptomoneda. Los autores del artículo [46] plantean un posible escenario en el contexto educativo, una red Blockchain compuesta por diferentes universidades.
Debido a que en una Blockchain cada nodo contiene la información de todas las transacciones que existen en la red, las universidades que la integran deben compartir los datos de sus estudiantes con otras universidades, lo cual compromete información sensible como ser el historial del estudiante en la universidad, sus datos personales etc. Para mantener una red Blockchain con

las características mencionadas, los autores cuestionan la motivación principal por la cual las universidades invierten recursos en almacenar información de otras universidades que no son de su interés. Este planteo, conlleva a un análisis de los escenarios para el cual la tecnología Blockchain puede ser una buena opción para implementar. Lo principal es la motivación que los participantes tienen en integrar una red con estas características así como los recursos que disponen dado que puede resultar caro e ineficiente si no se tiene un propósito común con el resto de los participantes. Blockchain no es una tecnología aplicable en todos los contextos, por lo que hay que analizar las ventajas y desventajas de dicha tecnología y el interés en común que exista para que sea beneficioso su uso.

IPFS tiene como principal objetivo cambiar la forma de interacción de la web. Si bien hoy se encuentra en una etapa alfa de desarrollo, se está evolucionando su desarrollo para lograr mitigar el problema de permanencia de datos en la web.

Juan Benet [5] es un gran impulsor de la descentralización de información y la no censura, motivando a que cada vez sean más los usuarios que se animen a cooperar en intercambiar información a través de la utilización de aplicaciones descentralizadas basadas en arquitecturas peer-to-peer. IPFS promueve la no censura de información alentando a alojar sitios web basados en IPFS que aseguran la permanencia de información carente de una autoridad centralizadora. Sin duda alguna IPFS está cambiando la forma de interactuar en la web, cada vez son más los proyectos que buscan integrar IPFS a su solución, lo cual se potencia con el proyecto Filecoin del mismo autor, abriendo un nuevo mercado, donde por primera vez el cliente firma un contrato con un agente como garantía. No solo es un nuevo mercado, sino que además genera una nueva forma de pensar la economía. A través de la moneda digital Filecoin (FIL), como incentivo a generar ganancias mientras se aprovechan recursos de hardware de aquellos que desean sacar beneficio del espacio libre en sus dispositivos.

6.3. Trabajos futuros

A lo largo de esta tesis hemos descrito el diseño y funcionamiento del protocolo IPFS, así mismo analizamos su viabilidad en la incorporación de este al proyecto SELI, evaluando el comportamiento del protocolo en tres escenarios diferentes. El primer escenario incorpora IPFS en el manejo de imágenes de perfil de usuario, el segundo plantea la idea de configurar un IPFS cluster [36] entre las instituciones educativas y como tercer escenario y trabajo a futuro, implementar la plataforma D-SELI descentralizada basada en IPFS.

Tomar al proyecto SELI como caso de estudio es un buen ejercicio, en especial porque el objetivo del proyecto es ser una plataforma de inclusión educativa abierta, proporcionando acceso a materiales y cursos que faciliten y ayuden al estudiante en su proceso de aprendizaje, y por último como

desafío mantener a los usuarios incentivados para seguir utilizando la plataforma.

Por lo anteriormente dicho en el contexto el cual se presenta SELI, promover el desarrollo de la plataforma D-SELI basada en una arquitectura descentralizada, utilizando IPFS como protocolo de almacenamiento de recursos, junto con la idea de crear un token propio como incentivo, puede generar buenos resultados a corto plazo.

Bibliografía

[1] Florian Adamsky, Syed Ali Khayam, Rudolf Jäger, and Muttukrishnan Rajarajan. Security analysis of the micro transport protocol with a misbehaving receiver. In *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 143–150. IEEE, 2012.

[2] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. .ºReilly Media, Inc.", 2014.

[3] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8. IEEE, 2007.

[4] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[5] Juan Benet. jbenetś blog - juan benet. `https://juan.benet.ai/`, 2018. [Fecha de última visita: 01/08/2021].

[6] Juan Benet. Ipfs specs. `https://github.com/wking/ipfs-specs`, 2021. [Fecha de última visita: 15/08/2021].

[7] Juan Benet and Nicola Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, pages 1–36, 2018.

[8] W Dean Bidgood Jr, Steven C Horii, Fred W Prior, and Donald E Van Syckle. Understanding and using dicom, the data interchange standard for biomedical imaging. *Journal of the American Medical Informatics Association*, 4(3):199–212, 1997.

[9] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[10] Cristian Cechinel, Alicia Viviana Diaz, Christiano Ávila, Regina Motz, Ellen Mendez, Maria García, and Patricia Diaz. Una experiencia de un curso online sobre recursos educativos abiertos en el marco del programa de apoyo al sector educativo del mercosur. *Anais temporários do LACLO 2015*, 10(1):286, 2015.

[11] Paulo Mazzoncini de Azevedo-Marques and Samuel Covas Salomão. Pacs: sistemas de arquivamento e distribuição de imagens. *Revista Brasileira de Física Médica*, 3(1):131–139, 2009.

[12] Reyna Der Boghosian. Github reyna der boghosian. `https://gitlab.fing.edu.uy/reyna.karine.der/ipfs-en-plataforma-seli`, 2020.

[13] David Dias. Docs: Add ipfs architecture diagram by daviddias. https://github.com/ipfs/js-ipfs/pull/1673, 2017. [Fecha de última visita: 10/03/2021].

[14] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Semantics and content, 2014.

[15] Linda Frederiksen. Digital badges. *Public Services Quarterly*, 9(4):321–325, 2013.

[16] Robert W Gehl. *Weaving the dark web: legitimacy on freenet, Tor, and I2P*. MIT Press, 2018.

[17] GoLang. The go programming language. (s. f.). https://golang.org/, 2021. [Fecha de última visita: 14/04/2021].

[18] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.

[19] Hackernoon. A complete analysis of the distributed web | hacker noon. https://hackernoon.com/ipfs-a-complete-analysis-of-the-distributed-web-6465ff029b9b, 2020. [Fecha de última visita: 23/08/2020].

[20] Stephan Hochhaus and Manuel Schoebel. *Meteor in action*. Simon and Schuster, 2015.

[21] Infura. Ethereum, a. p. i. & ipfs, a. gateway eth nodes as a service infura. https://infura.io/, 2015. [Fecha de última visita: 20/02/2021].

[22] Florian Jacob, Jens Mittag, and Hannes Hartenstein. A security analysis of the emerging p2p-based personal cloud platform maidsafe. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1403–1410. IEEE, 2015.

[23] ICEA Keranen. Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal draft-ietf-ice-rfc5245bis-20. 2018.

[24] Gary C Kessler. An overview of tcp/ip protocols and the internet. *Gary-Kessler website*, 2007.

[25] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[26] Mirja Kuehlewind. Low extra delay background transport (ledbat). *IETF RFC6817*, 2012.

[27] Randhir Kumar and Rakesh Tripathi. Implementation of distributed file storage and access framework using ipfs and blockchain. In *2019 Fifth International Conference on Image Information Processing (ICIIP)*, pages 246–251. IEEE, 2019.

[28] James Kurose and Keith W Ross. *Redes de computadoras*, volume 5. Pearson educación, 2010.

[29] Protocols Lab. Ipfs powers the distributed web. `https://ipfs.io/`, 2014. [Fecha de última visita: 08/07/2021].

[30] Protocl Labs. Bitswap protocol. `https://docs.ipfs.io/concepts/bitswap/`. [Fecha de última visita: 14/07/2021].

[31] Protocol Labs. ipfs/go-ipfs. `https://github.com/ipfs/go-ipfs`, 2014. [Fecha de última visita: 05/07/2021].

[32] Protocol Labs. How ipfs works». content addressing. `https://docs.ipfs.io/concepts/how-ipfs-works/`, 2015. [Fecha de última visita: 02/08/2021].

[33] Protocol Labs. How ipfs works». directed acyclic graphs (dags). `https://docs.ipfs.io/concepts/how-ipfs-works/`, 2015. [Fecha de última visita: 25/06/2021].

[34] Protocol Labs. Interplanetary linked data | ipld documentation. `https://docs.ipld.io/#what-is-ipld`, 2015. [Fecha de última visita: 17/04/2021].

[35] Protocol Labs. «lesson: Initialize your ipfs repository». `https://dweb-primer.ipfs.io/install-ipfs/initialize-repository`, 2015. [Fecha de última visita: 03/04/2021].

[36] Protocol Labs. Ipfs cluster. `https://cluster.ipfs.io/`, 2018. [Fecha de última visita: 01/07/2021].

[37] Protocol Labs. Ipfs documentation. `https://github.com/ipfs/ipfs-docs`, 2019. [Fecha de última visita: 17/08/2021].

[38] Protocols Labs. Working with ipfs in js. `https://docs.ipfs.io/reference/js/api/`, 2014. [Fecha de última visita: 20/05/2021].

[39] Taotao Li, Parhat Abla, Mingsheng Wang, and Qianwen Wei. Designing proof of transaction puzzles for cryptocurrency. *IACR Cryptol. ePrint Arch.*, 2017:1242, 2017.

[40] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). Technical report, RFC 5766, 2010.

[41] Ujjal Marjit and Prabhakar Kumar. Towards a decentralized and distributed framework for open educational resources based on ipfs and blockchain. In *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, pages 1–6. IEEE, 2020.

[42] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[43] Aldo Mejía. No es un simulacro, google se cayó y gmail presenta fallas. televisa news. https://www.televisa.com/noticias/no-es-un-simulacro-google-se-cayo-y-gmail-presenta-fallas/, 2021. [Fecha de última visita: 14/07/2021].

[44] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[45] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[46] OA Naumova, IA Svetkina, and DV Naumov. The main limitations of applying blockchain technology in the field of education. In *International Science and Technology Conference EastConf*, pages 1–4. IEEE, 2019.

[47] Javier Nava. «javascript - ¿qué significa el error "uncaught typeerror: Cannot set/read property «xxx» of undefined/null" y cómo solucionarlo?». https://es.stackoverflow.com/questions/169194/qu%c3%a9-significa-el-error-uncaught-typeerror-cannot-set-read-property-xxx-of-u 2020. [Fecha de última visita: 18/08/2021].

[48] The Blockchain Education Network. Long term info-structure - the long now. https://longnow.org/seminars/02018/aug/06/long-term-info-structure/, 2018. [Fecha de última visita: 21/05/2021].

[49] The Blockchain Education Network. Blockchain education network. https://blockchainedu.org/, 2019. [Fecha de última visita: 16/04/2021].

[50] Kien Nguyen, Thinh Nguyen, and Yevgeniy Kovchegov. A p2p video delivery network (p2p-vdn). In *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, pages 1–7. IEEE, 2009.

[51] ODEM.IO. Program staking & token architecture v1.2 at 1.technical whitepaper. https://odem.io/docs/whitepaper/ODEM.IO-Technical-Whitepaper.pdf, 2018. [Fecha de última visita: 15/06/2021].

[52] Maxwell Ogden, Karissa McKelvey, Mathias Buus Madsen, et al. Dat-distributed dataset synchronization and versioning. *Open Science Framework*, 10, 2017.

[53] Nigini Oliveira, Michael Muller, Nazareno Andrade, and Katharina Reinecke. The exchange in stackexchange: Divergences between stack overflow and its culturally diverse participants. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–22, 2018.

[54] Thomas Osterland and Thomas Rose. Model checking smart contracts for ethereum. *Pervasive and Mobile Computing*, 63:101129, 2020.

[55] Solomon Sunday Oyelere, Umar Bin Qushem, Vladimir Costas Jauregui, Özgür Yaşar Akyar, Łukasz Tomczyk, Gloria Sanchez, Darwin Munoz, and Regina Motz. Blockchain technology to support smart learning and inclusion: pre-service teachers and software developers viewpoints. In *World Conference on Information Systems and Technologies*, pages 357–366. Springer, 2020.

[56] Venkata N Padmanabhan, Helen J Wang, and Philip A Chou. Resilient peer-to-peer streaming. In *11th IEEE International Conference on Network Protocols, 2003. Proceedings.*, pages 16–27. IEEE, 2003.

[57] M Pavithra, S Vasanth, R Rajmohan, and D Jayakumar. Zeronet: An overview. *International Journal of Pure and Applied Mathematics*, 119(14):1347–1351, 2018.

[58] Labs. Protocols. «what is ipfs?» decentralization». `https://docs.ipfs.io/concepts/what-is-ipfs/`, 2015. [Fecha de última visita: 10/06/2021].

[59] Labs. Protocols. «what is ipfs?» decentralization». `https://github.com/ipfs/camp`, 2019. [Fecha de última visita: 08/07/2021].

[60] ProtoSchool. «multiformats tutorial | anatomy of a cid (lesson 2)». `https://proto.school/anatomy-of-a-cid/02/`, 2018. [Fecha de última visita: 02/05/2021].

[61] Yiannis Psaras and David Dias. The interplanetary file system and the filecoin network. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 80–80. IEEE, 2020.

[62] Siraj Raval. *Decentralized applications: harnessing Bitcoin's blockchain technology*. .ºReilly Media, Inc.", 2016.

[63] João Santos and Kim Hamilton Duffy. A decentralized approach to blockcerts credential revocation. *A White Paper from Rebooting the Web of Trust V*, 2018.

[64] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE, 2001.

[65] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[66] SELI Team. Seli project». smart ecosystem for learning and inclusion. `https://seli.uazuay.edu.ec/`, 2020. [Fecha de última visita: 21/08/2021].

[67] Antonio Tenorio-Fornés, Samer Hassan, and Juan Pavón. Open peer-to-peer systems over blockchain and ipfs: An agent oriented framework. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 19–24, 2018.

[68] Nicolás Verdejo. "estadísticas con las que youtube recibe el 2020". https://wwwhatsnew.com/2019/12/25/estadisticas-con-las-que-youtube-recibe-el-2020/, 2021. [Fecha de última visita: 17/06/2021].

[69] YouTube. About youtube. https://www.youtube.com/intl/en-GB/about/, 2021. [Fecha de última visita: 16/04/2021].

[70] Yinghui Zhao, Ru An, Dongyang Ou, and Congfeng Jiang. An interplanetary file system based picture archiving and communication system. In *2020 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5. IEEE, 2020.

[71] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress)*, pages 557–564. IEEE, 2017.

Glosario

API (del inglés: API Application programming interface) Es un conjunto de funcionalidades que conforman un software..

BITSWAP Protocolo de intercambio de bloques en una red IPFS..

BLOCKCHAIN Un modelo distribuido basado en una cadena de bloques donde cada bloque contiene una copia exacta de todas las transacciones así como una referencia al bloque previo que le precede en la cadena..

BLOQUE GENESIS Se lo define como bloque cero de una Blockchain, en el cual todos los nodos al conectarse por primera vez en una red Blockchain se deben unir a él..

BLOQUE IPFS Nodo que corre el protocolo IPFS..

CHORD Protocolo para la implementación de tablas DHT en una red distribuida..

CHUNKING Proceso mediante el cual se fracciona un archivo en bloques antes de agregarse a una red IPFS..

CHUNKS Bloques IPFS de tamaño 256 KB..

CID (Content Identification) es el hash identificador de un bloque IPFS según contenido del bloque..

CONMUTADOR (del inglés: SWITCH) Es un componente de libp2p responsable de encapsular múltiples protocolos de transportes en una sola interfaz, lo que permite que libp2p se adapte a cualquier protocolo de transporte..

DAG DAG (Directed acyclic graph) es un tipo de estructura de datos basada en un grafo dirigido que no contiene ciclos..

DAPPS Son aplicaciones basadas en arquitecturas descentralizadas cuyo funcionamiento no depende de un nodo central..

DEMONIO (del inglés: Daemon) Es un proceso el cual se ejecuta en segundo plano que puede ser controlado por el usuario..

FIL Es un token que oficia como sistema de pago digital a los mineros dentro del proyecto FILECOIN..

FILECOIN Proyecto para almacenar información mediante una red descentralizada de almacenamiento..

HASH CRIPTOGRAFICO Es un algoritmo matemático que transforma un bloque de datos en en una nueva serie de caracteres de longitud fija..

INFURA Empresa de software que ofrece servicios de mantenimiento y equipamiento para redes Ethereum e IPFS..

IPFS Protocolo de distribución de archivos basado en una arquitectura distribuida peer-to-peer.

IPFS-CLUSTER Red de servidores IPFS..

LIBP2P Sistema modular responsable de enviar y recibir datos a los nodos en una red P2P, provee una interfaz multi transporte la cual se adapta a cualquier protocolo de transporte..

LIBRERIA Es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca..

MERKLE TREE Estructura tipo árbol donde cada nodo que no es hoja está identificado como la concatenación de las funciones hash de los hijos..

MINERO En el sistema de Blockchain, los mineros tienen el rol de la creación de nuevos bloques y la verificación de los bloques añadidos a la cadena..

MULTIHASH Es un protocolo que interpreta las salidas de hash criptográficos, en el caso de IPFS se interpreta como versión IPFS + tamaño del hash + codificación..

P2P Es un modelo de arquitectura distribuida donde no existe un nodo central, cada nodo se comporta como cliente-servidor..

P2P-VDN Sistema de streaming basado en un modelo de arquitectura P2P..

PEER Se le denomina peer (par) a los nodos que integran una red peer-to-peer (P2P)..

PINNING Proceso en el cual se le indica a IPFS que el archivo permanecerá en la red IPFS..

RCN Técnica que particiona en múltiples segmentos un video de transmisión en tiempo real y lo envía a una cantidad aleatoria de nodos en la red P2P..

RED IPFS Red de dispositivos conectados entre sí a través del protocolo IPFS..

SPF Single Point of Failure (punto único de fallo), se les dice a las componentes de un sistema que tras un desperfecto en el funcionamiento ocasiona un fallo total en el sistema..

TABLA DHT (Distributed hash table) Son un tipo de tablas de hash que almacenan pares de clave-valor, las cuales almacenan información de nodos cercanos entre sí en arquitecturas distribuidas como P2P..

Appendices

Apéndice A

Bitácora sobre Integración de IPFS al proyecto SELI

La integración de IPFS en el proyecto SELI surge de una reunión de análisis con el equipo técnico de SELI, donde se planteó la preocupación por el inminente crecimiento de cursos online dentro de la plataforma, provocando que los servidores destinados a almacenar los recursos en SELI se vieran colapsados por falta de espacio en los mismos. Para mitigar el problema, cada institución integrante de SELI se veía en la obligación de adquirir más recursos de hardware ante la creciente demanda. Tal situación insume un gasto económico adicional al presupuesto asignado para cada institución y no todas pueden afrontarlo.
A partir del problema de almacenamiento, fue cuando se comenzó a investigar cómo aumentar la capacidad de almacenamiento con los recursos disponibles hasta el momento.
De la investigación surgió la idea de ahondar en IPFS como algo innovador a implementar en la plataforma y a su vez como una posible nueva forma de interacción entre las instituciones cuyo objetivo se basa en aprovechar al máximo los recursos disponibles descentralizando la información entre los servidores disponibles.

Al principio cuando se planteó al equipo la idea de integrar IPFS, no convenció demasiado. El proyecto de sistemas de archivos interplanetario conceptualmente es difícil de entender a simple vista y la información con la cual se cuenta no logra poner en manifiesto para el usuario de forma clara su potencial, generando más dudas que certezas acerca de su alcance. Esto generó una gran resistencia por parte del equipo técnico para aplicar IPFS dentro de SELI, como consecuencia de ello se pospuso la idea de implementarlo en SELI, pero se acordó seguir con la investigación.
En los inicios, se realizaron varias presentaciones con demos[12], el objetivo fue presentar el mecanismo de trabajo del protocolo IPFS para así comprender el gran potencial que promete ser. Luego de varias reuniones y presentaciones, se decidió implementar IPFS en SELI. Para ello, como hemos detallado en las sección de caso de estudio, se implementó el primer escenario en una parte del proyecto. Se intentó integrar IPFS de tal forma que cuando los usuarios suben una foto de perfil lo hagan a través del protocolo. Lamentablemente no se pudo obtener éxito y por tal motivo se tuvo que repensar los dos escenarios siguientes. La gran variedad de librerías que existen están

en fase alfa, si bien la integración de IPFS a un proyecto web básico resultó ser exitosa, en SELI no resultó debido a la incompatibilidad con Meteor [20], framework en el cual se basa el proyecto SELI. Esto desmotivó bastante al equipo técnico, por las expectativas que se habían generado al respecto, pero aun así se decidió continuar con la investigación y surgieron dos nuevos posibles escenarios. La segunda propuesta fue implementar un cluster [36] IPFS de servidores entre las instituciones participantes, para así lograr distribuir la información y evitar cuellos de botella. Si bien se logró configurar los servidores, la tarea no fue nada sencilla. En primer lugar no había mucha información al respecto de cómo configurar los servidores para que funcionen como IPFS cluster. Luego de varias pruebas y configuraciones se logró documentar un manual [12], en él se detalla de qué forma se tiene que proceder para implementar la solución propuesta. El poco expertise por parte del equipo técnico hizo aún más arduo el trabajo, lo cual realmente dificulto la comunicación entre las instituciones participantes del desarrollo, descartando el presente escenario por lo que la configuración del cluster quedó sin efecto.

Por último, se propone en 5.1.4 implementar D-SELI, una plataforma descentralizada basada en el protocolo IPFS para intercambio de archivos.