Sistema para la compartición de archivos

Zabala Mariano
Mariano.zabala@hotmail.com

Universidad Nacional de Luján
Lic. en Sistemas de Información
Sistemas distribuidos y Programación paralela (41409)

Abstract

Este documento presenta un método para conseguir la compartición de archivos entre distintos equipos, a través de la red, mediante un sistema distribuido que divide la carga de trabajo entre los nodos participantes, y siendo resistente a fallos.

Características generales

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.

El sistema está conformado por los siguientes 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. Además, facilita al peer la consulta por seeds, dado que 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 pública y privada 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.

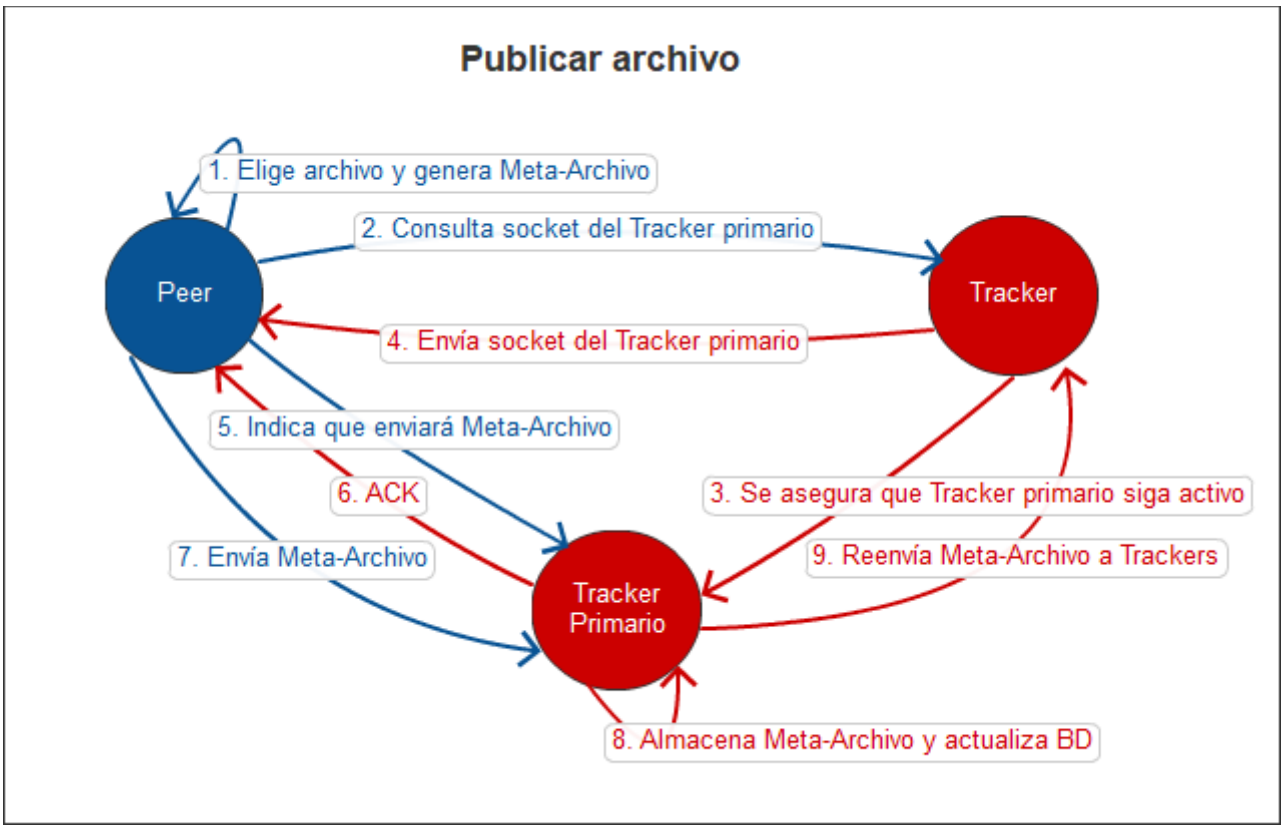


Imagen 2

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 seeds 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.

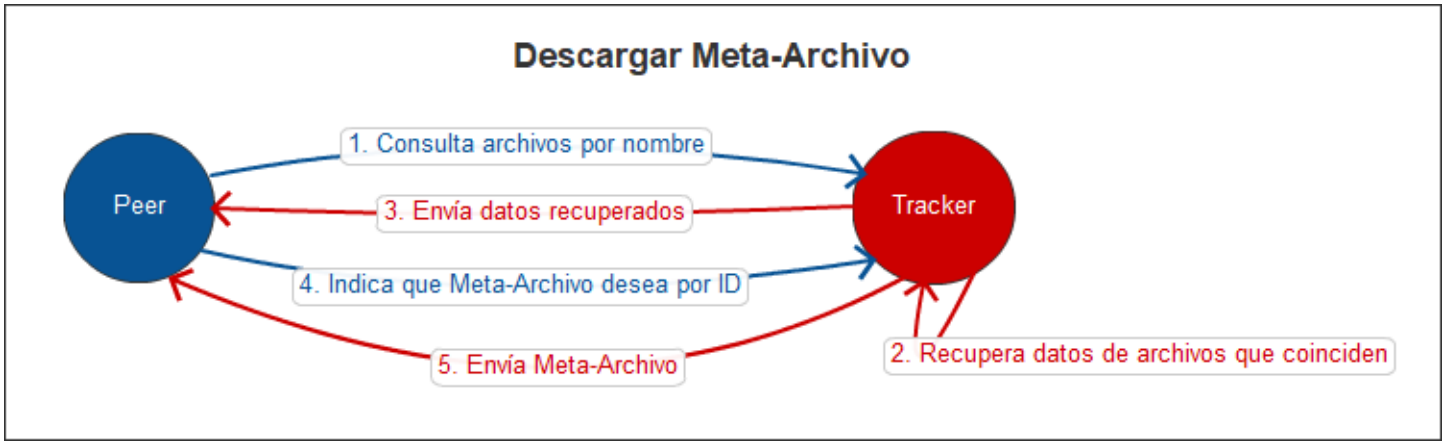


Imagen 3

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 pública y privada 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 pública y privada 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). La lista obtenida corresponde al swarm, a partir del cual el peer obtendrá el archivo deseado descargándolo de múltiples peers.

Una vez obtenido el swarm, se crea un “ThreadCliente” que se encargará de la descarga del archivo.
La clase “ThreadCliente” comienza leyendo el archivo JSON de partes y carga una lista con las partes pendientes a descargar. Luego, pregunta por medio de un objeto de la clase “PartesDisponibles” el porcentaje del 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 objeto "PartesDisponibles" 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, pidiendo un nuevo swarm a un tracker disponible. 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).

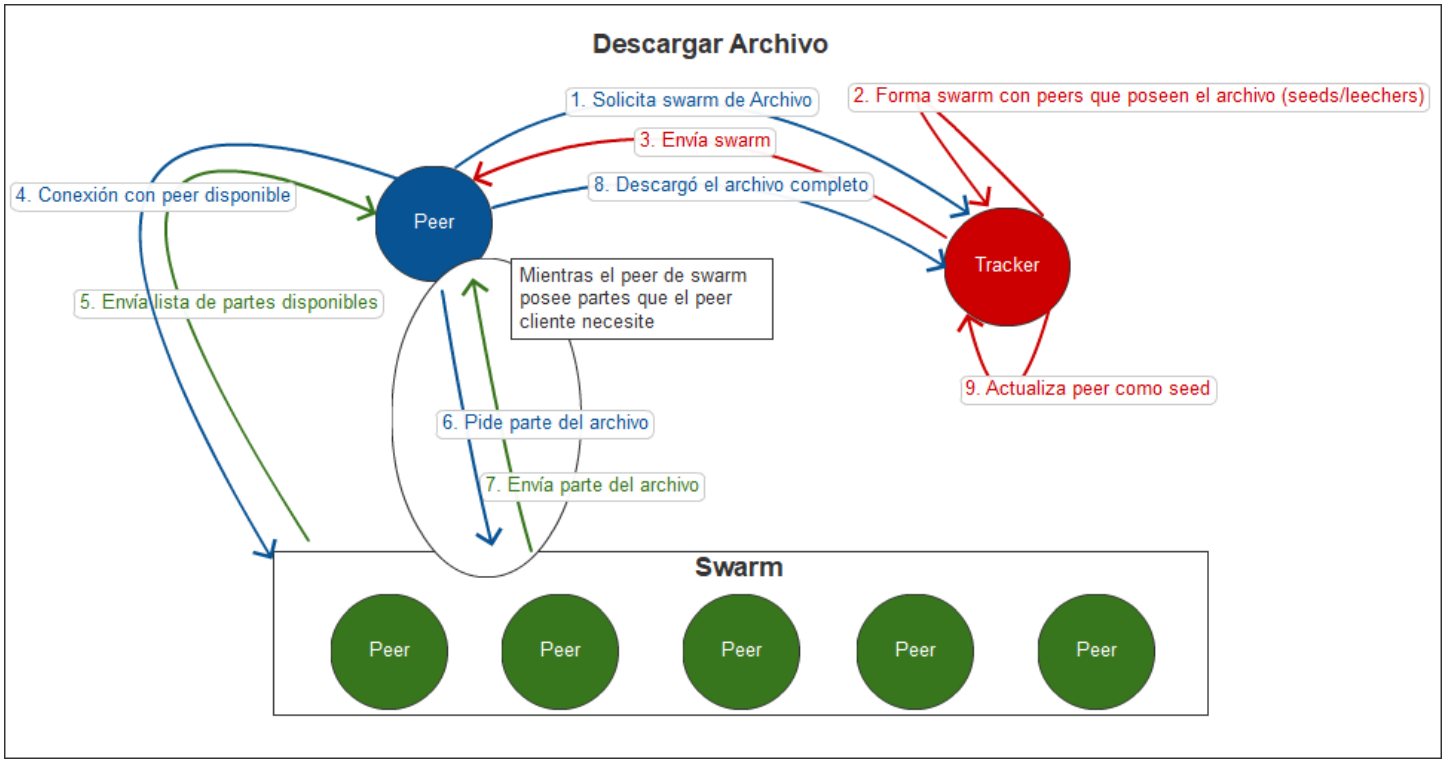


Imagen 4

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.

NAT traversal y Stateful Firewalls

Para que la comunicación entre peers sea posible, se deben superar dos obstáculos: (1) **Network Address Translators** (NAT) y (2) **Stateful Firewalls**. (1) NAT altera los paquetes que pasan por el router de la red local, cambiando su dirección IP y, potencialmente, su puerto. Esto supone un problema dado que los Peers deben compartir entre sí la dirección utilizada por sus routers, y no la de ellos, para saber dónde enviar los mensajes, con la condición añadida de que estos routers deben saber a qué equipo de su red interna enviar los paquetes entrantes.

Este problema fue solucionado, en primera instancia, mediante el uso de **UPnP automatic port forwarding**. Este protocolo permite que los peers reciban conexiones desde el exterior gracias a que se define una regla la cual indica al router redirigir todo tráfico que llegue a un puerto específico ‘Y’ a un equipo (IP privada y puerto ‘X’) ubicado en la LAN del router. En caso de no contar con esta funcionalidad, era necesario realizar port forwarding de forma manual, indicando en su router a qué IP y puerto redirigir el tráfico que llegue a ‘Y’ puerto del router.

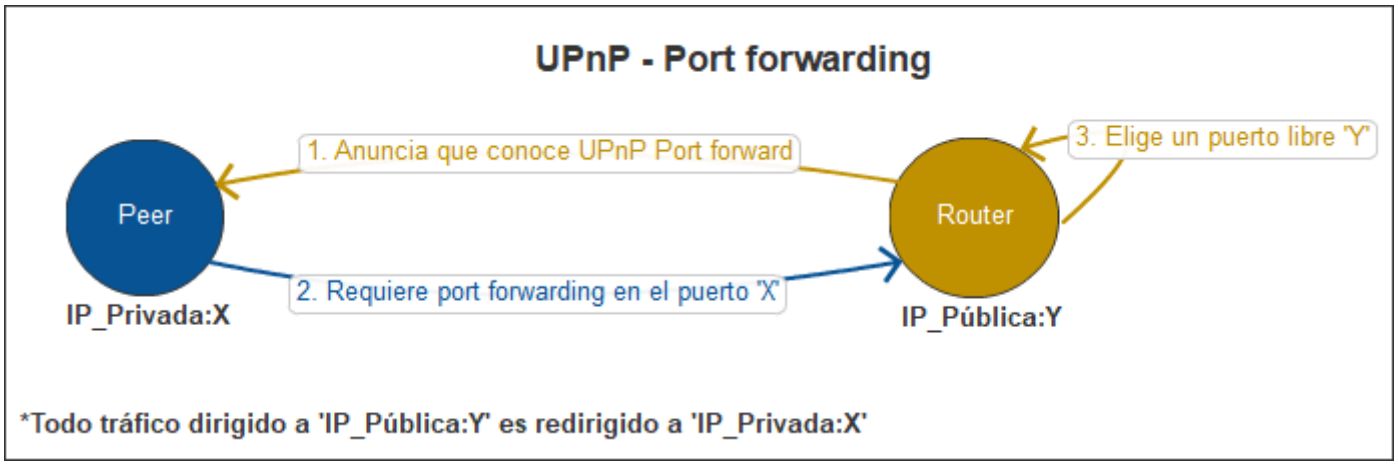


Imagen 5

Sin embargo, al ejecutar pruebas en redes públicas, se evidenció que UPnP (y port forwarding manual) no era suficiente para asegurar la conexión entre peers, debido a las características propias de cada firewall.
(2) Un firewall stateful se caracteriza por recordar qué paquetes "vió" en el pasado y puede utilizar este conocimiento para decidir qué hacer con nuevos paquetes que aparezcan en el futuro. Específicamente, la configuración más común es la de permitir el paso de paquetes salientes y el de desechar paquetes entrantes a menos que el firewall los identifique como pertenecientes a una conexión existente iniciada dentro de la red privada. De esta manera, nadie puede comunicarse con el Peer a menos que este se comunique previamente con el destino. En el escenario donde ambos Peers se encuentran detrás de un firewall de este estilo, la comunicación se vuelve imposible ya que sus respectivos firewalls "desecharían" los paquetes entrantes de direcciones (IP:puerto) desconocidas.

De esta manera, para sobrellevar estos problemas (NAT y stateful firewall) se propusieron nuevas soluciones, donde se incluye STUN (se encarga de solventar NAT) y Hole punching (se encarga de solventar stateful firewall).

El protocolo **STUN** soluciona el problema presentado por NAT ya que permite a los Peers preguntar a un Tracker cuál es su dirección pública (la designada por el router al realizar NAT), pudiendo así compartirla con el resto de peers.

Luego, Stateful firewall se enfrentó mediante la aplicación de **hole punching**. Primero, es necesario que los peers que desean comunicarse conozcan la dirección (IP:puerto) que su contraparte utilizará para establecer la conexión (resuelto por medio de STUN). Esta información permite que los peers establezcan una conexión (incluso para peers detrás de stateful firewalls) ya que el Peer A puede enviar un mensaje a la dirección específica de Peer B, y si el Peer B ya envió un mensaje a la dirección del Peer A, el router de Peer B dejará pasar el mensaje de A, pudiendo establecer una conexión. Si B no envió su mensaje, su router descartará el mensaje de A. Pero al enviar su mensaje, el router de A lo considerará como una respuesta que será aceptada.
En la **Imagen 6** vemos que el Maestro primario se encarga de compartir los EndPoints de los Peers entre ellos. Luego, si los peers comparten la misma IP pública, se conectan por medio de sus direcciones privadas. Caso contrario, se conectan utilizando sus direcciones públicas.

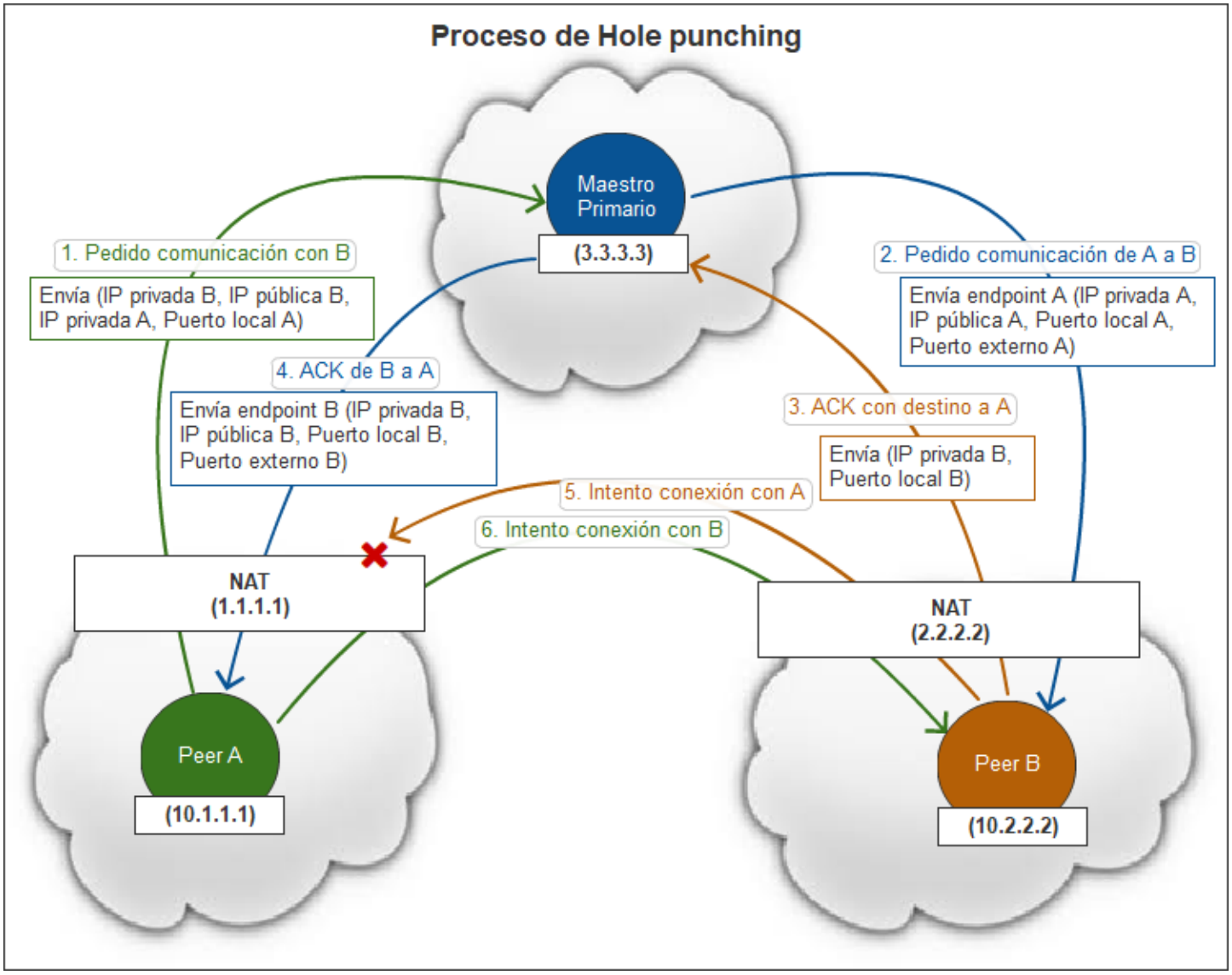


Imagen 6

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 7**).
   - Si no posee el ID más pequeño, define al Maestro con ID más pequeño como su Maestro primario (**Imagen 8**).

Imagen 7

Al finalizar la selección del Maestro primario, el Maestro iniciado debe registrarse (**Imagen 8**) 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 8

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 comienza la descarga de un archivo, convirtiéndose en "leecher".
3) Cuando un Peer finaliza una descarga y avisa al Maestro que posee un nuevo archivo completo, convirtiéndose en "seed" (**Imagen 4, paso 8**).
4) Cuando un Peer ya no posee un archivo dado y debe ser retirado del swarm.
5) Cuando un Peer es habilitado o deshabilitado por el número de conexión máximas.
6) Cuando un Maestro se acaba de iniciar y se registra ante el Maestro Primario (**Imagen 8**).

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 9**).

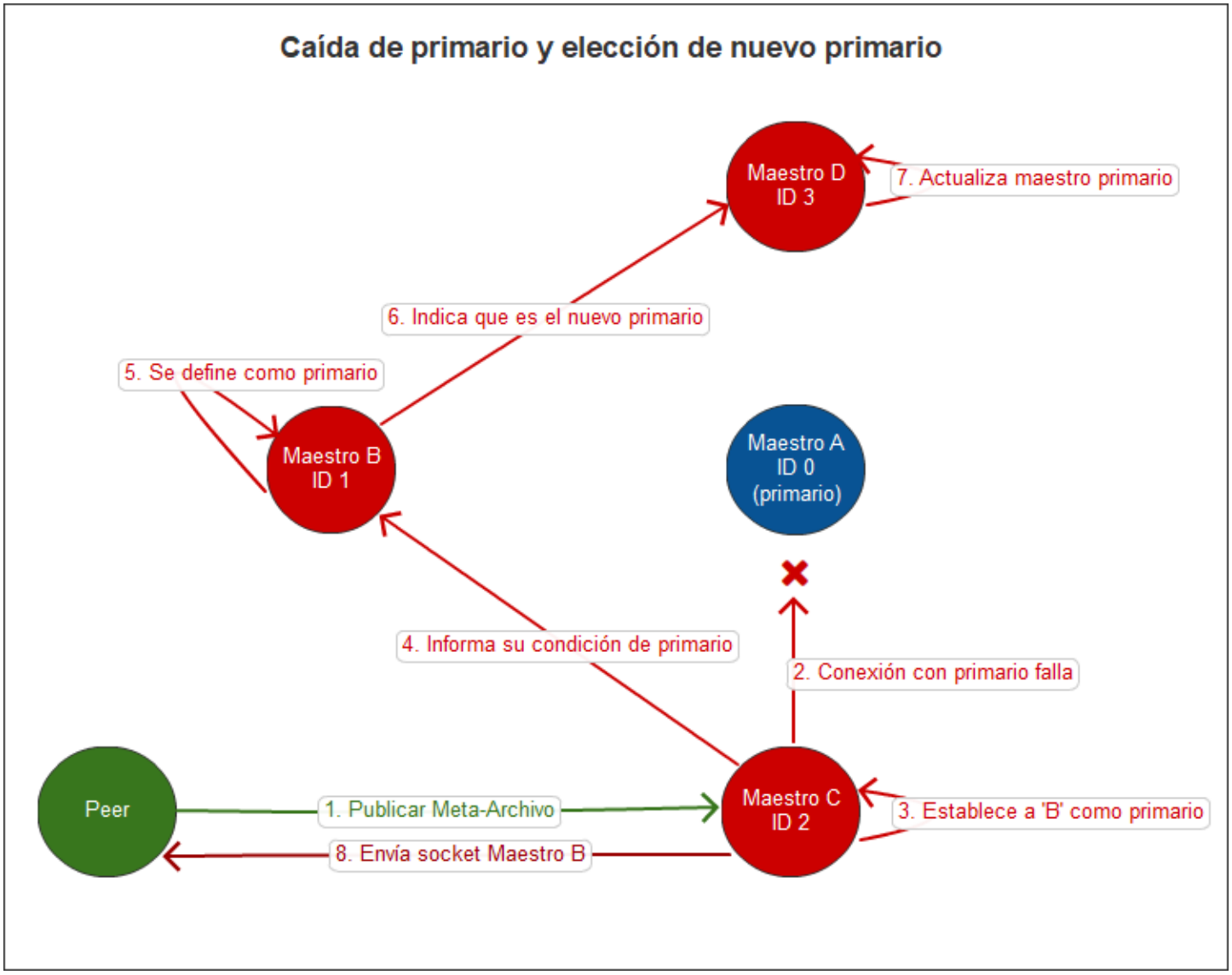


Imagen 9

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 e Integridad de los datos

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.

Instalar JDK

Asegúrese de instalar JDK antes de ejecutar la demo de descarga.

```
C:\Users\User>java -version
java version "12.0.2" 2019-07-16
Java(TM) SE Runtime Environment (build 12.0.2+10)
Java HotSpot(TM) 64-Bit Server VM (build 12.0.2+10, mixed mode, sharing)
```

Demo descarga

Iniciación tracker:

- Windows: doble-click “tracker.bat”.
- Linux: abrir consola, ejecutar “chmod +x ./tracker.sh”, seguido por “./tracker.sh”.

Luego, especificamos la carpeta donde se almacenarán los Meta-Archivos enviados por peers:
Path/to/tracker/jsons tracker

Finalmente, elegimos uno de los IDs existentes en el archivo “trackers.json”, “0” en este caso.

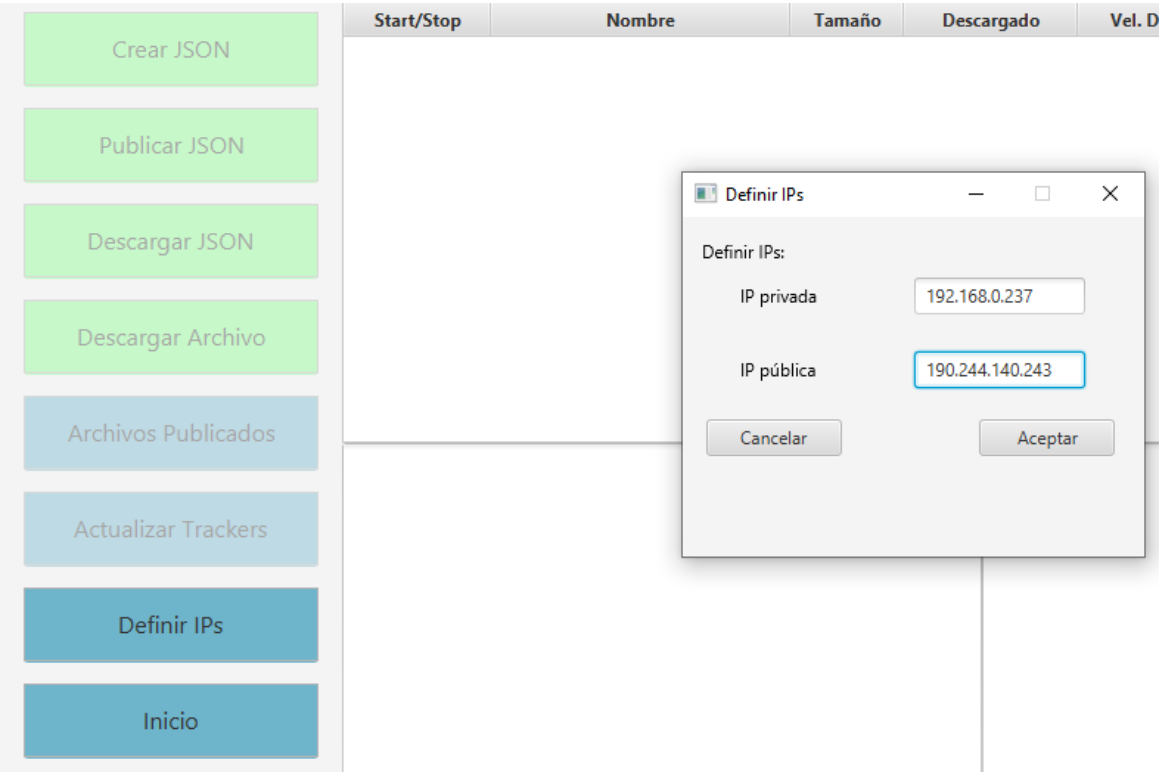
```
Ingrese el path donde se guardaran los META-Archivos publicados por peers: C:\Users\User\Desktop\Tracker\jsons tracker

-------------INICIAR TRACKER-------------
Ingrese el Id del tracker: 0
2021-09-06T12:09:29.132-0300 INFO Tracker 0 iniciado
2021-09-06T12:09:29.134-0300 INFO Path donde se guardaran los META-Archivos: C:\Users\User\Desktop\Tracker\jsons tracker

2021-09-06T12:09:29.149-0300 INFO id primario: 0, Mi id: 0
2021-09-06T12:09:29.150-0300 INFO Mi ID es: 0
Mi IP es: 190.244.140.243
Mi IP privada es: 192.168.0.237
Mi PUERTO es: 7000
Mi Tracker Primario es: 0, Pub (190.244.140.243:7000), Priv (192.168.0.237:7000)
2021-09-06T12:09:29.151-0300 INFO Mi lista de trackers es:
- 0, Pub (190.244.140.243:7000), Priv (192.168.0.237:7000)
-----------------------------------------------------
```

Iniciación peer:

- Windows: doble-click “peer.bat”.
- Linux: abrir consola, ejecutar “chmod +x ./peer.sh”, seguido por “./peer.sh”.

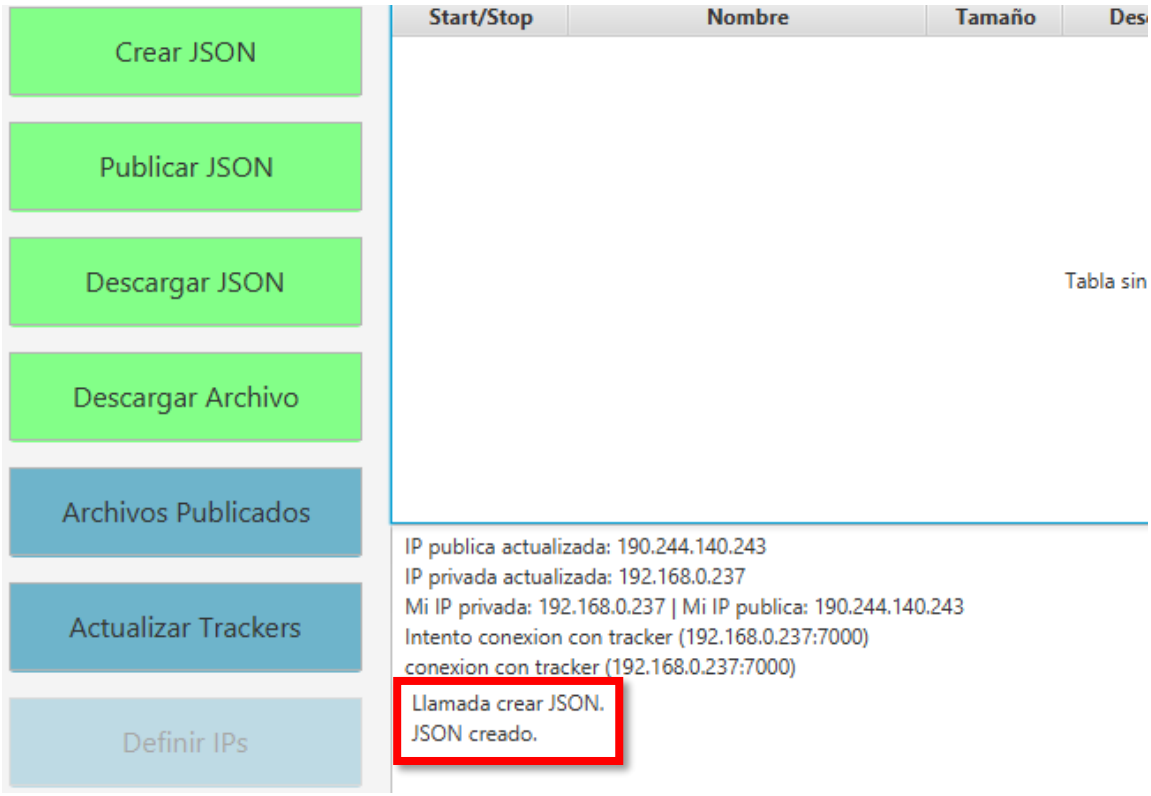
En caso de realizar una prueba local, con tracker y peer en la misma LAN, debe presionar “Definir IPs” donde se pide completar la IP privada e IP pública del peer. Es importante que la IP pública definida sea la misma vista en “trackers.json” para el tracker, de forma que el programa pueda determinar que están en la misma red y deben comunicarse por medio de sus IPs privadas. Luego, presione “Inicio”.

En caso de contar con un tracker en una red diferente a la del peer, presionar “Inicio”. El tracker se encarga de informar la IP pública del peer.



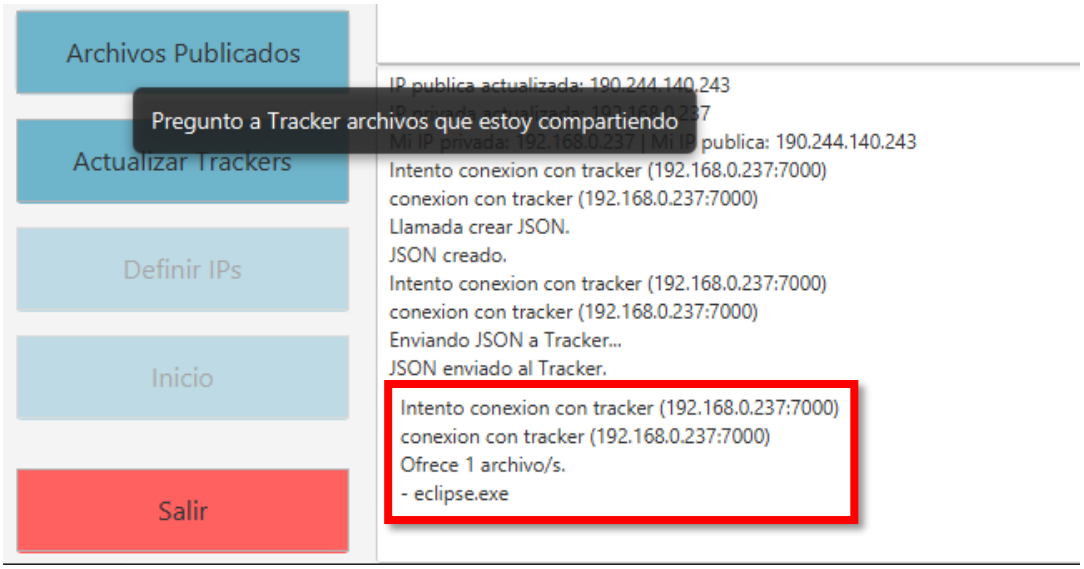
Creamos un Meta-Archivo por medio del botón “Crear JSON”.

Una vez creado, se indicará en el textarea del peer cliente. El textarea ubicado a la derecha corresponde al peer bajo el rol de servidor.



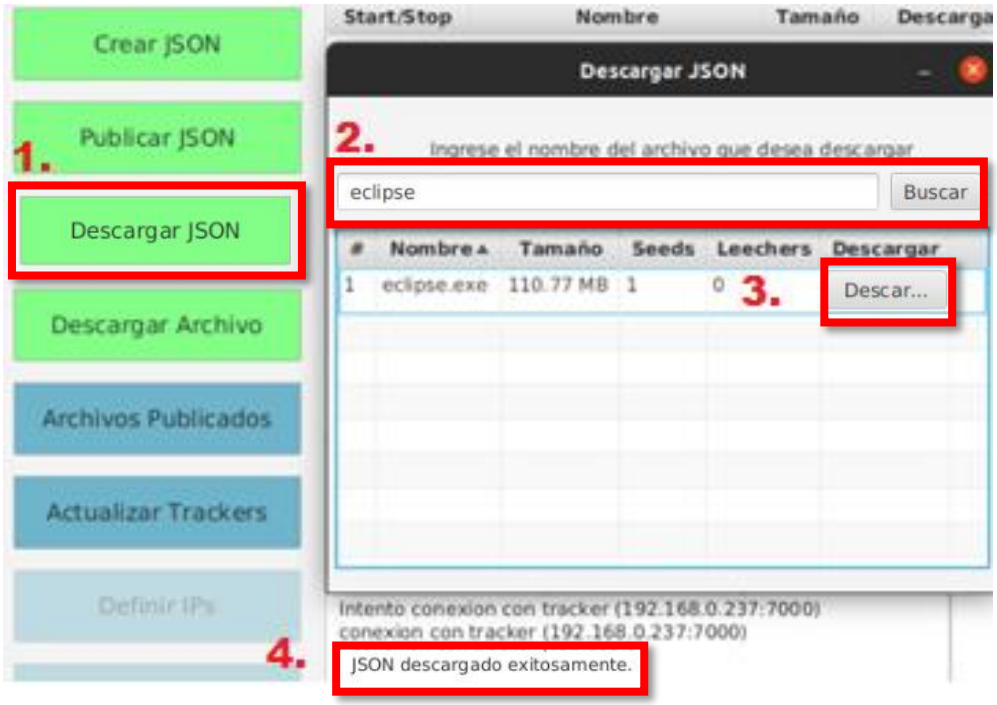
Publicamos el Meta-Archivo por medio del botón “Publicar JSON”. El Meta-Archivo creado se encuentra en la carpeta “JSONs”.

Confirmamos que el Meta-Archivo fue publicado correctamente mediante el botón “Archivos Publicados”.



Iniciamos un nuevo peer, en un equipo distinto (con otra IP privada), el cual llevará a cabo la descarga del archivo publicado por el primer peer.

Descargamos el Meta-Archivo publicado por el primer peer:
1. Presionar el botón “Descargar JSON”.
2. Buscar archivo por nombre y presionar “Buscar”.
3. Presionar “Descargar” sobre la fila del Meta-Archivo deseado.
4. Puede ver en el textarea del peer cliente cuando el Meta-Archivo se haya descargado.



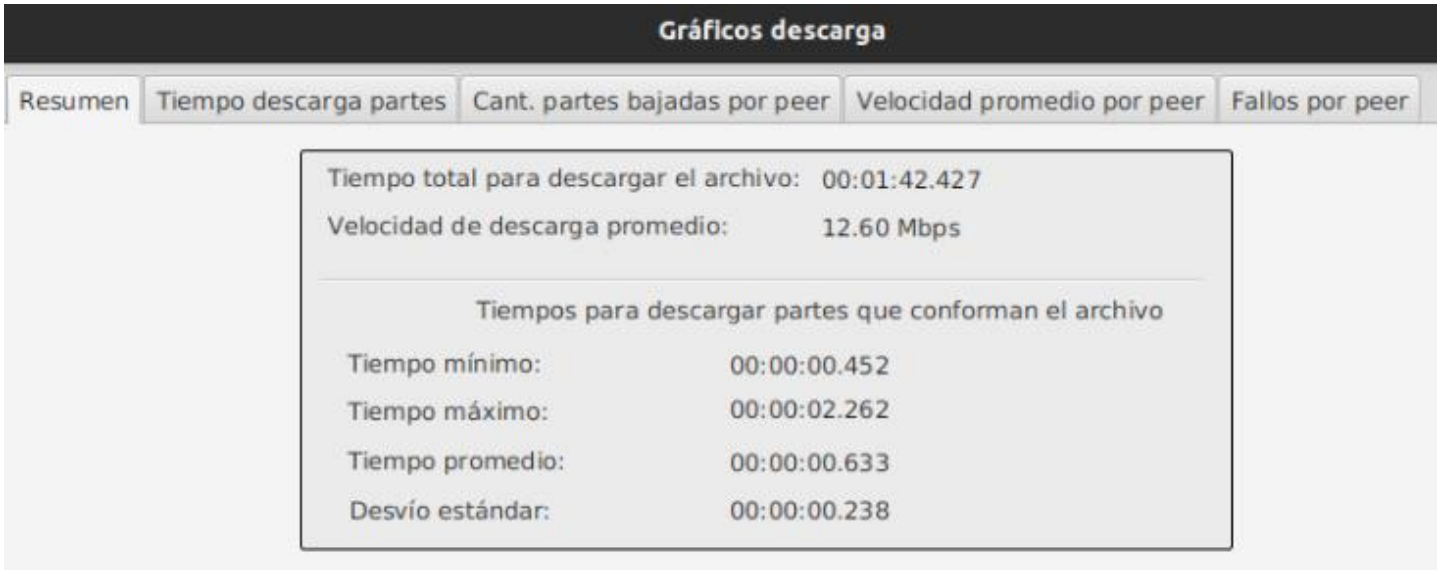
Finalmente, iniciamos la descarga del archivo mediante el botón “Descargar Archivo”.
1. Presionamos el botón “Descargar Archivo”. Primero debemos elegir el Meta-Archivo descargado en el paso anterior. Este se ubica en la carpeta “JSONs”.
Luego, debemos elegir la carpeta donde se guardará el archivo a descargar.
2. Podemos ver que se añade una fila en la tabla de descargas. En la primera columna “Start/Stop” vemos el botón “Stop”. Este botón permite detener y continuar la descarga en cualquier momento.

Al terminar la descarga desaparece el botón “Stop” o “Start” y aparece “Borrar” el cual borra la descarga de la tabla y sus archivos asociados en las carpetas “Graficos” y “Descargas pendientes”.
Además, aparece el botón “Ver” en la última columna llamada “Gráficos”.

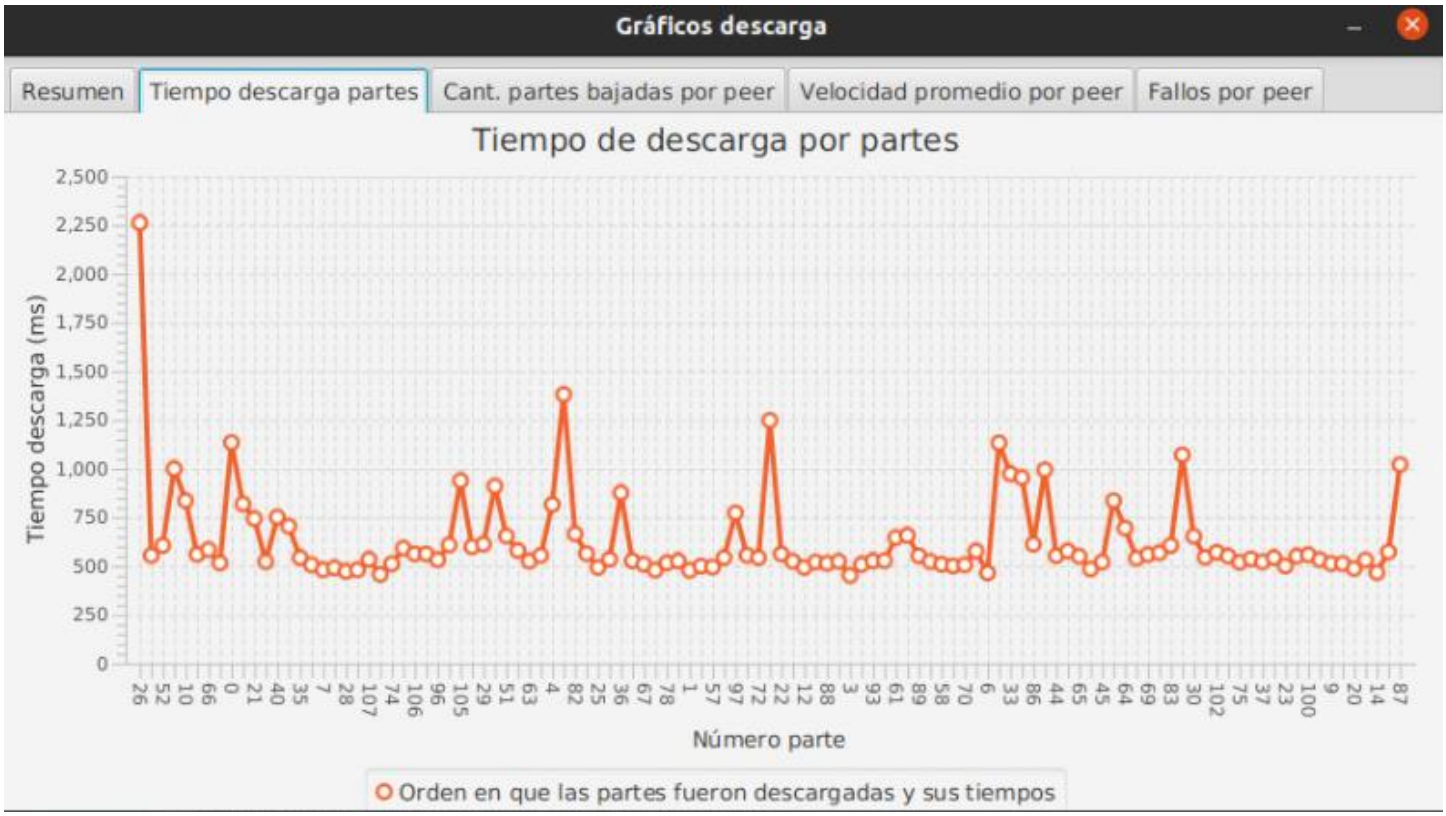
Start/Stop	Nombre	Tamaño	Descargado	Vel. Descarga	Disponible en swarm	Gráficos
Borrar	eclipse.exe	110.77 ...	100.00%	7.84 Mbps	100.00%	Ver

El botón “Ver” abrirá una nueva ventana donde podemos ver:

- Un resumen de la descarga que incluye los siguientes datos.



- El tiempo que llevó descargar cada parte del archivo, pudiendo ver la performance a lo largo de toda la descarga.



- Cantidad de partes descargadas de cada uno de los peers a los que me conecté durante la descarga. Dado que en el ejemplo existía un único peer servidor con el archivo, aparece una sola columna la cual descargó las 111 partes del archivo.

- Velocidad promedio a la que se descargaron las partes desde cada uno de los peers servidores. Dado que en el ejemplo existía un único peer servidor con el archivo, aparece una sola columna.

- Errores existentes según cada uno de los peers servidores a los que me conecte durante la descarga.

Cada peer tiene su propia columna y en cada una se stackean los tipos de errores y se indica el número de errores por tipo.

Logs

Puede ver los logs generados en la carpeta “Logs Demo TP-Final”. Estos contienen muchos más detalles que aquellos que puedan aparecer en los textarea de la aplicación.

- peerCliente.log (Información logeada por peer que llevó a cabo la descarga).
- peerServidor.log (Información logeada por peer que publicó el archivo).
- trackerNro0.log (Información logeada por tracker).

Posibles mejoras

- Ofrecer una forma de conexión híbrida entre peers. Es decir, permitir la comunicación mediante UPnP en los casos que sea posible, y no solo STUN + Hole punching. Esto es beneficioso para el tracker primario, al reducir el tráfico del mismo, además de ser más veloz al comunicar los peers de forma directa desde el inicio.
- Los archivos son divididos en partes iguales de 1MB. En su lugar, dividirlos en partes de un tamaño definido por el del archivo. Mientras mayor sea el tamaño del archivo, mayor será el de las partes en que es dividido. De esta forma se evita la creación de Meta-Archivos JSONs con tamaños demasiado grandes.
- Análisis de velocidad de descarga en base al tamaño de las partes en que son divididos los archivos y su relación con número de fallas/reenvíos al utilizar grandes tamaños.
- Cambiar la estrategia de descarga de partes, dejando de descargar partes random, tomando el enfoque “rarest-first” el cual busca que el swarm de leechers disponga de todas las partes del archivo, permitiendo la descarga completa del mismo incluso en caso de no haber seeds conectados.

Bibliografía

[1] Andrew S. Tanenbaum & Maarten van Steen (2007): Distributed systems: principles and paradigms, 2nd ed edition. Pearson Prentice Hall, Upper Saddle RIiver, NJ. OCLC: ocm70707891.
[2] The BitTorrent Protocol Specification v2. Disponible en bep_0052.rst_post (bittorrent.org).
[3] David Anderson: How NAT traversal works. Disponible en How NAT traversal works · Tailscale.
[4] Bryan Ford, Pyda Srisuresh & Dan Kegel: Peer-to-Peer Communication Across Network Address Translators. Disponible en – Bryan Ford's Home Page (bford.info).
[5] Jenny Bengtsson & Prarthanaa Khokar:Computer Communication & Distributed Systems. Disponible en [PDF] Computer Communication & Distributed Systems | Semantic Scholar.
[6] Girish Venkatachalam:Developing P2P Protocols across NAT. Disponible en Developing P2P Protocols across NAT (lut.fi).
[7] Gertjan Halkes & Johan Pouwelse (2011): UDP NAT and Firewall Puncturing in the Wild. doi:10.1007/978-3-642-20798-31. Disponible en UDP NAT and Firewall Puncturing in the Wild (inria.fr).