**Proyecto Final de Curso**

**Aplicación Distribuida y Paralela**

Zabala Mariano

Legajo: 143116

[Marianol.zabala@gmail.com](mailto:Marianol.zabala@gmail.com)



Universidad Nacional de Luján

Lic. en Sistemas de Información

Sistemas distribuidos y Programación paralela (41409)

**Propuesta**

El servicio permite a los nodos participantes descargar un archivo desde múltiples peers simultáneamente, distribuyendo la carga de trabajo.

El servicio se centra en la descarga de archivos de gran tamaño. Permite detener la descarga en cualquier momento, continuando al iniciar el servicio nuevamente. Esto es posible al descargar un archivo por partes, e indicando qué partes posee y cuáles le faltan.

**Agentes**

* **Seed**: Son los hosts que tienen el 100% de un archivo disponible para descargar.
* **Downloaders**/**Leechers**: Host que se encuentra descargando archivos de otros peers y, a la vez, publican la parte que tienen del archivo a otros leechers.
* **Peers**: Los seeds y downloaders.
* **Tracker**: Nodo central al que los demás peers se conectan. Funcionan como servidores centralizados replicados, siendo al menos 2. Almacenan información sobre los archivos compartidos por peers, como su nombre, tamaño, y partes en la que está dividido. Además, mantiene una lista de los peers que descargaron o están descargando cada archivo (swarm).
* **Swarm**: conjunto de peers que descargaron o están descargando un archivo. A medida que un peer descarga partes de un archivo puede comenzar a compartirlas con los demás peers. Un peer descubre que otros peers pertenecen a un swarm por medio de un tracker.

**Arquitectura**

Está basado en una arquitectura híbrida entre un peer to peer y un cliente-servidor. Los peers necesitan de un servidor (tracker) para conocer qué archivos son ofrecidos en la red, y qué peers poseen partes o la totalidad del archivo que les interesa descargar.

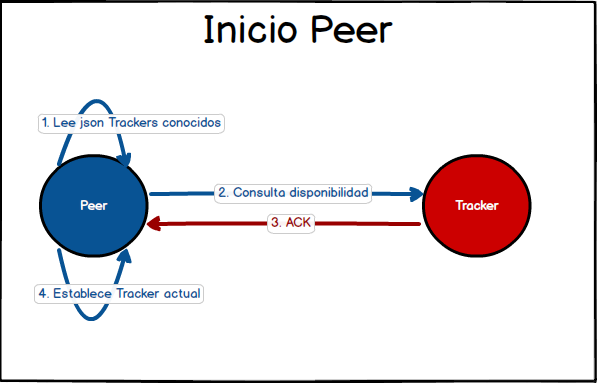
Cuando un peer inicia la aplicación puede conectarse con un tracker (servidor), el cual obtiene de un archivo que posee previamente, para:

- Subir pequeños archivos con meta-data de archivos que desea compartir. Para esto, se conecta a un tracker disponible y solicita el socket de escucha del tracker primario. Es al tacker primario a quien envía el META-Archivo.

- Descubrir qué archivos son ofrecidos en la red, y descargar el meta-archivo de un archivo de interés.

-Solicitar un swarm de un archivo específico, para poder llevar a cabo una descarga.

-Actualizar la lista de trackers conocidos.



**Imagen 1**

Una limitación de la arquitectura cliente-servidor, como la caída del tracker (servidor) que funciona como un índice centralizado de los archivos, es prevenida por medio del uso de múltiples trackers que funcionan en conjunto. Los trackers se conocen entre ellos y mantienen la misma información. Cuando los datos del tracker primario se actualizan luego de que un peer envíe un META-Archivo, realiza una replicación que llega a todos los trackers.

Otra limitación cliente-servidor tal como el cuello de botella, producido por las consultas de los peers contra el tracker, también es resuelta al utilizar N trackers. Los peers descubren qué archivos existen en la red y quienes los ofrecen consultando un tracker al azar. Facilita al peer la consulta por seeds. Solo debe preguntar a su tracker y no comenzar un flooding en busca de peers con los archivos requeridos.

Se decidio utilizar una arquitectura híbrida, y no puramente peer to peer, para delegar el almacenamiento de la meta-data a un agente centralizado. De esta forma, en lugar de llevar a cabo un flooding con cada nueva actualización, un peer solo debe enviar el nuevo META-Archivo al tracker primario y este se encarga de llevar a cabo la replicación, produciendo únicamente N-1 conexiones (donde N es el número de trackers).

**Publicar archivo**

El usuario indica el path del archivo que desea publicar. A partir de este se genera un JSON con la siguiente información:

* Nombre del archivo
* Path donde se encuentra el archivo real a compartir
* Tamaño del archivo
* Lista de partes que forman el archivo, de 1 a N. Todas las partes pesan 1MB, excepto la última que puede tener un peso menor o igual a 1MB.
  + Hash de cada parte (para conseguir integridad)
  + Tamaño de la parte
* Tamaño parte (1MB)

Luego, el peer envía un mensaje al tracker conocido donde requiere el socket de escucha del tracker primario. Crea una conexión contra el tracker primario y envía el META-Archivo y un hash de este. Al recibir el mensaje, el tracker primario intenta almacenar el META-Archivo en un directorio común para estos archivos utilizando como nombre el hash recibido. De existir otro archivo con ese nombre, se agrega al final del hash un número, partiendo de 0, que irá creciendo hasta obtener un nombre único. Finalmente, genera una nueva tupla en su tabla de archivos compartidos (file\_table) indicando:

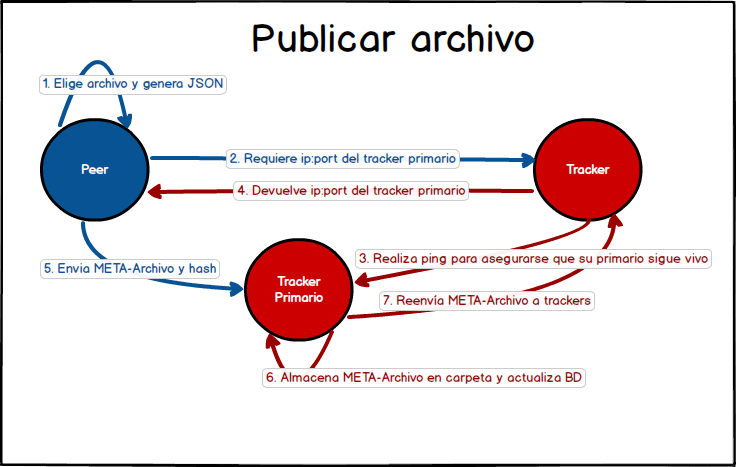
* Nombre del archivo concreto a compartir
* Tamaño
* ID JSON (hash)

El propósito de esta tabla es el de mejorar el rendimiento cuando un peer consulta por los archivos disponibles a un tracker.

Además, se crea una segunda tupla en otra tabla (seed\_table) con la siguiente información:

* ID JSON (hash)
* IP:PORT del peer que ofrece archivo
* Boolean isSeed = true (posee el archivo completo)
* Boolean disponible = true (Siendo false al superar el número de conexiones máximas del peer)
* Path del archivo en el peer

Esta tabla es consultada por el tracker cuando un peer pregunta por los peers que poseen un archivo (total o parcialmente) que desea descargar.

****

**Imagen 2**

Una vez almacenado el META-Archivo en la carpeta común de JSONs y creadas las filas en la base de datos,

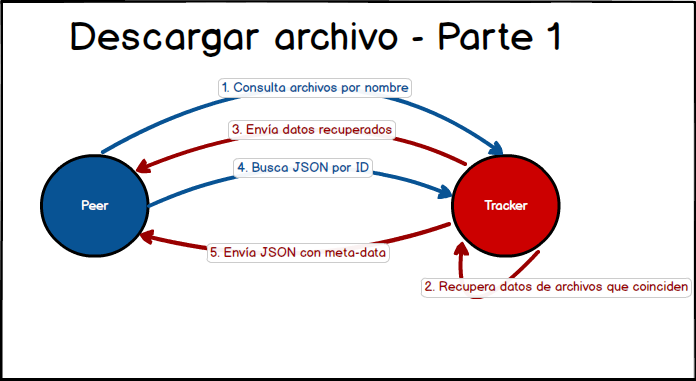
el tracker primario reenvía el META-Archivo a todos los trackers conocidos.

**Descargar archivo**

**Parte 1 – Descargar meta-data y estructura del archivo**

Para descargar un META-Archivo (**Imagen 3**), el peer escribe el nombre del archivo buscado y lo envía a su tracker. El tracker consulta la tabla file\_table de su base de datos por todas las tuplas cuyo nombre contenga el nombre buscado, y devuelve una lista con el nombre, tamaño, ID del JSON, cantidad de sedes y cantidad de leechers de todas las coincidencias encontradas.

El peer elige uno de los archivos y envía al tracker el ID del JSON solicitado. Finalmente, el tracker recupera el JSON y lo entrega al peer.



**Imagen 3**

Con el JSON, el peer puede decidir en cualquier momento descargar el archivo concreto al que está vinculado. Durante la descarga debe asegurarse que cada parte del archivo obtenida genera el mismo hash que aquel indicado en el JSON, asegurando la integridad de la parte descargada.

**Parte 2 – Descargar archivo**

El peer elige un archivo JSON, correspondiente al archivo que desea descargar, y un path donde almacenará el archivo. Leyendo el JSON elegido, la aplicación genera un nuevo JSON que indica qué partes faltan descargar, en principio todas, y será utilizado durante la descarga.

Este archivo está formado por un array de partes, con los siguientes datos:

* Hash parte
* Size parte
* Número parte
* Estado (Pendiente/Descargada)

Este archivo permite reanudar una descarga, ya que conocemos qué partes específicas faltan descargar. Se guarda en el path indicado para almacenar el archivo concreto a descargar.

Además, se crea otro JSON también necesario para la pausa y reanudación de descargas. Este JSON, almacenado en una carpeta llamada “Descargas pendientes” con los demás JSONs de archivos aún no completamente descargados, dispone de los siguientes datos:

* Path donde se almacenan las partes descargadas y el JSON de partes (mencionado anteriormente).
* Hash del archivo.
* Nombre del archivo.
* IP:PORT del peer (para que, al consultar el swarm, no sea ofrecido el mismo peer).
* Cantidad de partes del archivo original.

Al iniciar la aplicación, con los archivos existentes en la carpeta “Descargas pendientes” se forma una lista que permite al usuario conocer cuáles son aquellos archivos que aún no han terminado de descargarse. Además, permite conocer su estado (Pausado/Descargando) y cambiar el mismo.

Cuando se inicia la descarga de un archivo, también se agregan los datos de este nuevo archivo a la lista de descargas pendientes, con estado activo.

Una vez creados los JSONs mencionados, el peer envía al tracker el hash del archivo, para obtener un swarm del mismo, y toda la información necesaria para ser registrado como leecher. El tracker añade una nueva tupla a la tabla seed\_table:

* ID JSON (hash)
* IP:PORT del peer
* Boolean isSeed = false
* Boolean Disponible = true
* Path del archivo en el peer

El tracker obtiene todas las tuplas de la tabla seed\_table que tengan el ID JSON enviado por el peer y estén disponibles (sin carga máxima). De esta lista, elige 25 peers al azar. Si no se eligió un seed entre esos 25, se busca específicamente un seed, asegurando entregar un swarm que disponga del 100% del archivo.

En caso de no conseguir un seed, o no llegar a 25 peers, el swarm se envía igualmente.

La lista final corresponde al swarm, a partir del cual el peer obtendrá el archivo deseado descargándolo de múltiples peers.

Dicho swarm es obtenido por medio de un Thread de la clase ThreadCliente, encargado de llevar a cabo la descarga del archivo. Se crea un ThreadCliente por cada archivo a descargar.

La clase ThreadCliente comienza leyendo el archivo JSON de partes y carga una lista con las partes pendientes a descargar. Luego, inicia 1 Thread de la clase ThreadPartesDisponibles para obtener un swarm e indicar al usuario el porcentaje que dispone el swarm del archivo a descargar.

Una vez obtenido el swarm y conociendo que partes faltan descargar, el ThreadCliente intenta conectarse a peers pertenecientes al swarm. Pregunta por la lista de partes disponibles del peer al que se conectó (para saber que partes puede pedirles). Si falla la comunicación, se retira al peer de la lista de peers disponibles y se consulta a otro. Caso contrario, también se retira al peer de la lista de peers disponibles (para evitar que todos los ThreadLeecher consulten un mismo peer, aprovechando el swarm). El peer con que se establece conexión sube el número de conexiones entrantes.

Por cada peer al que logra conectarse (máximo 3), inicia 1 nuevo Thread de la clase ThreadLeecher, los cuales descargarán las partes pendientes que posee el peer.

Por cada parte que necesite nuestro peer y el peer seed posea, será solicitada al mismo y descargada. Al terminar la descarga de la parte, se retira esta de la lista de partes pendientes y se actualiza el JSON de partes pendientes.

Cuando el peer no tenga más partes para ofrecernos, se envía un mensaje al peer indicando el cierre de la conexión, bajando así su número de conexiones entrantes, y termina la ejecución del ThreadLeecher. De haber más partes faltantes, el ThreadCliente busca un nuevo peer en la lista de peers disponibles e inicia un nuevo ThreadLeecher. Si la lista de peers disponibles se vacía, esta se renueva con el swarm antes pedido. Esto permite que, con la constante renovación de la lista de peers disponibles, los ThreadLeecher se puedan conectar a un mismo peer al mismo tiempo (siendo este peer potencialmente uno con mayor cantidad de partes), logrando no perder tanta velocidad de descarga al llegar al final de esta cuando falten menos partes.

El ThreadPartesDisponibles puede volver a ser llamado en cualquier momento de la descarga por el ThreadCliente en caso de que el swarm de peers disponibles se vacíe. Un peer se retira del swarm cuando: No posee partes que nos falten descargar, falló la conexión con él, ya no posee el archivo a descargar.

Una vez descargadas todas las partes, finalizan todos los ThreadLeecher y se reconstruye el archivo. Además, se notifica al tracker que el peer es ahora un seed del archivo (posee el 100% de las partes).

Imagen que contiene mapa

Descripción generada automáticamente

**Imagen 4**

Si el peer consultado responde que no existe el path pedido, ya que elimino el archivo o lo cambio de directorio, deberá avisar a su tracker que ya no posee el archivo en cuestión, siendo eliminado de la tabla seed\_table (no perteneciendo más al swarm).

Si el peer consultado responde con su archivo donde informa de qué partes dispone, el número de conexiones aumenta y solo vuelve a bajar cuando el peer cliente haya descargado todas las partes que el peer servidor ofrece y el cliente no posea. De esta forma, cuando otro peer consulte por su archivo de partes de un archivo dado, si tiene carga máxima de conexiones responderá con un mensaje que indica dicho estado.

La comunicación entre peers que hacen uso de NAT se logra mediante UPnP automatic port forwarding. Esto permite que los peers reciban conexiones desde el exterior.

Imagen que contiene texto, captura de pantalla, mapa

Descripción generada automáticamente

**Imagen 5**

**Iniciación Trackers**

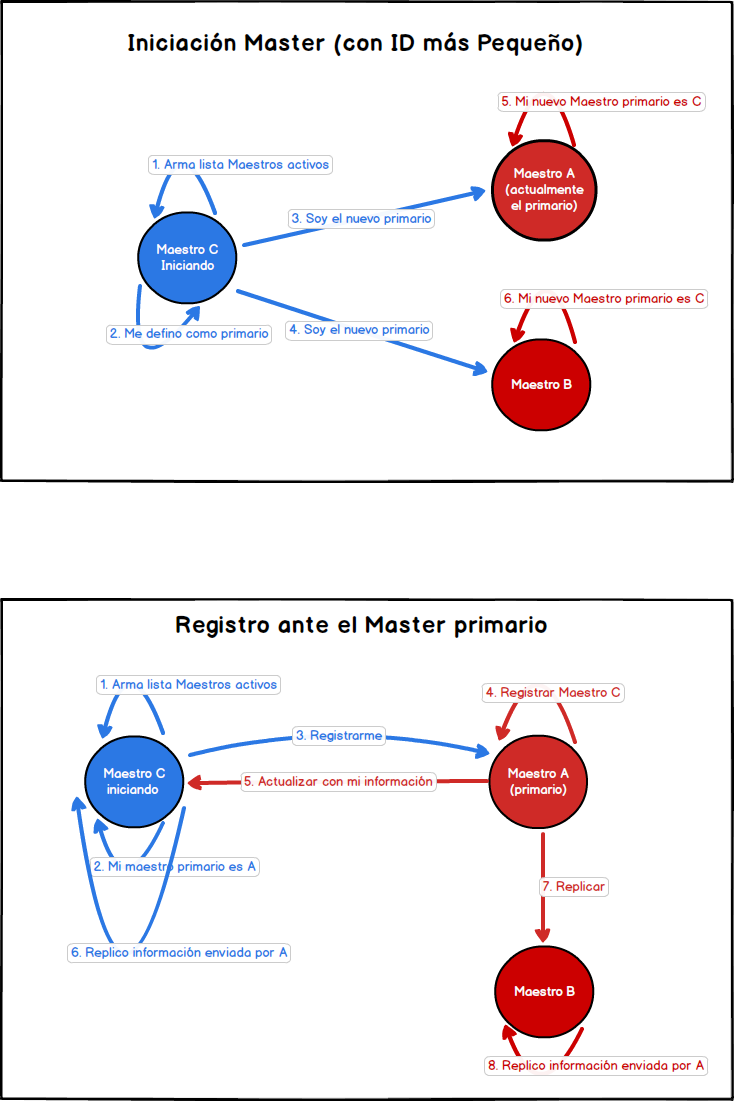
Los maestros (trakers) al momento de iniciarse deben definir quién es su Maestro primario. La política de selección de Maestro primario es elegir siempre el Maestro de menor ID. Es decir, en todo momento el Maestro primario debe ser aquel maestro que esté activo y además posea el menor ID.

Para la selección del Maestro primario, el Maestro que está iniciando recorre un archivo propio el cual posee una lista de todos los Maestros. En base a ella genera una lista de los Maestros activos (aquellos que confirman estar vivos) en la cual él también se incluye.

Luego de construir la lista de Maestros activos pueden darse los siguientes casos:

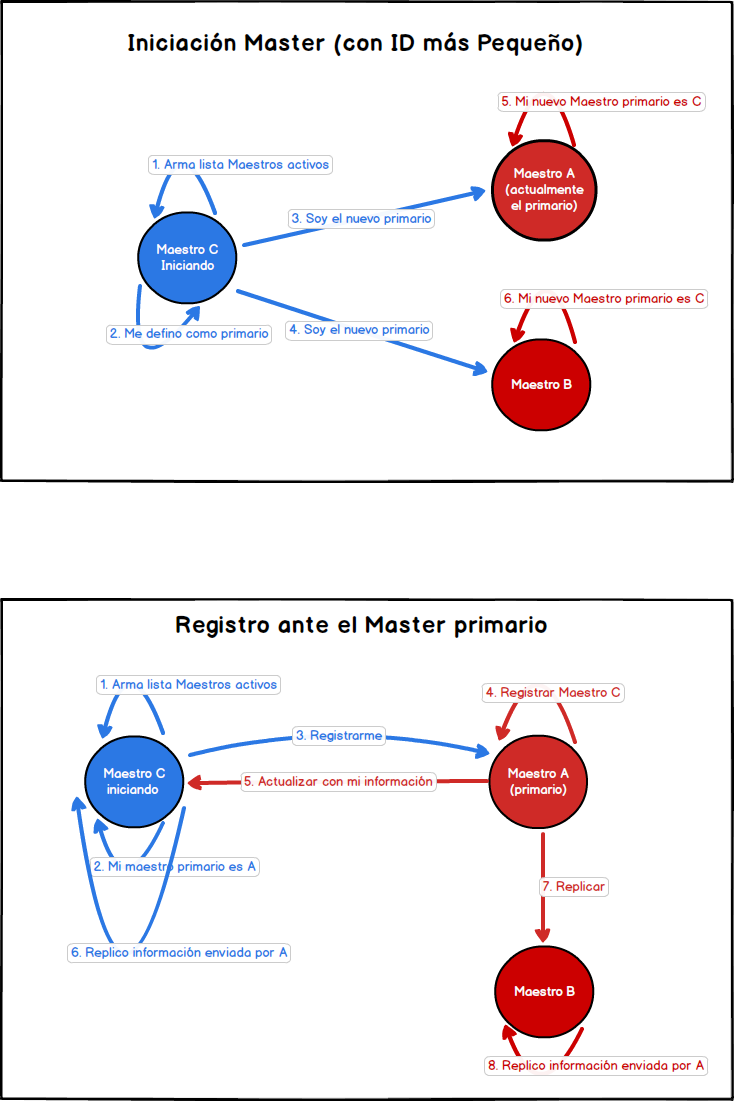
1. De ser el único Maestro en la lista, se autodefine como primario.
2. De haber al menos dos Maestros en la lista (él y uno más), se procede a elegir el Maestro con ID más pequeño:

* Si él es el Maestro con ID más pequeño, se autodefine como Maestro Primario, y avisa al resto de los Maestros que es el nuevo Maestro primario (**Imagen 6**).
* Si no posee el ID más pequeño, define al Maestro con ID más pequeño como su Maestro primario (**Imagen 7**).



**Imagen 6**

Al finalizar la selección del Maestro primario, el Maestro iniciado debe registrarse (**Imagen 7**) ante el Maestro primario (excepto que fuese él mismo). El Maestro primario al recibir un mensaje de registro, realizará una réplica sobre el Maestro que solicitó el registro, y comunicará a los demás Maestros de la existencia de este nuevo Maestro (replicación).

****

**Imagen 7**

**Replicación**

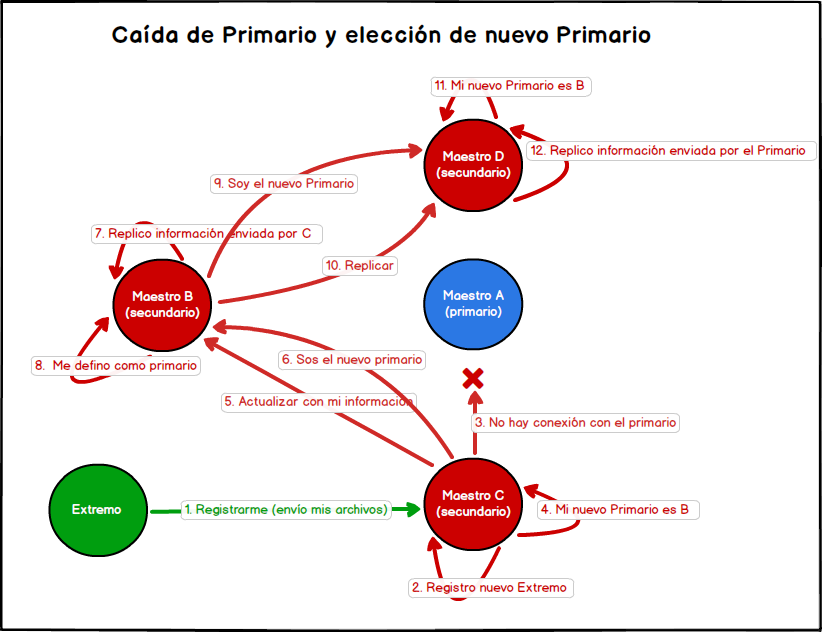
El único Maestro que está habilitado a realizar las replicaciones es el Maestro Primario. Es por ello que los Maestros secundarios ante cualquier actualización que sufran deben informar al Maestro primario para que este se actualice y además realice la réplica sobre los demás Maestros activos (la réplica no se realiza sobre el Maestro que disparó la actualización).

Las replicaciones se llevan a cabo en las siguientes situaciones:

1. Cuando un Peer envía un archivo JSON (**Imagen 2**).
2. Cuando un Peer finaliza una descarga y avisa al Maestro que posee un nuevo archivo completo (**Imagen 4, paso 8**).
3. Cuando un Peer ya no posee un archivo dado y debe ser retirado del swarm.
4. Cuando un Peer es habilitado o deshabilitado por el número de conexión máximas.
5. Cuando un Maestro se acaba de iniciar y se registra ante el Maestro Primario (**Imagen 7**).

Siempre que los Maestros secundarios realizan una operación y necesitan que su información sea replicada hacen el pedido de replicación al Maestro Primario. Si llegaran a detectar que el Maestro Primario no está activo (se cayó por algún motivo), deberán comenzar un proceso para definir el nuevo Maestro Primario. En este proceso, el Maestro secundario revisa en su lista de Maestros activos quién posee el ID más pequeño:

* En caso de que él tenga el ID más pequeño, avisa a los demás Maestros activos que él es el nuevo Maestro primario. Y luego realiza la replicación.
* En caso de que no tenga el ID más pequeño, le replica al Maestro con ID más pequeño su información (actualización reciente), y además le indica que es el nuevo Maestro primario. El nuevo Maestro primario al recibir estos mensajes, se autodefine como maestro primario, se actualiza, informa que es el nuevo Maestro primario y por último realiza la replicación hacia los demás Maestros activos (**Imagen 8**).



**Imagen 8**

**Problemas que resuelve**

La ventaja de este servicio aparece al compararla con el tipo de arquitectura cliente-servidor, donde un único servidor (centralizado) se encarga de atender todas las solicitudes de los clientes y ofrecer el recurso solicitado.

Cada cliente descarga una copia completa de un archivo del servidor. Si una gran cantidad de clientes intenta descargar al mismo tiempo, el servidor debe dividir el bandwidth entre los distintos clientes, bajando la velocidad de descarga proporcionalmente a la cantidad de clientes.

Por el contrario, el sistema propuesto solo hace uso de un servidor (tracker) como fuente de meta-datos (índice) para encontrar un archivo requerido por un peer. La descarga del archivo en concreto es llevada a cabo por medio de conexiones contra N peers que poseen distintas partes del archivo, aprovechando la velocidad de subida de cada nodo individual.

A medida que un nodo comienza a descargar partes de un archivo desde el nodo origen, puede ser aprovechado para que otros descarguen de él, de esta forma no se exprime únicamente la velocidad de subida de un único servidor.

Otra ventaja es la reducida cantidad de mensajes intercambiados entre peers durante la descarga, y el uso de una única conexión para la descarga de múltiples partes de un mismo peer servidor.

Respecto a la cantidad de mensajes, un peer no debe avisar a todos los peers de su swarm cuando ha descargado una parte exitosamente, ni recibir este mismo mensaje del resto de peers. Al contrario, el peer cliente pregunta por las partes descargadas en el peer servidor solo después de establecer una conexión.

Luego, esta conexión se mantiene hasta que el peer cliente haya descargado todas las partes que este peer servidor ofrece y el peer cliente no posea.

De esta manera, se consigue:

* Reducir el overhead del protocolo, minimizando la cantidad de mensajes intercambiados (ya sea por actualización de partes disponibles o handshakings por cada conexión).
* Aprovechar la máxima velocidad de transmisión entre los peers durante la transmisión (evitando un nuevo slow start con cada nueva conexión).

**Base de Datos**

Se implemento una base de datos orientada a objetos, conocida como DB4O. Su uso se debe a su sencilla implementación, evitando la creación de la base de datos y sus tablas por medio de algún software externo, simplificando así el uso de la aplicación en cualquier equipo que se desee correrla.

Para importar la librería en eclipse debe hacer click derecho en la carpeta raíz del proyecto, seleccionar “Build Path”, “Configure Build Path…”, “Add External JARs…” y elegir el jar llamado “db4o-8.0.249.16098-all-java5.jar” ubicado en la carpeta principal del proyecto.

**Seguridad**

Al crear el JSON, con la meta-data del archivo a compartir, se especifica una lista con el hash de cada parte que forma el archivo. Luego, al descargar una parte, se calcula su hash y se lo compara con el hash del JSON. De esta forma se garantiza la integridad de cada pieza descargada. El hash es calculado con la función SHA-1.

También se considera la privacidad de los datos intercambiados, encriptándolos antes de cada envío. La encripción previene que nodos intermedios sepan qué información se está compartiendo entre los peers y trackers. Esto se logra mediante el sistema criptográfico RSA, que permite la generación de claves asimétricas. Cada peer y tracker dispondrá de un par de claves (pública y privada), generadas cuando se inician.

Quien inicie la comunicación, comienza enviando un mensaje CHECK\_AVAILABLE que incluye la clave pública del emisor. En caso de que el receptor este activo, genera una clave simétrica utilizando el algoritmo de cifrado simétrico (Blowfish) que crea una llave de 64 bits.

Esta clave simétrica es luego encriptada con la clave pública del emisor y será utilizada durante el resto de la conversación. Puede considerarse una llave pequeña, de baja seguridad, pero es apropiada para el sistema que busca velocidad por sobre seguridad.

**Bibliografía**

**Tanenbaum: Distributed Systems -** <https://komputasi.files.wordpress.com/2018/03/mvsteen-distributed-systems-3rd-preliminary-version-3-01pre-2017-170215.pdf>

**Comparison of Peer-to-Peer Overlay Network Schemes -** <http://snap.stanford.edu/class/cs224w-readings/lua04p2p.pdf>

**The Role and uses of Peer-to-Peer in file-sharing -**<https://pdfs.semanticscholar.org/6126/e3cc8512dced8cd2ee63b4830285b3cfa9ff.pdf>

**Wikipedia: BitTorrent -** <https://en.wikipedia.org/wiki/BitTorrent>