PROYECTO

BITTORRENT RÚSTICO

1er Cuatrimestre 2022

Equipo:
Los 4Rustásticos

Profesor:
Pablo A. Deymonnaz

Alumnos:

Dante Reinaudo	102848
Sofia Feijoo	101148
Facundo Milhas	102727
Tizziana Mazza Reta	101715

Índice

Objetivo 2

Funcionamiento 4

Aplicación 6

La conexión 10

Servidor 12

Concurrencia 13

Diagramas de secuencia 15

Interfaz gráfica 18

Agregado al proyecto: Tracker BitTorrent 20

Objetivo

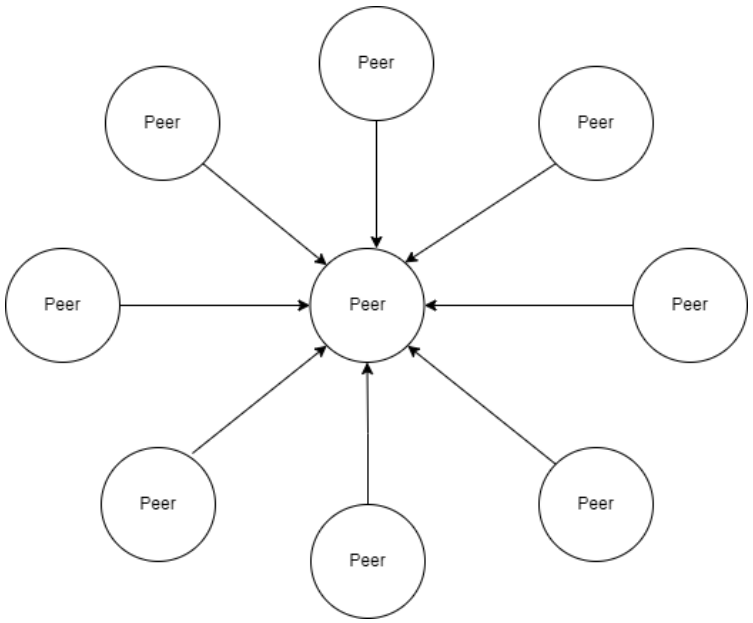
El objetivo de este proyecto es armar un Cliente BitTorrent con un conjunto de funcionalidades, como por ejemplo, siendo cliente, poder descargar torrents, y siendo servidor poder recibir pedidos de otros clientes, en el lenguaje de programación RUST.

A su vez, el trabajo consiste en la aplicación del protocolo HTTP y una arquitectura de intercambio de archivos denominada P2P, que se detallarán más adelante. Para la realización de este proyecto se debe implementar el uso de threads, lo cual hace que sea más dinámico.

Por último, se espera tener una interfaz gráfica, desde la cual se puedan realizar las funcionalidades creadas para nuestro cliente BitTorrent.

Arquitectura

La arquitectura, como se nombró anteriormente, es de tipo P2P. Esto significa que no hay un servidor común a todos los peers presentes que los conecte entre sí, sino que es una arquitectura en donde los participantes, o sea los clientes, comparten sus recursos, y estos son accesibles para cualquier usuario sin necesidad de una entidad de por medio. Por lo que, cada uno de los clientes, se convierte tanto en proveedores como consumidores de servicios.



De esta forma, los nodos están todos conectados, y asumen roles equivalentes, en donde cada peer puede unirse o irse de la red cuando quiera. Además, no se depende de ningún nodo específico, lo cual le agrega robustez a la arquitectura, y la hace menos vulnerable.

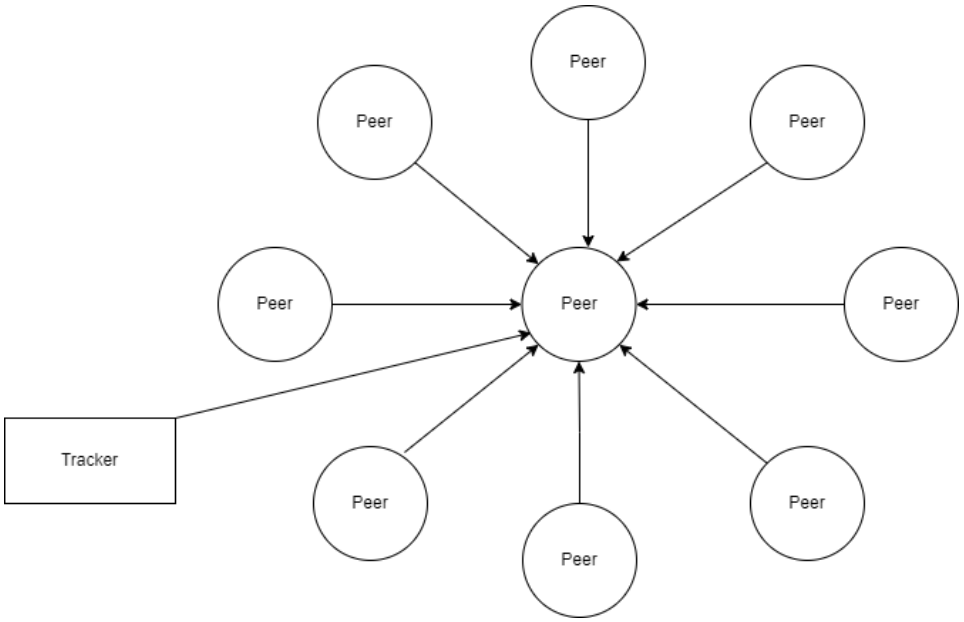
Funcionamiento

BitTorrent es un protocolo P2P diseñado para el intercambio de archivos, en donde se crea una super capa no estructurada sobre internet. No estructurada quiere decir que la super capa tiene una gran flexibilidad para construirse, y una gran tolerancia a que muchos peers se sumen y se bajen constantemente de la red.

En una red BitTorrent hay dos tipos de participantes: los peers y el tracker. Los peers son los encargados de distribuir las piezas por la red, y el tracker registra y coordina a los participantes.

Los peers pueden ser clasificados como seeders o leechers. Los seeders son peers que tienen todas las piezas del torrent, mientras que a los leechers aún le faltan piezas por descargar.

Los peers sólo se comunican con el tracker cuando quieren unirse a la red o cuando quieren intercambiar piezas con algún otro cliente.

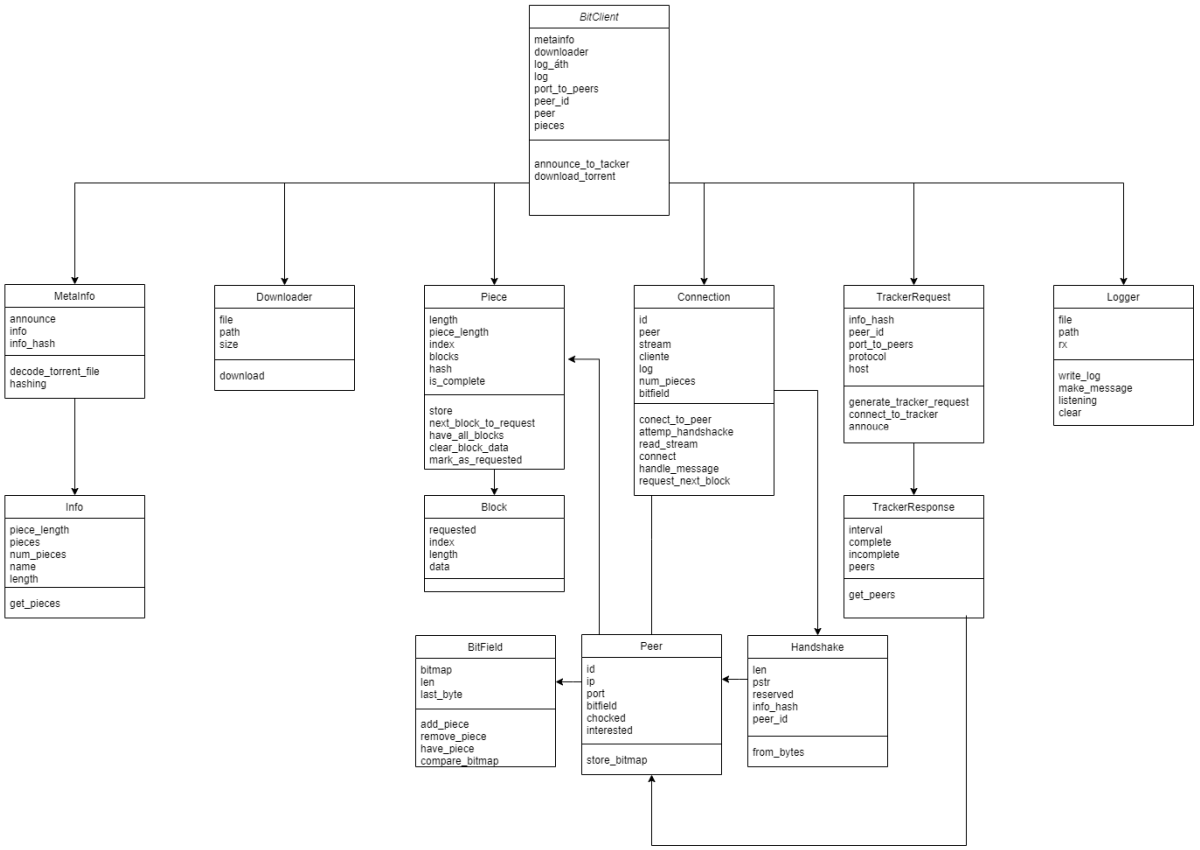


Cuando un peer quiere unirse a la red, lo primero que hace es descargar un archivo .torrent. Este archivo contiene los metadatos esenciales para el funcionamiento de la red. Además, estos metadatos identifican de forma única al torrent, contienen la dirección URL del tracker , la colección de archivos que forman el torrent, el tamaño de las piezas y el resumen de cada una de ellas. Una vez que el peer tiene el archivo .torrent, se conecta con el tracker mediante HTTP para anunciarse, es decir, para que el tracker sepa que hay un

nuevo peer que quiere participar en la red. Luego, el tracker devuelve una lista con la información de contacto de otros peers para que pueda conectarse con ellos y pueda intercambiar piezas.

Aplicación

La estructura principal de nuestro proyecto es BitClient, que ejecuta el peer local.



Podemos observar que se BitClient se relaciona con MetaInfo, que representa los metadatos contenidos en el archivo .torrent que le pasamos a través de su URL. MetaInfo contiene el info_hash, el announce y la información de las piezas a descargar, como fue explicado en el inciso anterior.

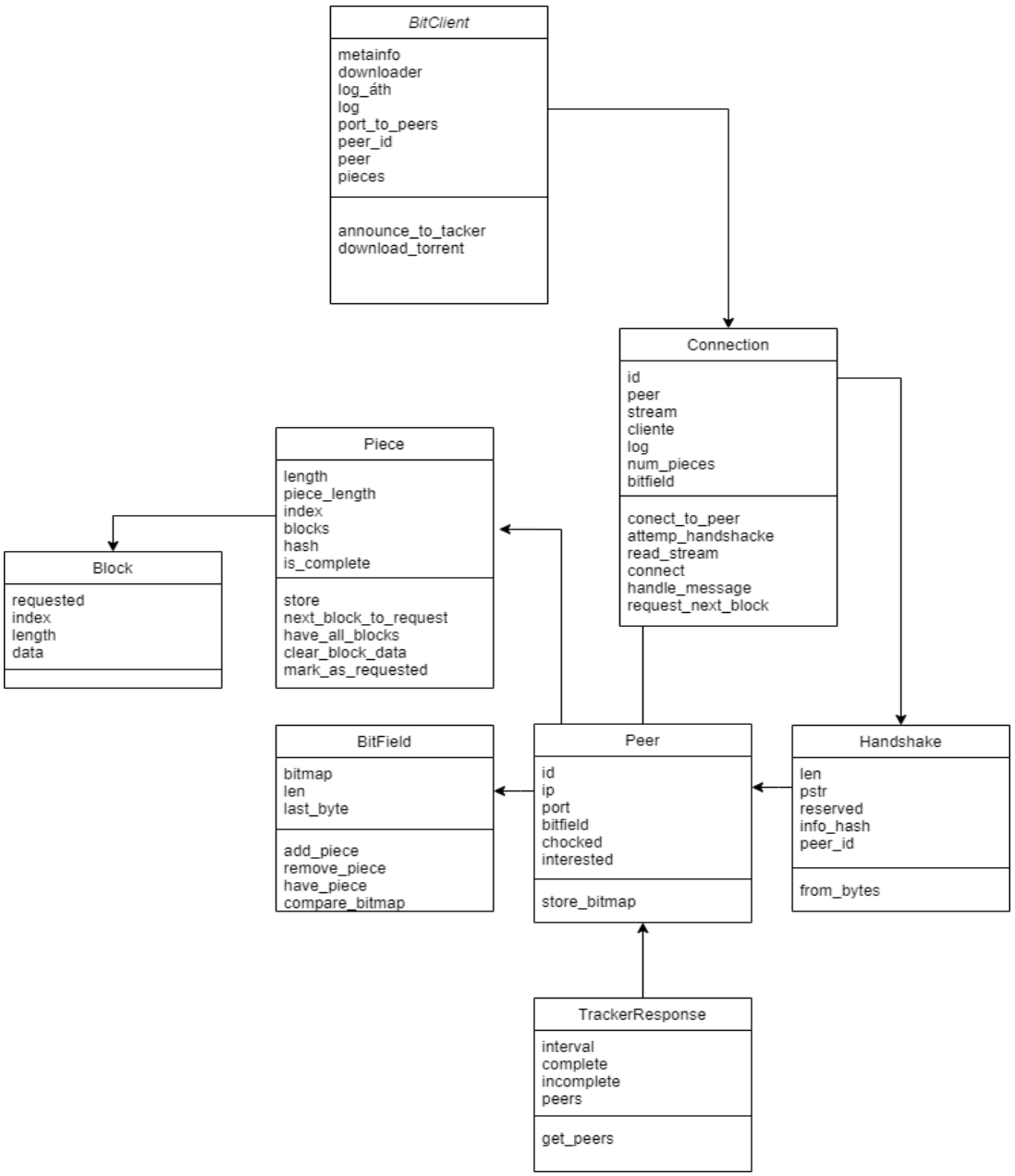
Para poder iniciar con la descarga, como peers, lo que debemos hacer, una vez inicializados, es anunciarnos ante el tracker mediante una TrackerRequest.

Para comunicarnos con el tracker debemos enviarle:

- info_hash -> 20 bytes que identifican al torrent de forma única.
- peer_id -> 20 bytes que sirve como identificador único del cliente, encodeada con urlencode.
- port -> número de puerto en el cual el cliente escucha conexiones.
- ip -> dirección ip del cliente.

- uploaded -> número que indica la cantidad de bytes subidos hasta el momento.
- downloaded -> número que indica la cantidad de bytes descargados hasta el momento.
- left -> número de bytes pendientes por descargar.
- event -> cadena que indica el estado del cliente (started, completed o stopped).

Luego, se obtiene una TrackerResponse con una respuesta que, en el mejor de los casos, contiene una lista de peers que tienen lo solicitado.



Una vez que tenemos los peers nos empezamos a conectar con cada uno de ellos, realizando lo que se llama un handshake. Para ello debemos enviar:

- El valor 19 en u8, que indica la longitud de la cadena de caracteres del nombre del protocolo.
- La cadena “BitTorrent protocol".
- El info_hash.
- El peed_id

Lo que sucede acá es que el peer al que le mandamos el handshake nos manda una handshake response con la misma información, la única diferencia es que nos manda su peer_id esta vez. Además, debe fijarse que el info_hash enviado corresponde a uno de los torrents que comparte.

Una vez exitoso el handshake entre peers, se sigue con un intercambio de mensajes para ver si efectivamente tienen las piezas que necesitamos y así descargarlas. Entre los mensajes de intercambio se encuentran:

- keep alive -> para mantener la conexión viva.
- bitfield -> un mapa de bits, que nos indica qué piezas tiene y cuáles no el peer en cuestión.
- interested -> se le comunica al receptor que estamos interesados en alguna de sus piezas.
- not interested -> se le comunica al receptor que no está interesado en ninguna de sus piezas
- have -> comunica si tiene o no una pieza determinada.
- chocke -> se bloquea el receptor del mensaje.
- unchocke -> se desbloquea el receptor del mensaje.
- request -> sirve para pedir un bloque previamente solicitado.
- piece -> sirve para enviar un bloque previamente solicitado.
- cancel -> cancela la petición.

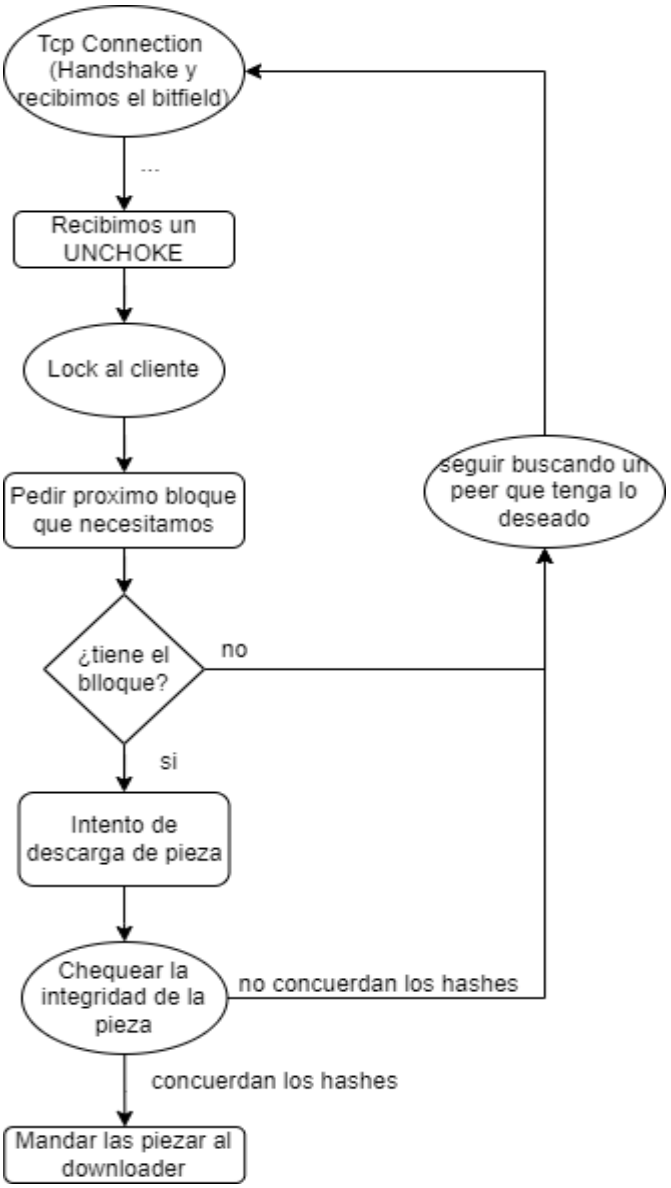
Este intercambio de mensajes se da hasta que tenemos la información necesaria y correcta para comenzar a descargar piezas. Cada una de estas piezas está conformada por bloques de igual tamaño. Cuando nos conectamos con los respectivos peers, le vamos pidiendo, en orden, cada bloque de la primera pieza. Una vez que tenemos todos los bloques de nuestra pieza, se verifica que esta esté completa y correcta, comparando con el hash_info obtenido anteriormente, y asegurándonos de que salió todo bien. Una vez que esto es verificado, se agarran

todos esos bloques de la pieza, se unifican y se descarga mediante el Downloader. Así hasta completar el proceso con todas las piezas necesarias.

Así mismo, algo que sucede en todo momento es que se loggean estos pasos nombrados anteriormente, indicando si se obtuvo o no una respuesta correcta del tracker, por ejemplo, o si la descarga fue o no exitosa. Teniendo un seguimiento exacto de lo que va ocurriendo mientras corre el programa.

La conexión

Ahondando un poco más en la conexión entre peers, tenemos una entidad, Connection, que se encarga de toda la parte de el handshake y los mensajes entre ellos, algo que ya comentamos. Pero además de eso, lo que también maneja, es la parte desde la cual ya sabemos que se empiezan a enviar y recibir las piezas.



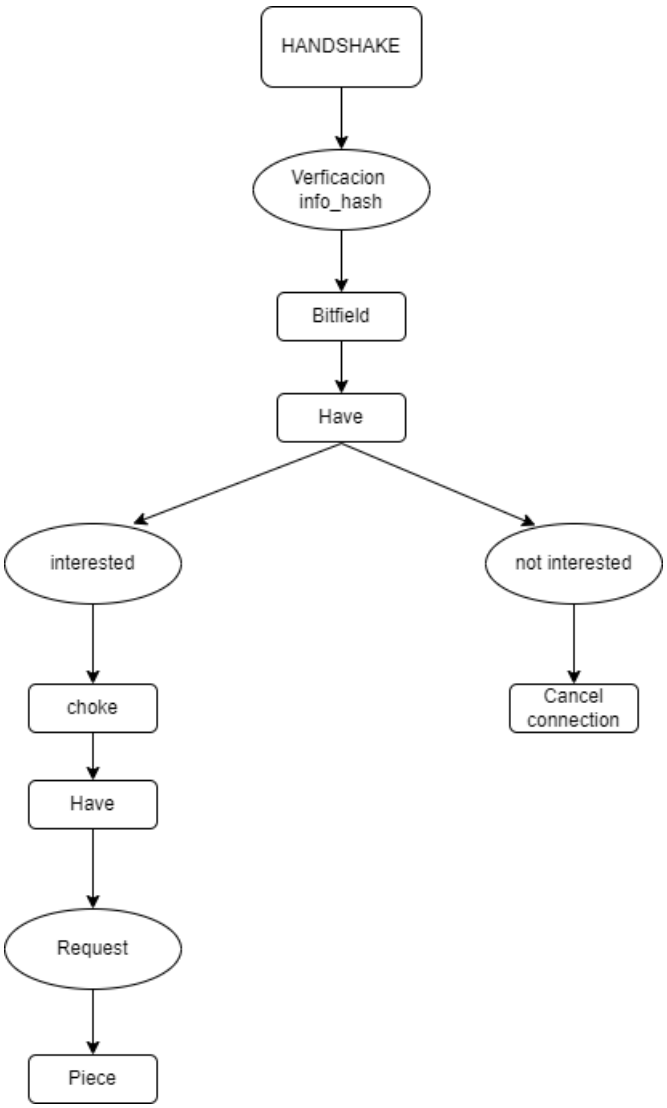
Cuando, como cliente, se recibe unchoke desde el otro lado, se sabe que ya está todo listo para comenzar a pedir piezas. La conexión le pregunta al cliente cual es el próximo bloque que esta necesita, por lo que el cliente busca en su arreglo de piezas, comparándolo con el bitfield del peer al que está conectado, y saber cual es la próxima pieza que se necesita y que puede llegar a pedirle a dicho peer. Cuando la encuentra,

lo que sucede es que se fija en dicha pieza cuál es el siguiente bloque que necesita, y entonces la pieza busca en su arreglo y le devuelve el bloque buscado. De esta manera el cliente le devuelve a la conexión el índice de la pieza que necesita, el offset y el tamaño para descargar. Y de esta manera vamos obteniendo los bloques de cada pieza que estamos pidiendo.

Servidor

Nosotros, como server, lo primero que esperamos es el handshake. Una vez que lo recibimos nos fijamos en que tengamos el mismo info_hash. En caso de no tenerlo, entonces no es válido.

Una vez que el handshake sale satisfactorio, comenzamos con el envío de mensajes con el cliente. Primero enviamos el bitfield, y después el mensaje de have de alguna pieza. Una vez que sucede esto esperamos el siguiente mensaje que debería ser un interested o un not interested. En el caso de que el peer no esté interesado lo que hacemos es cerrar la conexión, sino enviamos un unchocke y luego un have, para que después nos empiezan a pedir piezas a partir de request.



En el caso de recibir otro tipo de mensajes que no están contemplados, el manejo que decidimos tomar es de no aceptarlo y cerrar la conexión.

Concurrencia

Ahora pasamos a comentar cómo es que es que manejamos la concurrencia. Para ello vamos a dividirnos en tres partes:

Logger:

Disparamos un thread para el logger, por lo que tenemos un hilo principal y abrimos un hilo en paralelo. Al tener este hilo en paralelo en el logger, lo que se hace es mandarle la información que queremos que obtenga por un canal, un channel, y se la pasamos desde el cliente. El logger la recibe y la guarda. De esta forma, podemos tener el seguimiento deseado del programa, y no estamos, en cada paso, parando y reanudando las ejecuciones para ir a loggear.

Servidor:

Tenemos un hilo que es utilizado por el servidor. Esto se debe a que estamos escuchando conexiones, y además estamos descargandonos piezas y comunicándonos con peers como clientes, por lo que necesitamos un hilo en paralelo así no debemos ir parando las ejecuciones que se están realizando en un momento dado para ocuparnos de otras cosas que nos vienen llegando y debemos responder.

En el caso del servidor, tenemos un único hilo, en el que nos llegan las requests pidiéndonos piezas, y nosotros lo que hacemos es encargarnos de a una request a la vez, haciendo los pasos que ya sabemos, handshake, mensajes y envío de piezas y cuando terminamos la comunicación con ese peer en concreto, atendemos al siguiente, y así sucesivamente.

Clientes:

Aquí necesitamos utilizar los threads para poder hacer las descargas en paralelo y que, además, sea muchísimo más rápido este proceso. Pero debemos tener cuidado y manejar el hecho de no estar pidiéndole a todos lo mismo!

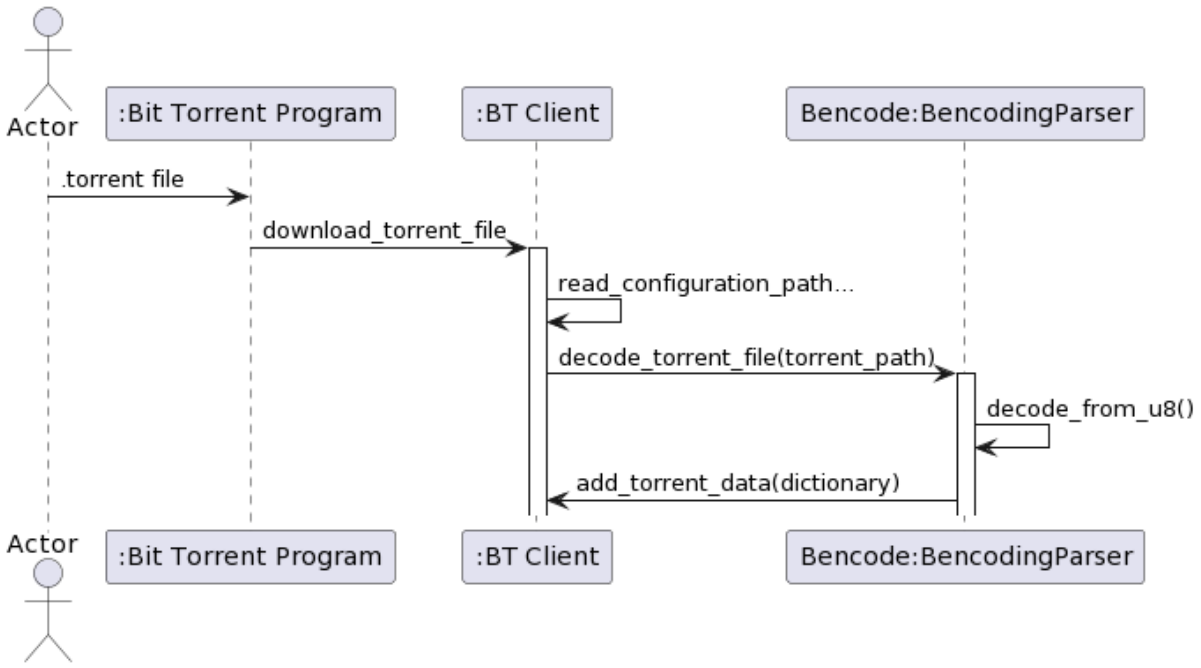
Lo que sucede es que se establece la conexión en simultáneo con cada peer que nos mandan en la response, y nos comunicamos en paralelo con cada uno de ellos.

¿Cómo sucede esto? Bueno, tenemos una referencia a las conexiones, la disparamos en distintos threads, por lo que al cliente lo metemos en un mutex, para evitar que todos al mismo tiempo intenten comunicarse con el mismo, esto dentro de una referencia mutable.

¿Cómo hacemos para, dado que cada conexión le quiere pedir un bloque al cliente, que este no le pida en simultáneo a muchas conexiones el mismo bloque y no descargue muchas veces lo mismo? Lo que se hace es que se piden los bloques por turnos. Por lo que las conexiones se turnan para pedir los bloques. Por ejemplo, si yo ahora voy a hablar con el cliente, solo hablo yo, conseguimos el lock, y el cliente ahora, si es que tiene lo que necesitamos, va y busca que bloque necesita y se lo comunica. Entonces, cuando yo busque otro bloque y venga la próxima conexión, no sabe si ya se descargo o no el bloque pedido anteriormente, pero sí sabe que ya lo pidió, entonces a esta nueva conexión le pide otro bloque, que no es ninguno de los ya pedidos con anterioridad.

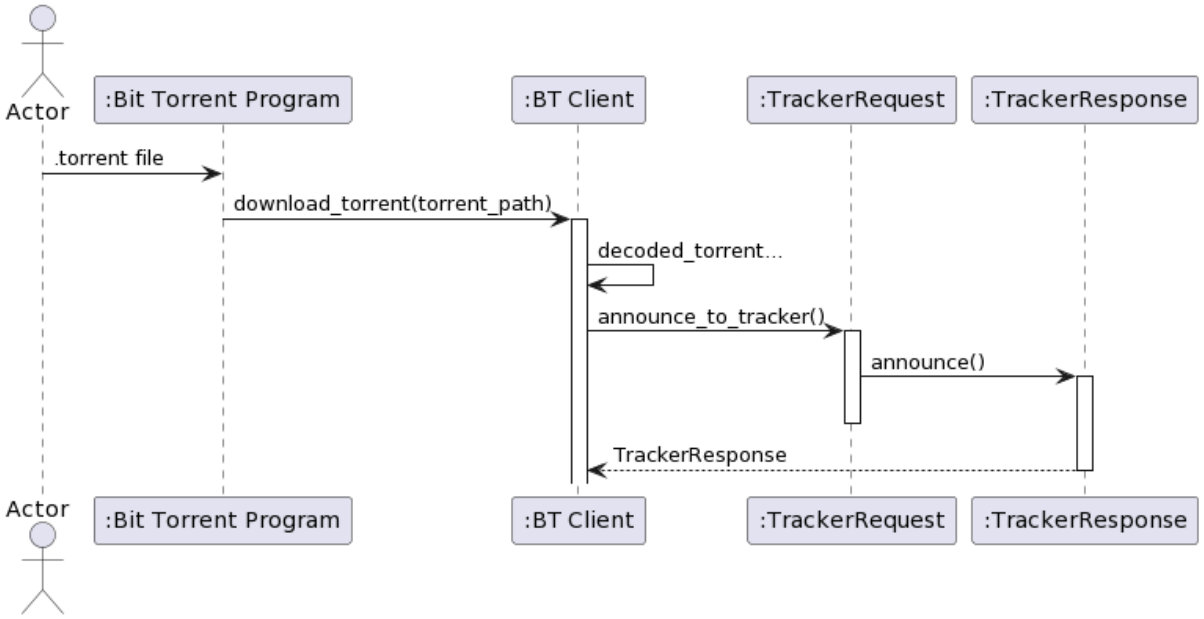
Diagramas de secuencia

Comenzamos con el siguiente diagrama de secuencia, que se da a partir de la llegada de un archivo .torrent.

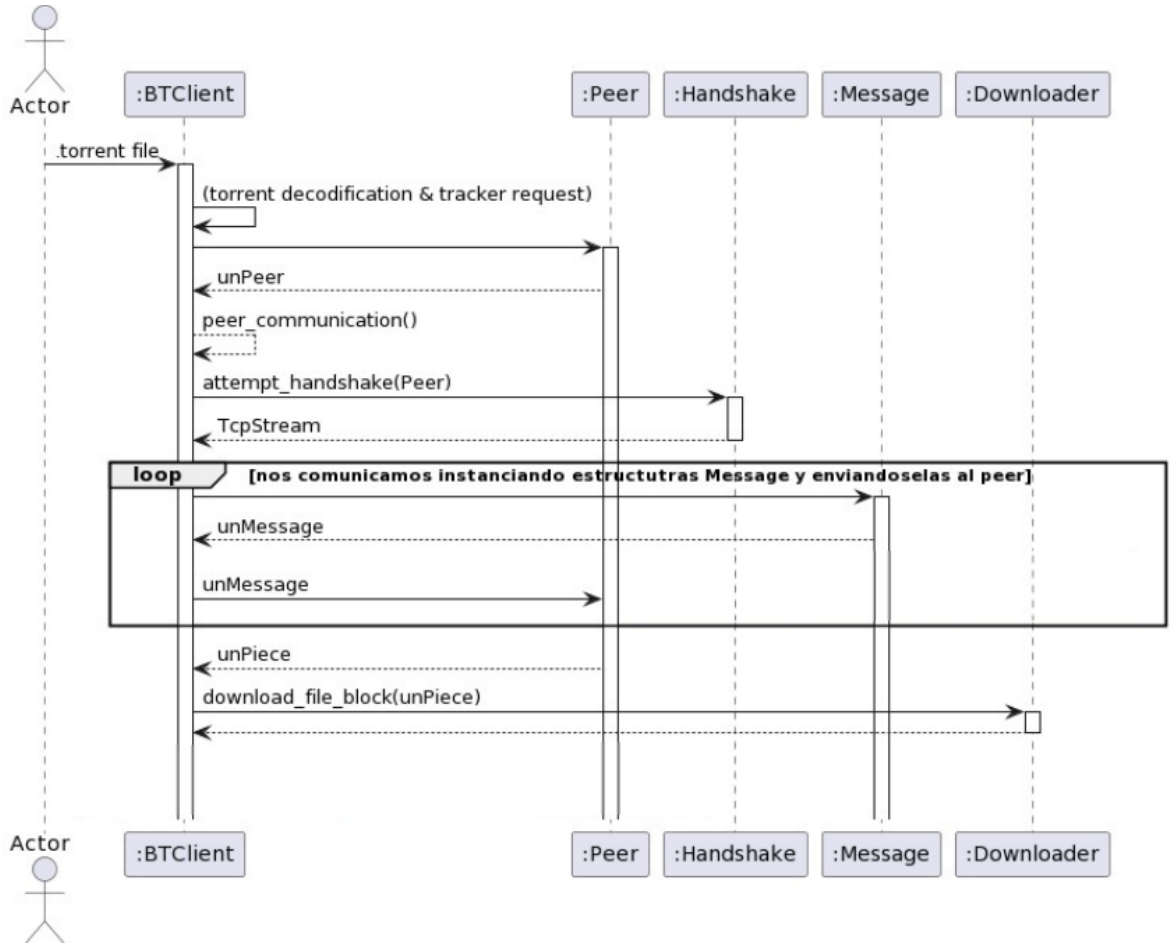


Una vez que tenemos el archivo .torrent lo que hacemos es bencodearlo, ya que nos llega en un formato u8, el cual nosotros no podemos interpretar. Este diagrama pasa por alto algunas de los pasos que pasan en el medio hasta llegar al parser, sólo para que se entienda que en algún momento, desde que llega el torrent file hasta que empezamos, efectivamente, a comunicarnos con el tracker, sucede esta transformación de la información para poder interpretarla correctamente, y acceder a las piezas.

El segundo diagrama nos muestra más en detalle la comunicación se da con el tracker. El bitClient hace uso de la estructura TrackerRequest para ordenar y modelar los campos a enviar al Tracker. Con la misma hacemos el announce, recibiendo la respuesta y almacenandola en la estructura TrackerResponce. Esta response contiene una lista de los peers con los que debemos comunicarnos a continuación.



El diagrama a continuación muestra en detalle la comunicación con los peers devueltos por la tracker response vista anteriormente.



Podemos apreciar cómo es que al comienzo de la comunicación, lo que se hace es el handshake con el peer, para luego pasar al intercambio de mensajes.

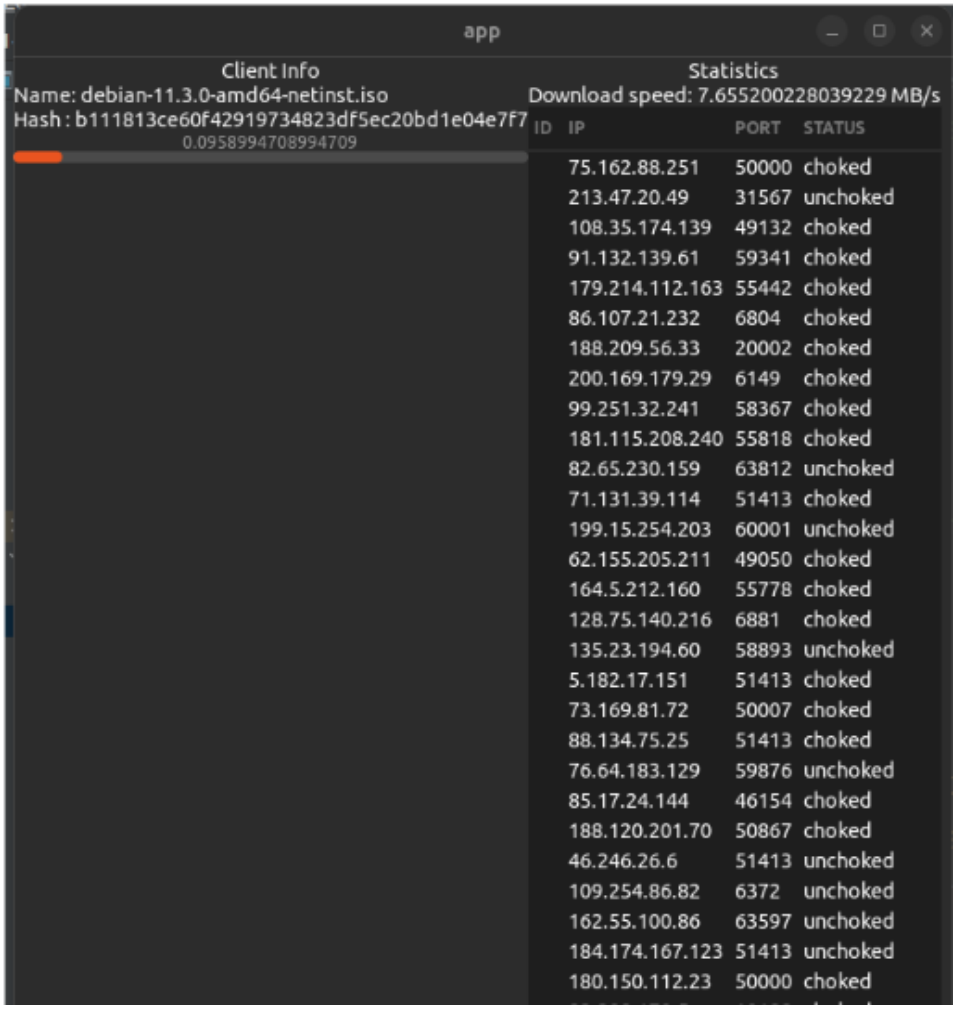
Una vez enviados los mensajes correspondientes, se realiza la request de la pieza que nos interesa. Cuando esa pieza es recibida se pasa a la entidad Downloader, que crea un archivo y lo almacena en disco.

Interfaz gráfica

Para la realización de la interfaz gráfica se utilizó GTK4.

La estructura de la interfaz está en un XML, que se levanta desde el main del mismo, y se van actualizando los contenidos de los componentes que trae el esqueleto de esa XML.

Desde la app levantamos un cliente, creamos un channel, que es nuestro EventBus, donde la api, o sea el cliente, va a registrar distintos eventos que la app va a estar escuchando, y en base a eso se actualiza la vista, que se puede ver a continuación.



Algunos eventos pueden ser: que se descargó una pieza, la velocidad de descarga, la información del cliente, la información de los peers, que se escuchan a través del EventBus.

El programa nos muestra que se descargaron todas las piezas, y que el torrent fue descargado exitosamente.

```
[CONEXION 28] Descargamos todas las piezas posibles de este peer
[CONEXION 41] Descargamos todas las piezas posibles de este peer
[CONEXION 7] Descargamos todas las piezas posibles de este peer
[CONEXION 0] Recibi una pieza: 1508, offset: 163840
[DESCARGA] Se completo la pieza 1508 y es correcta, la guardo en el archivo.
[CONEXION 0] Descargamos todas las piezas posibles de este peer
[CONEXION 0] Recibi una pieza: 1511, offset: 180224
[DESCARGA] Se completo la pieza 1511 y es correcta, la guardo en el archivo.
[CONEXION 0] Descargamos todas las piezas posibles de este peer
[CONEXION 48] Recibi una pieza: 1469, offset: 180224
[DESCARGA] Se completo la pieza 1469 y es correcta, la guardo en el archivo.
Torrent is complete
```

Y por otro lado, desde la interfaz, se puede apreciar la descarga completa.

Por último, mostramos el archivo que se creó de logger, mostrando cada una de las acciones que se fueron llevando a cabo a medida que se hacía la descarga del torrent.

Tracker BitTorrent

Como parte final del proyecto, se hizo un agregado que consiste en implementar un tracker de BitTorrent. El mismo recibe notificaciones de disponibilidad de archivos que están disponibles por los peers, teniendo en cuenta el protocolo HTTP/1.1.

El protocolo es accedido a través del endpoint '/announce'. También se implementó el endpoint '/stats' que, a través del tracker, muestra las estadísticas sobre los peers conectados, los peers con descarga completa y la cantidad de torrents en el tracker. Las estadísticas nombradas se manifiestan en un sitio web, que es accedido desde un browser. En la página web se puede visualizar la información de las estadísticas mediante unas pestañas que indican los períodos de tiempo fijos establecidos para ser mostrada. Los períodos de tiempo elegidos son: última hora, últimas 5 horas, último día y últimos 3 días. De esta forma, las estadísticas a mostrar son amenas, ya que se las ve con gráficos de barras, en donde se representa la línea de tiempo, cualquiera de las nombradas anteriormente, y en ese tiempo se ve la cantidad de, por ejemplo, los peers que completaron la descarga en las últimas 5 horas.

A continuación, se pasará a detallar la solución propuesta para resolver la consigna del agregado. En primer lugar utilizamos la aplicación *Transmission* para poder crear nuestros propios archivos torrents, y así poder asociarlos a nuestro tracker, al cual le asignamos el puerto: 127.0.0.1:8080.

Dado que nuestro Tracker debe tener conocimiento de aquellos torrents que debe hostear, al levantar el programa por línea de comando le pasamos la ruta a un archivo de configuración, el cual cuenta con las rutas de todos los torrent asociados a él. Estas son las estructuras intervinientes:

- BitTracker: esta estructura cuenta con una lista de struct Torrents a hostear, un id como identificador y el sender de un canal para comunicarse con el struct Logger. Al inicializarse, recibe la ruta al archivo de configuración, el cual cuenta con una ruta a un torrent por cada línea. Inicializa un struct Torrent por cada línea del archivo. Por otro lado esta estructura actúa como servidor concurrente: se queda a la espera de

nuevas conexiones y dispara un hilo por cada nueva conexión establecida.

- Torrent: sus atributos son la metainfo asociada al torrent y una lista de struct Peers conectados. Al inicializarse recibe una ruta al archivo .torrent que está modelando, parsea su metainfo y la almacena. Cada vez que un peer a través del endpoint de /announce solicita este torrent específico, se almacena.

- Connection: Modela la conexión entre nuestro tracker y los peers o la aplicación web. Se encarga de leer desde la conexión abierta, manejar los mensajes y responder acorde a ellos. Mediante este canal podemos recibir dos peticiones: announce o stats. En caso de que un peer se anuncie mediante el endpoint de announce, el tracker almacenará esta información y generará la respuesta. En caso de que recibir una request en el endpoint de stats, expondrá la información necesaria para calcular las estadísticas deseadas. En particular, devuelve una lista de peers con información de la última conexión, estado y torrent solicitados de cada uno de ellos.

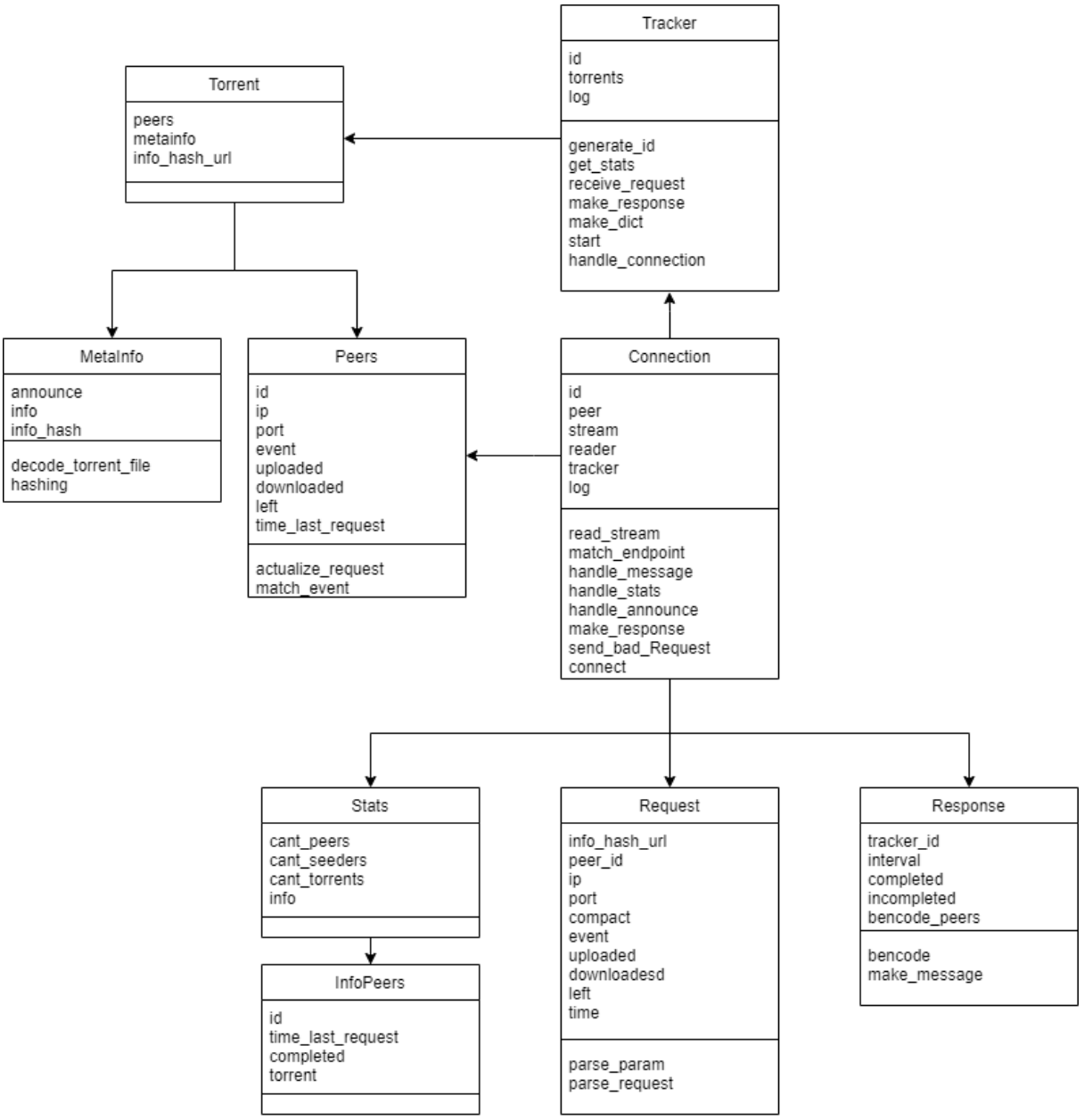
- Peer: en este struct guardaremos información del estado de cada peer y la actualizaremos cuando sea necesario. Es necesario que el tracker almacene datos de los peers tales como el estado de su descarga (que puede ser completo o incompleto), la fecha y la hora de su última conexión, progreso de la descarga, etc. Esta información es la que el tracker puede recolectar con las requests de cada peer al endpoint announce.

- Request: modela la request de un peer cuando se comunica con el tracker a través del endpoint de announce. La request tiene información como el id del peer, su puerto, la hora y demás valores útiles para el cálculo de estadísticas relacionadas a las descargas de torrents.

- Response: modela la respuesta del tracker en el endpoint de announce. Esta estructura cuenta con un builder que construye el mensaje y lo encodea como corresponda.

- Stats: modela la respuesta del tracker en el endpoint de stats. En esta respuesta se envía a quien lo solicite la cantidad de seeders, peers y torrents y una lista de peers con información (como id, hora, estado) sobre cada uno de ellos. Esta respuesta va a ser serializada a un Json para poder ser enviada correctamente.

Podemos apreciar a continuación un diagrama, en el que se modela las relaciones que existen entre las estructuras nombradas.

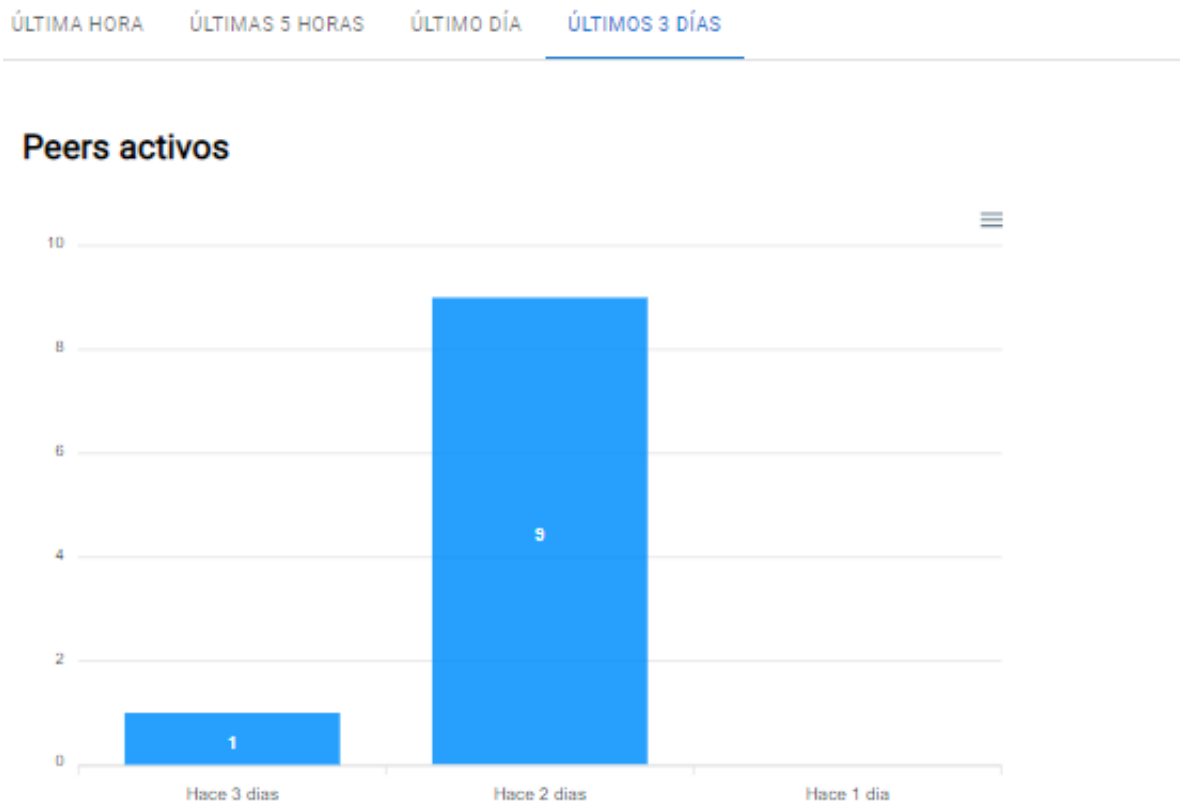


Además, creamos una aplicación web que le solicita los stats al tracker y los visualiza en un browser. Esta aplicación fue creada con React y algunas librerías para estilos y conexiones http.

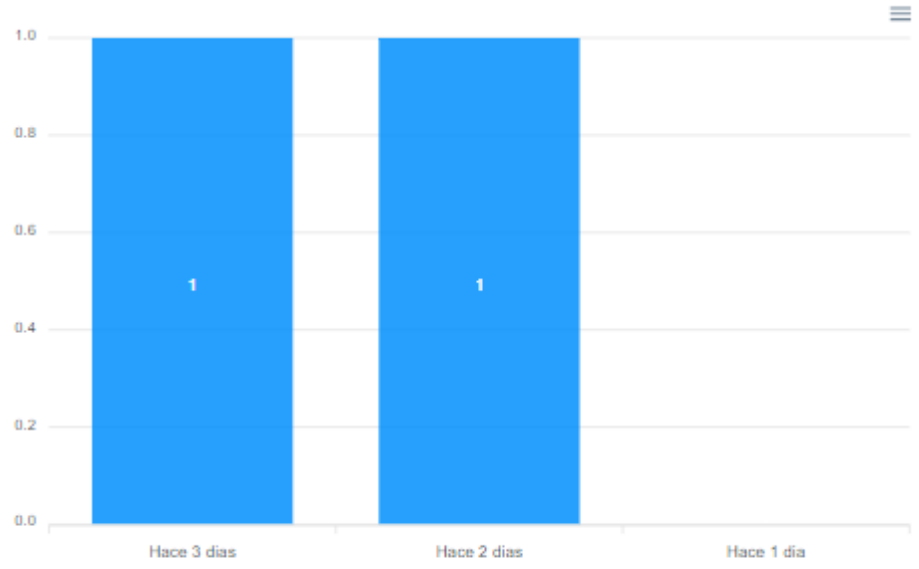
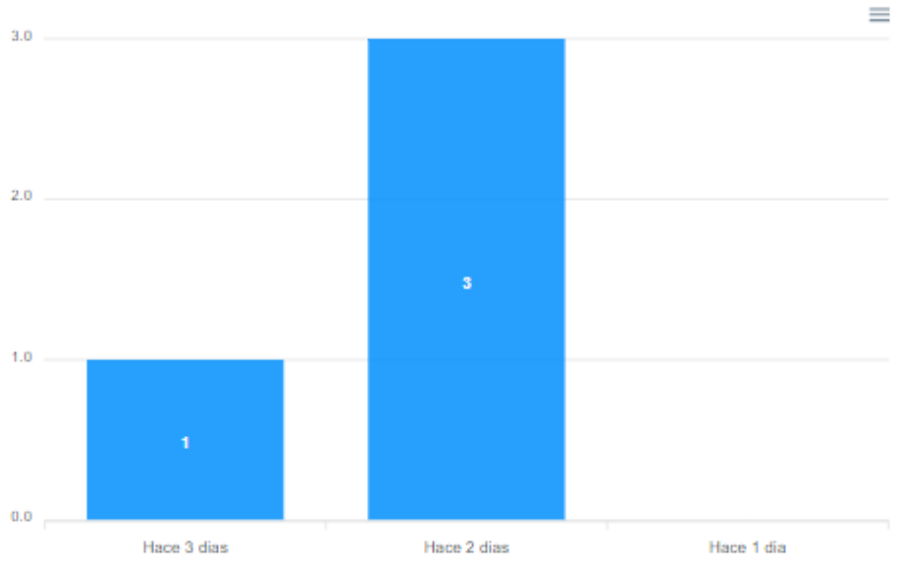
En concreto la aplicación web realiza un get al endpoint de stats del tracker y así recibe en un json la respuesta con la información esperada. Con esta información filtra los peers según el tiempo y la información a visualizar. Los filtros temporales son según registros de la última hora, últimas dos horas, último día y últimos tres días. A su vez cada filtro mostrará la información mediante un gráfico de barras dividido en buckets de una hora o de un día dependiendo el filtro.

Con la información que nos envía el tracker podemos reconstruir tres stats principales: la cantidad de torrents descargados en un cierto tiempo, la cantidad de peers activos en un cierto tiempo y la cantidad de descargas completadas en un cierto tiempo también.

A continuación se muestran una seguidilla de imágenes, en las que se puede observar esto mismo, seleccionando la pestaña de "Últimos 3 días" para ver las estadísticas.



Peers que completaron la descarga



Torrents descargados