FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

# Taller de Programación
# Proyecto Cliente de BitTorrent

## Grupo Crab Rave - 1er Cuatrimestre 2022


| | | |
|---|---|---|
| Semorile, Gabriel | 105681 | gsemorile@fi.uba.ar |
| Rizzo Ehrenbock, Gonzalo | 106475 | grizzoe@fi.uba.ar |
| Nicolini, Franco | 105632 | frnicolini@fi.uba.ar |
| Torres Dalmas, Nicolás | 98439 | ntorresdalma@fi.uba.ar |

# Índice

# Introducción

## Protocolo BitTorrent

Protocolo utilizado para el intercambio de archivos de manera descentralizada usando la arquitectura Peer to Peer (P2P). En una arquitectura P2P, los clientes pueden conectarse entre sí, sin la necesidad de pasar por un servidor centralizado, de esta manera, todos los clientes colaboran entre sí transfiriendo pequeñas partes del archivo en cuestión.

En particular, este sistema de clientes conectados pueden interpretar los archivos con la extensión .torrent. Este archivo contiene datos como la longitud del archivo, cantidad de piezas, e información para conectarnos con el Tracker. El Tracker funciona de orquestador de clientes para el archivo que se quiere descargar, éste nos brindará, a través del protocolo HTTP, una lista de hasta 50 clientes con su dirección ip, puerto e identificador.
Esta red de nodos nos garantizará la transferencia del archivo. Estos nodos se pueden identificar como Seeders ( nodo que tiene el archivo del .torrent completo y comparte sus piezas), Peers ( nodo que está descargando un archivo y tiene piezas que puede compartir a la red) y Leechers ( nodo que está descargando el archivo pero no comparte sus pieza). Entonces una vez me conecte con un nodo y me comparta una pieza, ya puedo compartir esa pieza a quien me lo pida, es decir, me agrego a la red de nodos del archivo en cuestión.

La información del .torrent y la respuesta del tracker están codificadas con el formato Bencoding.

## Herramientas

Para el desarrollo de la implementación de un Cliente de BitTorrent Rústico, se utilizaron las siguientes herramientas:

- *Rust 1.61.0*
- *Cargo 1.61.0*
- *Dependencias:*
    - *native-tls 0.2*
    - *chrono 0.4.19*
    - *sha1 0.10.1*
    - *gtk3*
    - *glib*

# Arquitectura

## Comportamiento general y sus componentes principales

Para la implementación de este Cliente de '*BitTorrent*' comenzamos por leer el '*.torrent*' y decodificarlo con nuestras estructuras '*Bdecoder*' y '*Bencoder*'. Luego armamos el '*request*' para el *'Tracker*' con los datos previamente decodificados. Una vez obtenemos la lista de nodos que nos pueden compartir las piezas del archivo, vamos creando nuestros '*Peers*' para comenzar la conexión usando nuestro 'PeerConnection' y posteriormente pedirle las piezas que necesitamos a través del *'DownloadManager'*. Al terminar la descarga de cada pieza se verificará que sea correcta y se guardará en el directorio establecido. Por otro lado, también vamos a estar escuchando mensajes con nuestro *'Listener'* para poder compartir nuestras piezas con otro nodo de la red. La entidad encargada de esto es ell *'UploadManager'*.

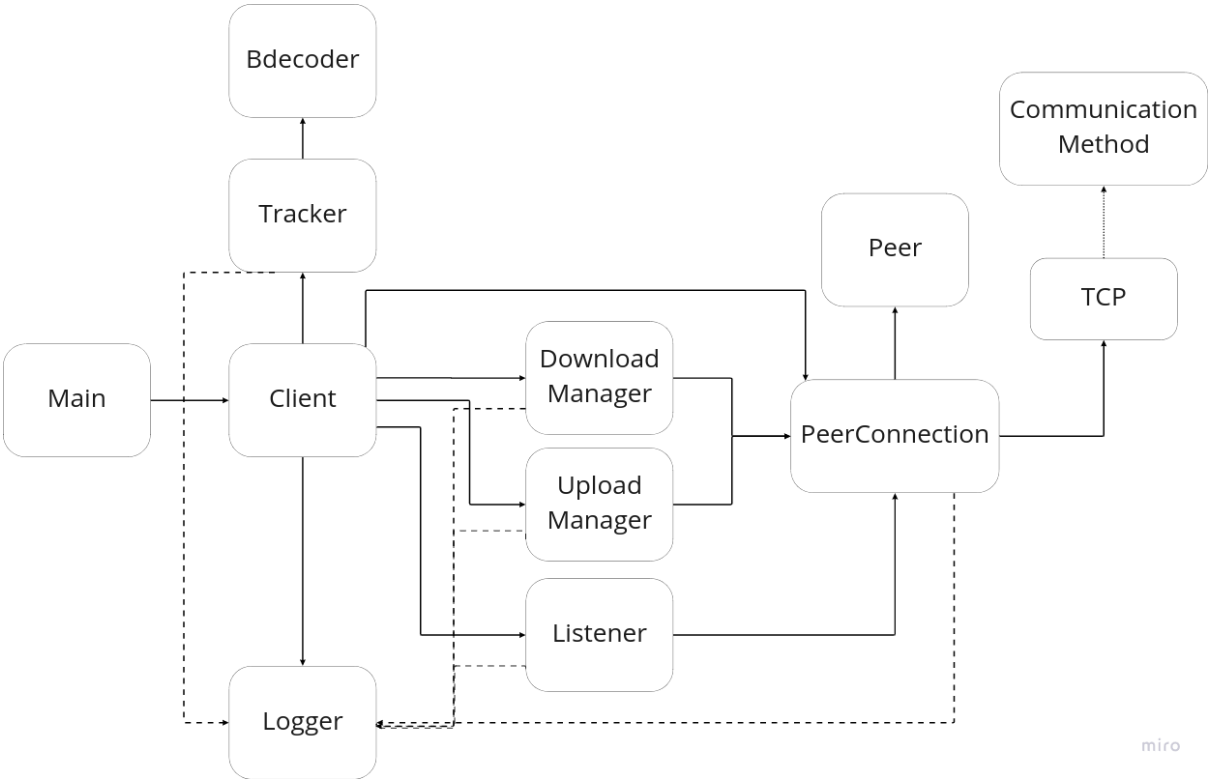



Imagen 1 - Diagrama General

# Comunicación entre nodos de la red

En este protocolo para hablar con nodos de la red se necesita un '*id*' y un '*infohash*'. Con estos datos se arma un '*handshake*' con el nodo a interactuar. Una vez terminado satisfactoriamente el intercambio del handshake con el nodo ya podemos comenzar a interactuar con los mensajes P2P del protocolo. También vamos a recibir el '*bitfield*' indicando que piezas nos puede compartir este nodo.

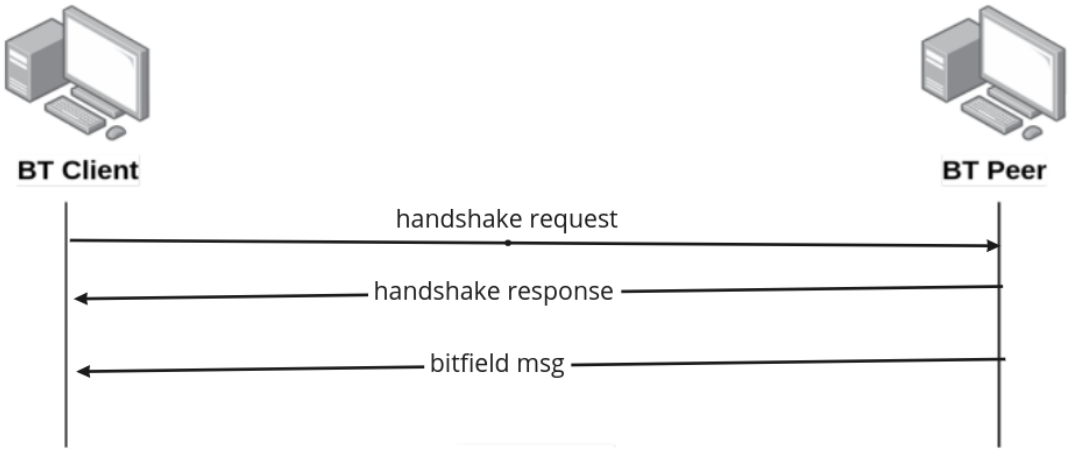



Imagen 2 - inicio de comunicación

Para descargar un archivo, vamos a desbloquear el nodo con un '*unchoke*' y vamos a mostrar interés en al menos una de sus piezas con un '*interested*', luego aguardamos a que nos desbloquee con un '*unchoke*' y a partir de ahí comienza el intercambio de mensajes '*request*' y '*piece*' para las piezas que nos pueda compartir, respectivamente.

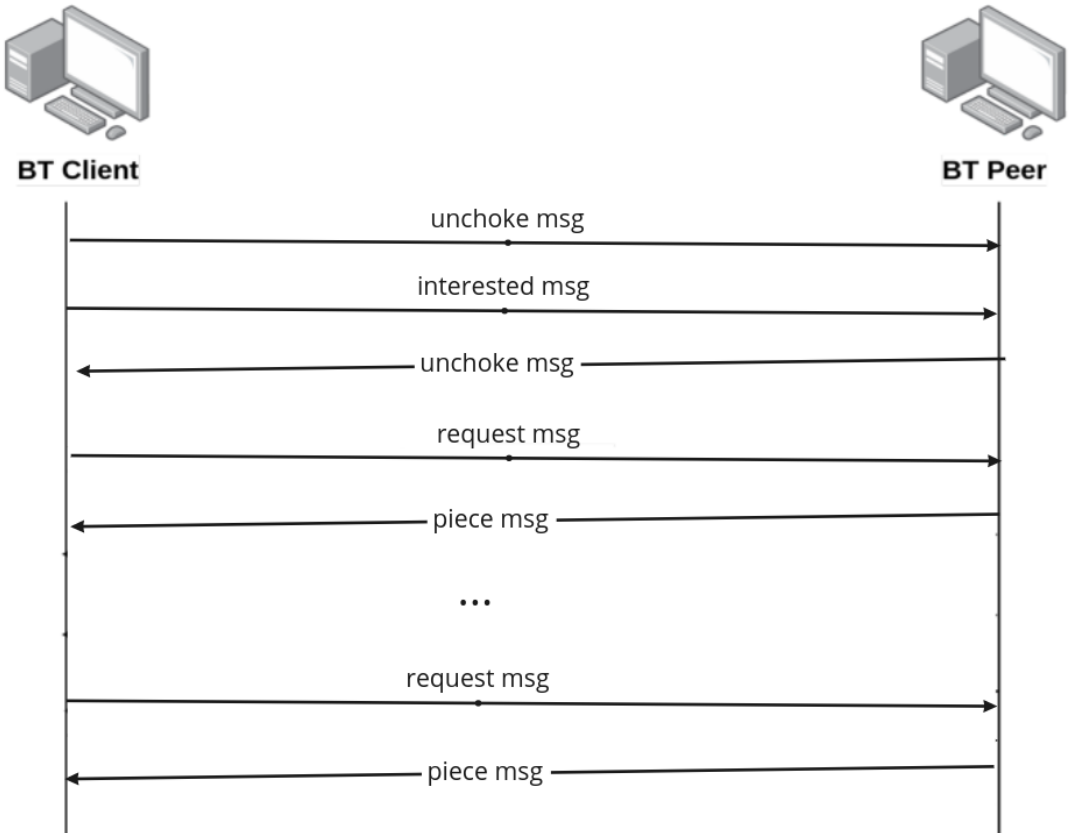



Imagen 3 - descarga de piezas

# Threading

Para la comunicación entre las partes, contamos con un thread principal. En el caso de descarga concurrente de muchos archivos se tendrán múltiples clientes, cada uno en un hilo.

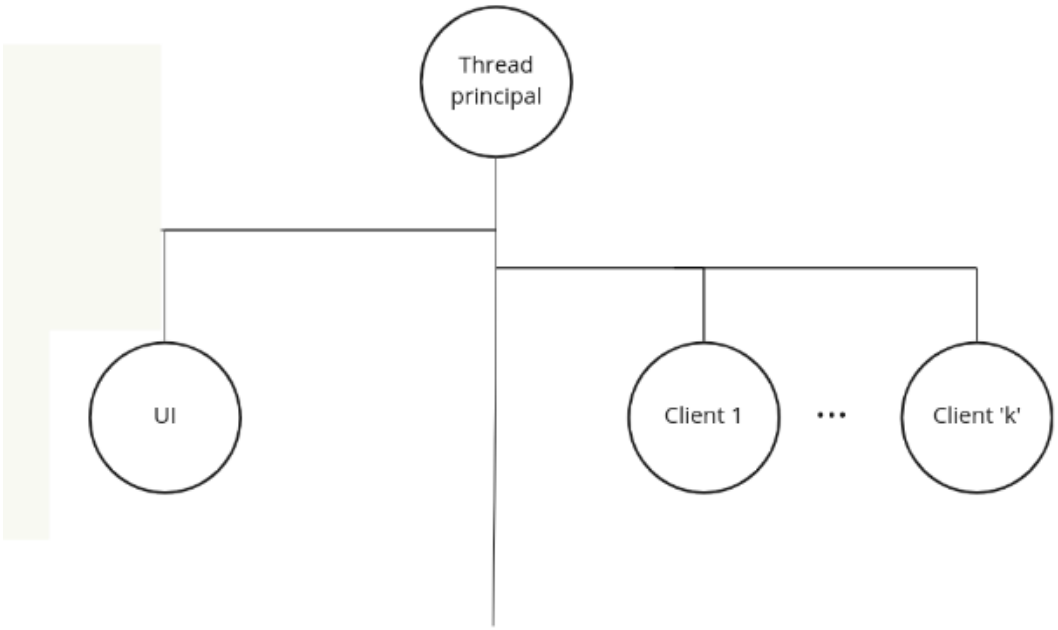



Imagen 4 - Thread Principal

A su vez, cada cliente se comunica con su propio Listener, UploadManager, Logger y DownloadManager. En este último para descargar las distintas piezas se hará uso de una threadpool que genera un hilo por cada peer al que se peticiona.

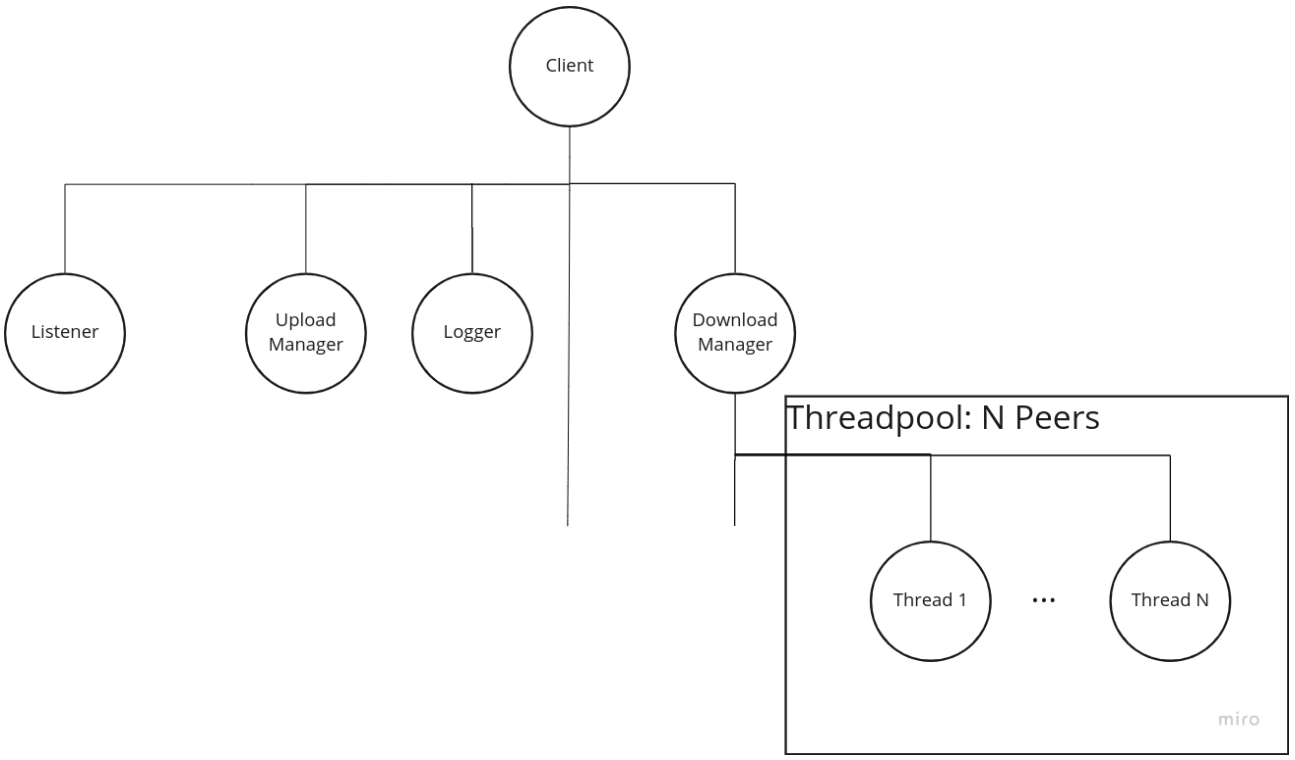
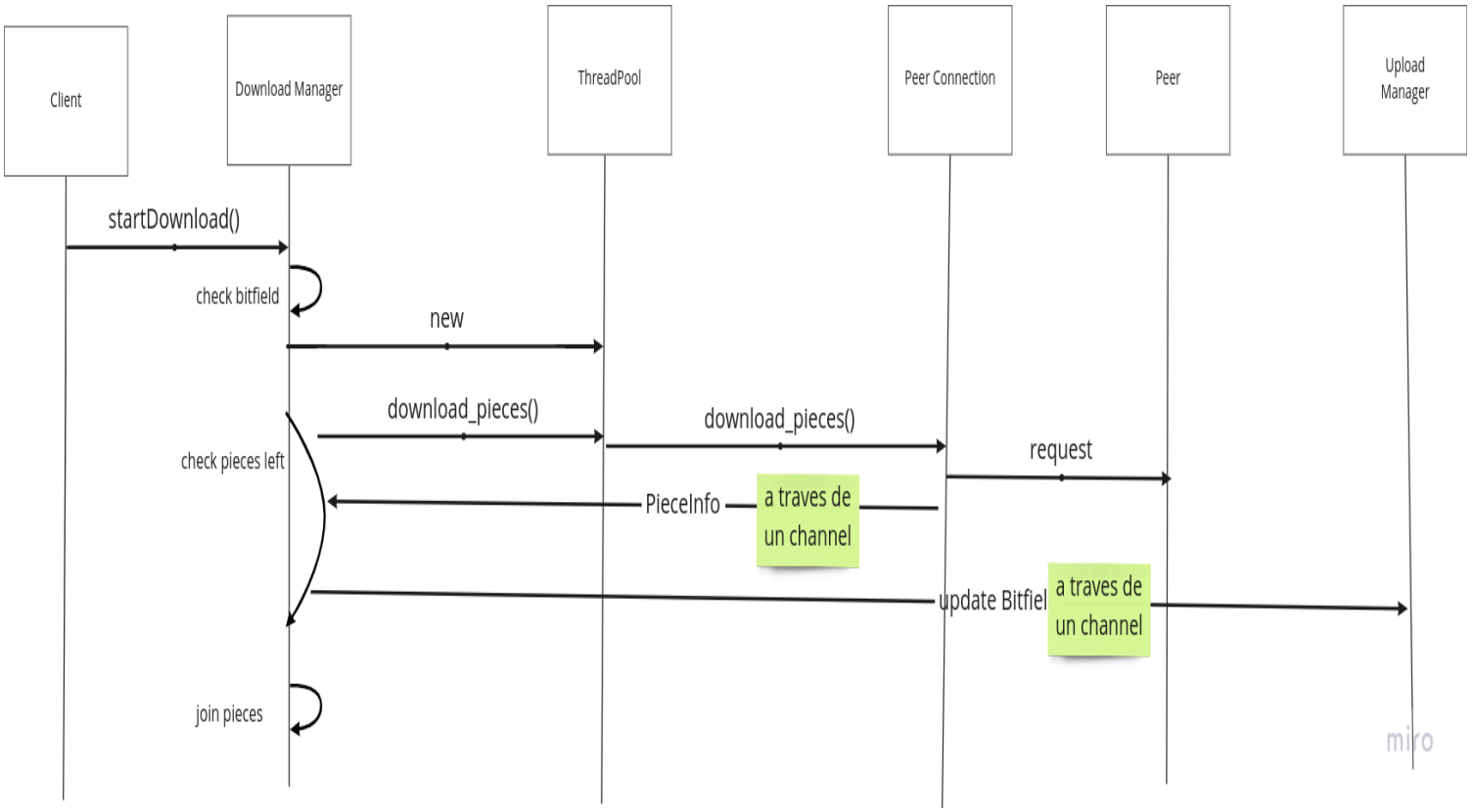



Imagen 5 - Threadpool y Thread Client

## Descargas

Primero para las descargas vamos a realizar las conexiones con '*PeerConnection*' a la lista de '*Peer*' del '*Client*'. Utilizaremos nuestra '*ThreadPool*' en donde se generara un hilo por cada peer disponible para descargar cada pieza r. Una vez tengo estas piezas verificadas, nuestro '*Download Manager*' se encarga de ensamblar todas las piezas para construir el archivo en cuestión.

Imagen 6 - descarga de piezas



## Cargas

Por otro lado, para las cargas vamos a chequear nuestro '*bitfield*' para saber que piezas tenemos descargadas. En este caso, vamos a tener '*IncomingPeer*' a diferencia del '*Peer*' del '*DownloadManager*', ya que de antemano no sabemos el '*id*' del nodo que nos quiere hablar.

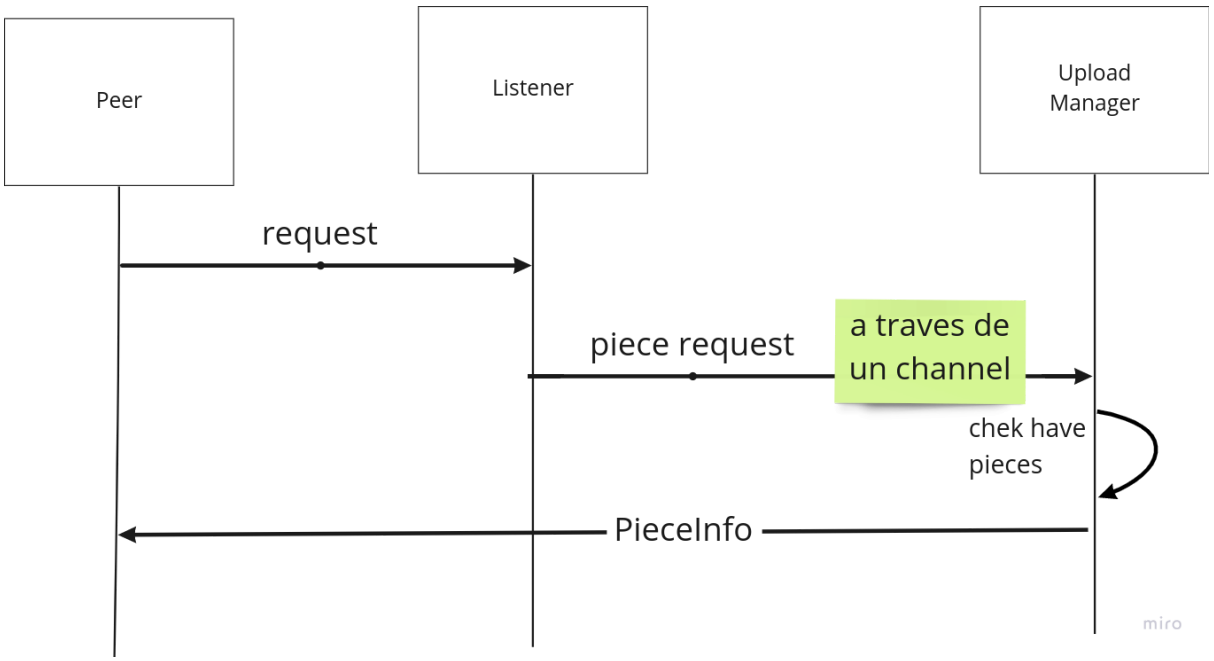



Imagen 7 - carga de piezas

# Interfaz gráfica

Para el desarrollo de la interfaz gráfica se usó la herramienta GTK3, y a través de su complemento ‘glade’ pudimos generar el diseño fácilmente. Al correr el programa lo primero en correr es la UI la cual es la encargada de iniciar al Client en un thread diferente.

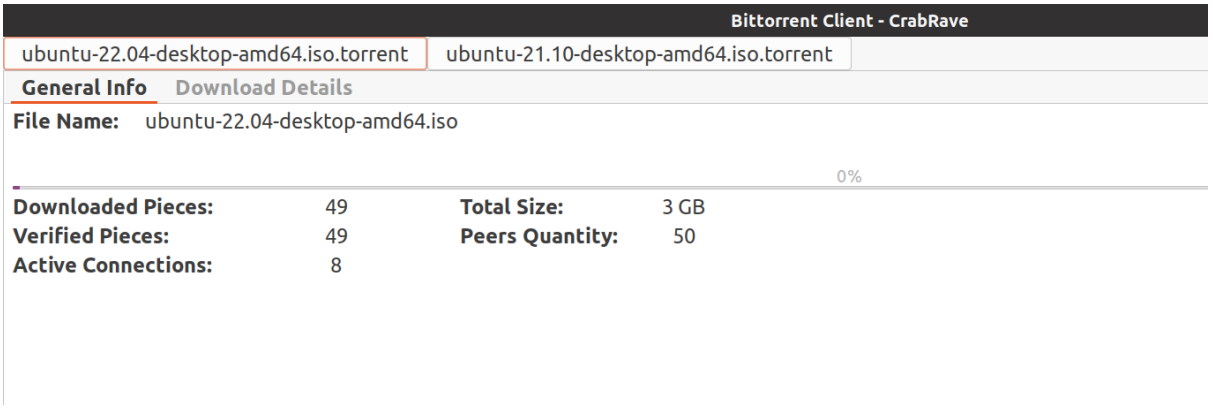



Imagen 8 - pestaña de información general

En la imagen anterior se muestra la **Pestaña de información general:** la cual cuenta con la información general de los torrents en curso, incluyendo: nombre del archivo, tamaño total, cantidad de piezas, cantidad de peers, además del estado general de la descarga (porcentaje de completitud, cantidad de piezas descargadas y verificadas) y la cantidad de conexiones activas correspondiente a cada torrent.

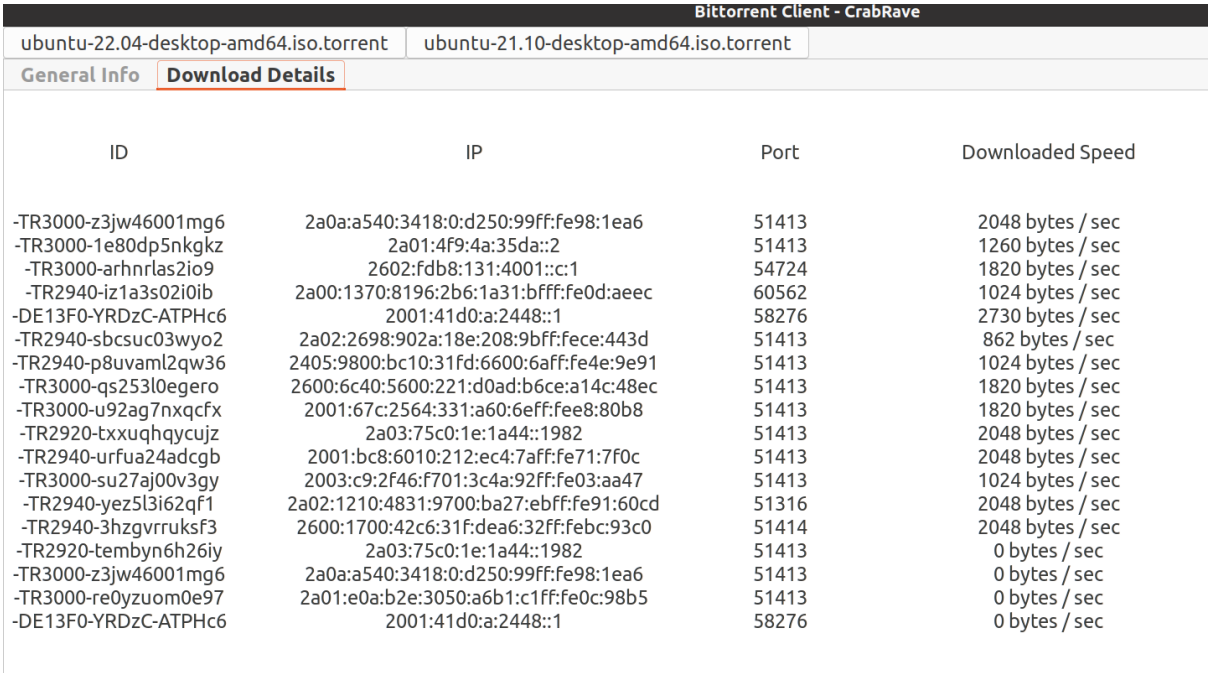
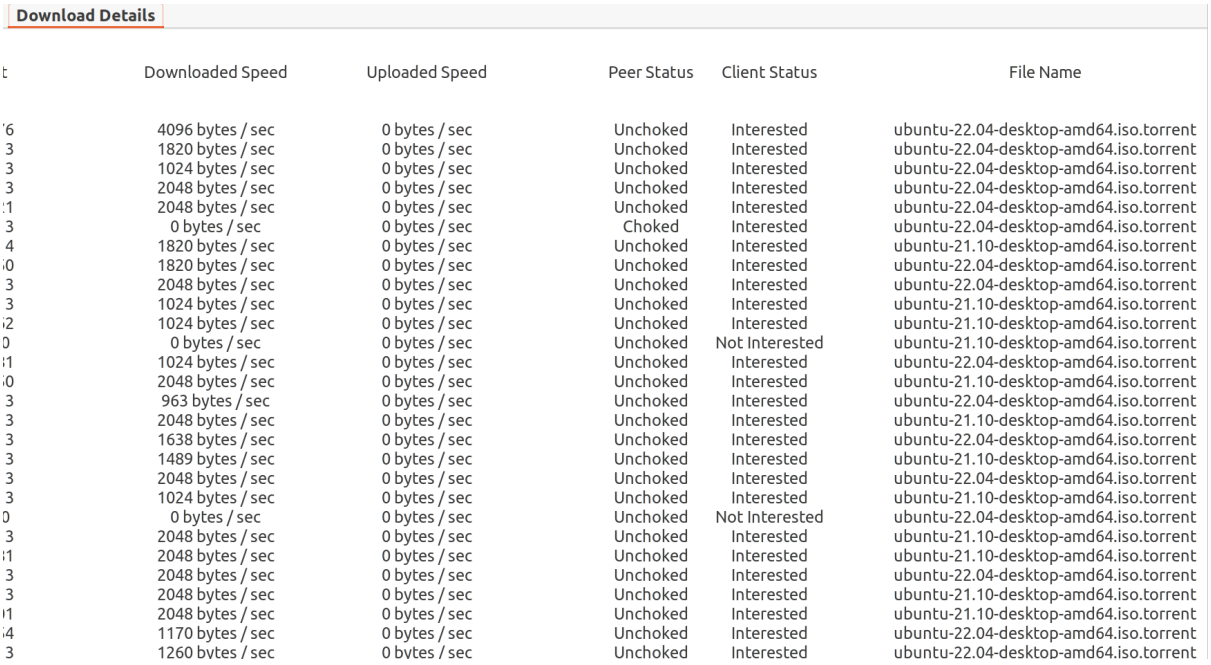



Imagen 9 - estadística de descargas

En las imágenes de la ventana ‘Download Details’ se muestran las **estadísticas de descargas.** La cual cuenta con todas las estadísticas de las descargas en curso, incluyendo: lista de Peers con ID, IP y puerto de cada una de las conexiones activas, velocidad de bajada y subida de cada conexión, estado del Peer (choked / unchoked, interested/not interested) y estado del cliente en el Peer (choked/unchoked, interested/not interested).
Aclaración: La columna Upload Speed se muestra en 0 debido a que en el momento de la captura de la Imagen ningún Peers nos estaba pidiendo piezas.



Download Details

| t | Downloaded Speed | Uploaded Speed | Peer Status | Client Status | File Name |
|---|---|---|---|---|---|
| '6 | 4096 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1820 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1024 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| :1 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 0 bytes / sec | 0 bytes / sec | Choked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 4 | 1820 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :0 | 1820 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1024 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :2 | 1024 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 0 | 0 bytes / sec | 0 bytes / sec | Unchoked | Not Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :1 | 1024 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| :0 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 3 | 963 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 3 | 1638 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1489 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1024 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 0 | 0 bytes / sec | 0 bytes / sec | Unchoked | Not Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :1 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :1 | 2048 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-21.10-desktop-amd64.iso.torrent |
| :4 | 1170 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |
| 3 | 1260 bytes / sec | 0 bytes / sec | Unchoked | Interested | ubuntu-22.04-desktop-amd64.iso.torrent |

Imagen 10 - detalle de descargas

# Logger

Otra entidad importante es el Logger, que nos ayudará a registrar las actualizaciones que vayan ocurriendo durante la ejecución del cliente.

El cliente crea al logger y este a su vez crea un archivo .txt donde se guardarán los mensajes a loggear.

La entidad Logger comenzará a escuchar los mensajes que se reciban

En este ejemplo, cuando se logra establecer una conexión con un peer, PeerConnection envía el mensaje “Peer Connection Established” mediante el channel que conecta a este y el Logger.
El logger lo recibe y luego lo escribe en el archivo.
Si hay muchos archivos, la descarga concurrente de estos crea un logger por cada uno.

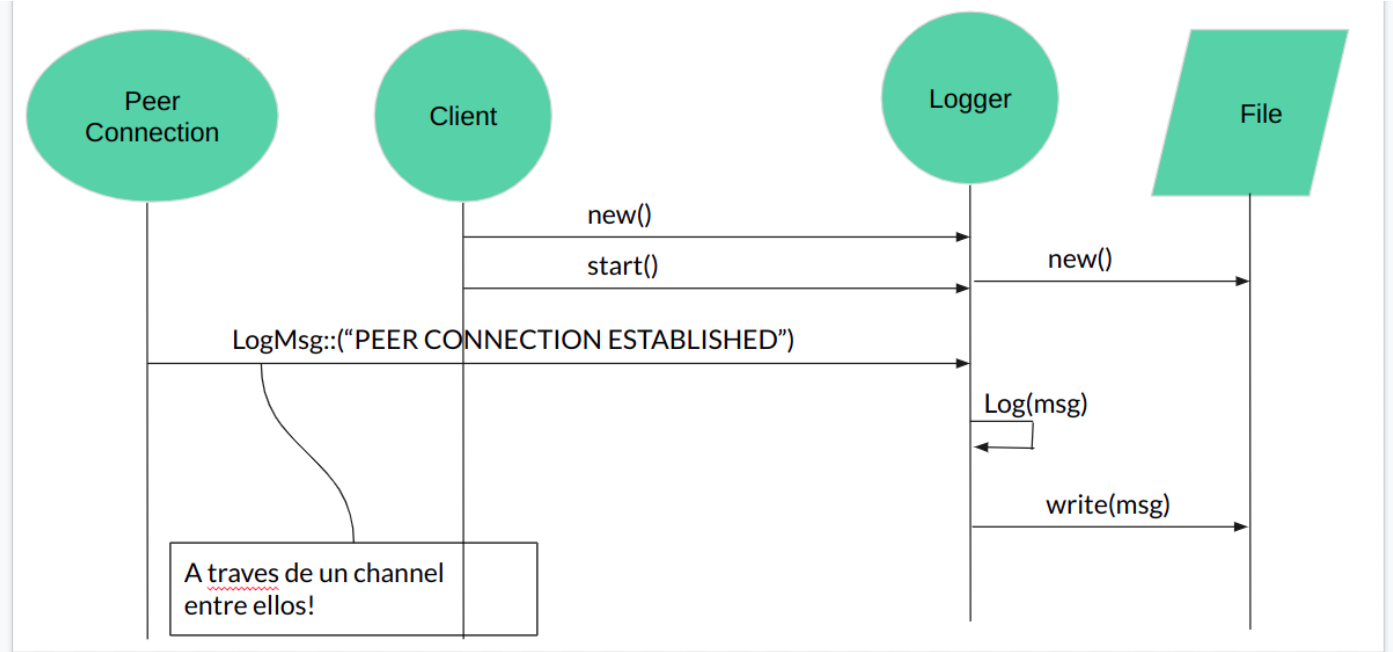



Imagen 11 - Logger

# Conclusiones

El desarrollo de este proyecto nos supuso grandes desafíos y un mayor aprendizaje al resolverlos.

Al intentar conectar con el '*Tracker*', descubrimos cómo realizar conexiones y peticiones '*HTTP*' y la importancia de hacerlas con una capa de seguridad, en este caso '*TLS*'.

Si bien hasta la entrega intermedia no habíamos implementado el multithreading, una vez implementado el mismo es evidente la diferencia entre las velocidades de descargas entre una sin '*multithreading*' y otra con.
Antes de la entrega intermedia, descargábamos las piezas del archivo una por una, luego la estrategia multithreading nos permitió pedirle varias piezas a varios '*peers*' en simultáneo, logrando así una descarga más rápida del archivo en su totalidad.

Para realizar la interfaz de usuario notamos que lo más conveniente era correr la interfaz y el cliente por hilos separados, conectados estos dos por un '*channel*'. De esta manera, al actualizar alguna información del cliente, se le envía por el '*channel*' a la UI un código que identifica el objeto a actualizar y el valor con el que actualizar. La interfaz lo reconoce y se modifica en consecuencia.
Para lograr una comunicación eficaz entre el cliente y la UI hemos utilizado los '*channels*' especiales de '*GTK*'.

# Bibliografía y Referencias


- Especificaciones de parámetros del .torrent: https://wiki.theory.org/BitTorrentSpecification
- BitTorrent Developers: https://www.bittorrent.org/
- Ubuntu .torrents: https://torrent.ubuntu.com/tracker_index
- Rust Book: https://doc.rust-lang.org/book/
- Rust Crates: https://crates.io/crates/
- Gtk: https://github.com/gtk-rs/gtk3-rs.git
- Glib: https://github.com/gtk-rs/gtk-rs-core.git