FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Taller de Programación
Proyecto Tracker 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

Tracker de BitTorrent

También llamado rastreador del protocolo de BitTorrent, es un servidor que almacena datos necesarios para matchear los peers que contengan piezas de un .torrent en específico.

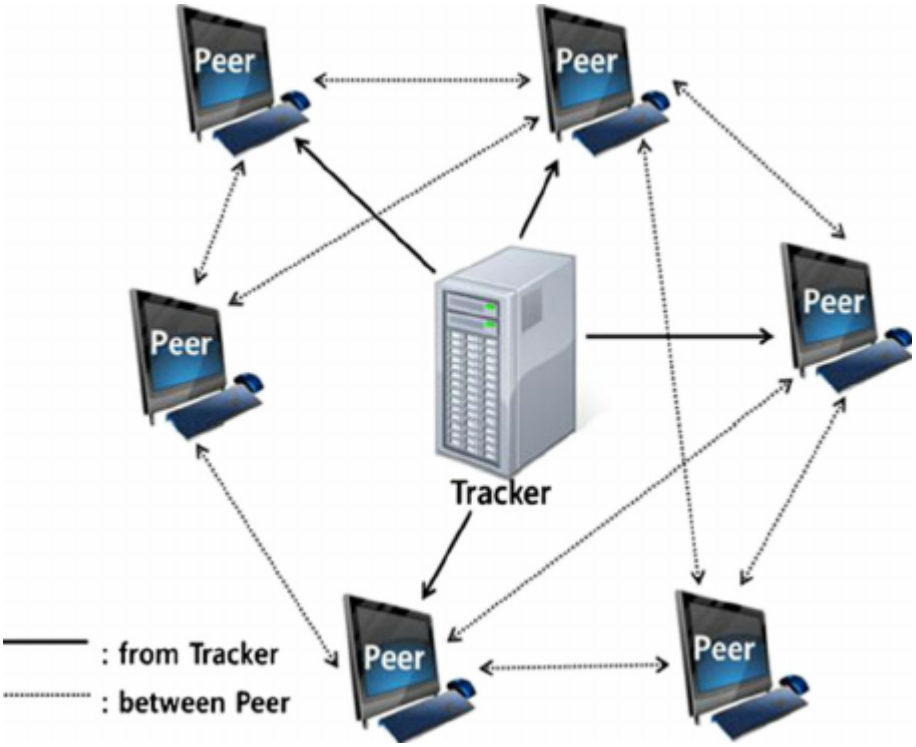

Imagen 1 - Esquema Tracker

El mismo es el punto de inicio para comenzar la descarga de un archivo ya que es la primera conexión por protocolo HTTP a realizar para obtener la lista de nodos que contienen piezas del .torrent. Esto se realiza a través del endpoint */announce*, con ciertos parámetros para identificar el .torrent en cuestión.
Por otro lado, no sólo brinda la lista de peers, sino guarda información de los nuevos nodos que descargan piezas y así obtiene ciertas estadísticas, tanto del .torrent como de los peers y sus comportamientos.
Cabe destacar que la información de la respuesta del tracker está codificada con el formato Bencoding.

Herramientas

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

- *Rust 1.61.0*
- *Cargo 1.61.0*
- *Dependencias:*
    - *serde = 1.0.100*
    - *serde_json = 1.0.40*
    - *chrono = 0.4.19*
    - *sha1 = 0.10.1*
- *HTML5*
- *CSS3*
- *JavaScript*

Arquitectura

Comportamiento general y sus componentes principales

Para la implementación de este Tracker '*BitTorrent*' comenzamos por el Data Manager, el cual inicializa al Tracker con la información correspondiente mediante la lectura de un archivo .json, el cual contiene toda la información de la última sesión.
Luego se inicia un ciclo en el Data Manager, en el cual se guardará el estado del Tracker en el .json cada 10 segundos. Almacenando en disco así cualquier nuevo cambio que haya en este.
El próximo paso es iniciar el Listener que escuchará a través de los diferentes endpoints las peticiones HTTP. Si es la primera vez que el Peer hace announce, se guardará la información que este envie, pero si no es la primera vez se actualizará la información necesaria.
Todos los datos y estadísticas de conexiones, peers y completitud de archivos se guardarán para ser luego utilizados en los gráficos de estadísticas.
Una vez finalizado este proceso de guardado de información, el Tracker le devuelve al peer la lista de peers integrantes de ese torrent, y los valores de interval, complete e incomplete.

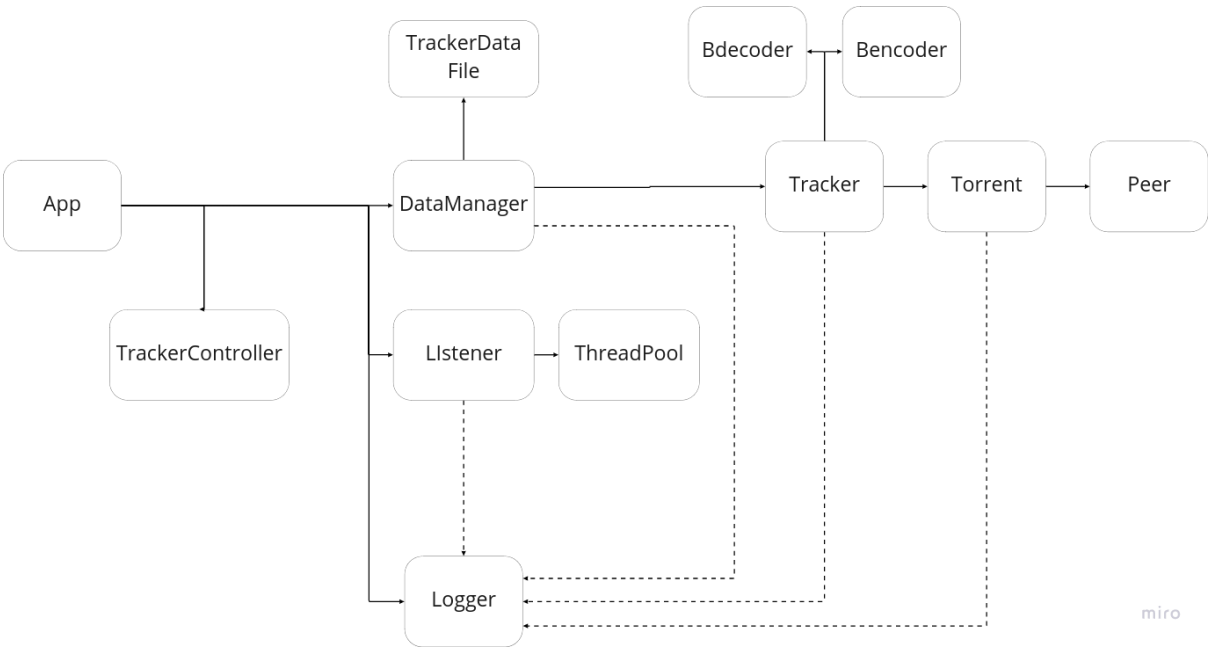

Imagen 2 - Gráfico General

Comunicación con los peers que efectúan el announce

Los peers se comunican con nuestro *Tracker* mediante peticiones que siguen el protocolo HTTP a través del endpoint */announce*. Es nuestro *Listener* quien está constantemente escuchando por estas conexiones, decodifica su contenido mediante la función *parse_announce* y guarda toda la información provista en un diccionario.

La información que los peers envían mediante el announce es:

- **info_hash**: identificador del archivo torrent.
- **peer_id**
- **port**
- **downloaded**: cantidad total descargada.
- **uploaded**: la cantidad total cargada.
- **left**: El número de bytes que este cliente todavía tiene que descargar.
- **compact**: Establecer esto en 1 indica que el cliente acepta una respuesta compacta.
- **no_peer_id**
- **event**: si se especifica, debe ser iniciado, completado o detenido.
- **ip**
- **numwant**: cantidad de peers que se devolverán en la respuesta.
- **key**
- **trackerid**

La respuesta que obtendrá el peer del *Tracker* es:

- Una lista con los Peers pertenecientes a ese torrent.
- El dato interval que son los segundos que el cliente debe esperar entre peticiones.
- Complete: número de peers con el archivo descargado completamente.
- Incomplete: número de peers con el archivo no descargado completamente.

MultiThreading

Para el inicio del programa se inicializa el *DataManager*, el *Listener* y el *Logger* en threads separados. Finalmente se crea el *TrackerController* recibiendo los *JoinHandle* de los threads previamente mencionados, para terminar el programa correctamente, cuando corresponda.

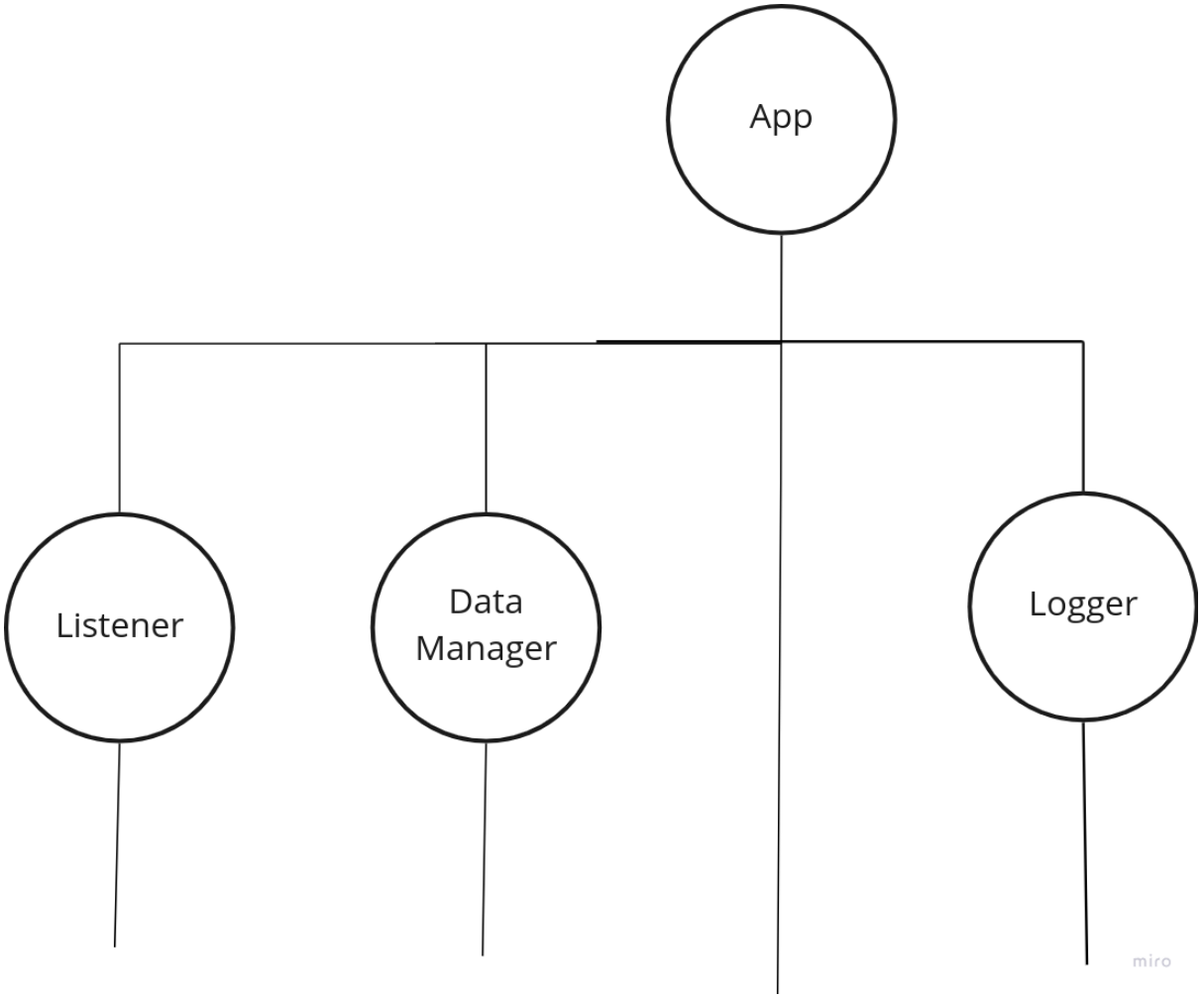

Imagen 3 - Thread Principal

Listener

Componente que hace un *bind* para conexiones entrantes, para ello invoca una *ThreadPool* de 8 threads para aceptar cada una de las solicitudes y posteriormente procesarlas con el *handle_connection()* que realizará las tareas correspondientes según el endpoint indicado,

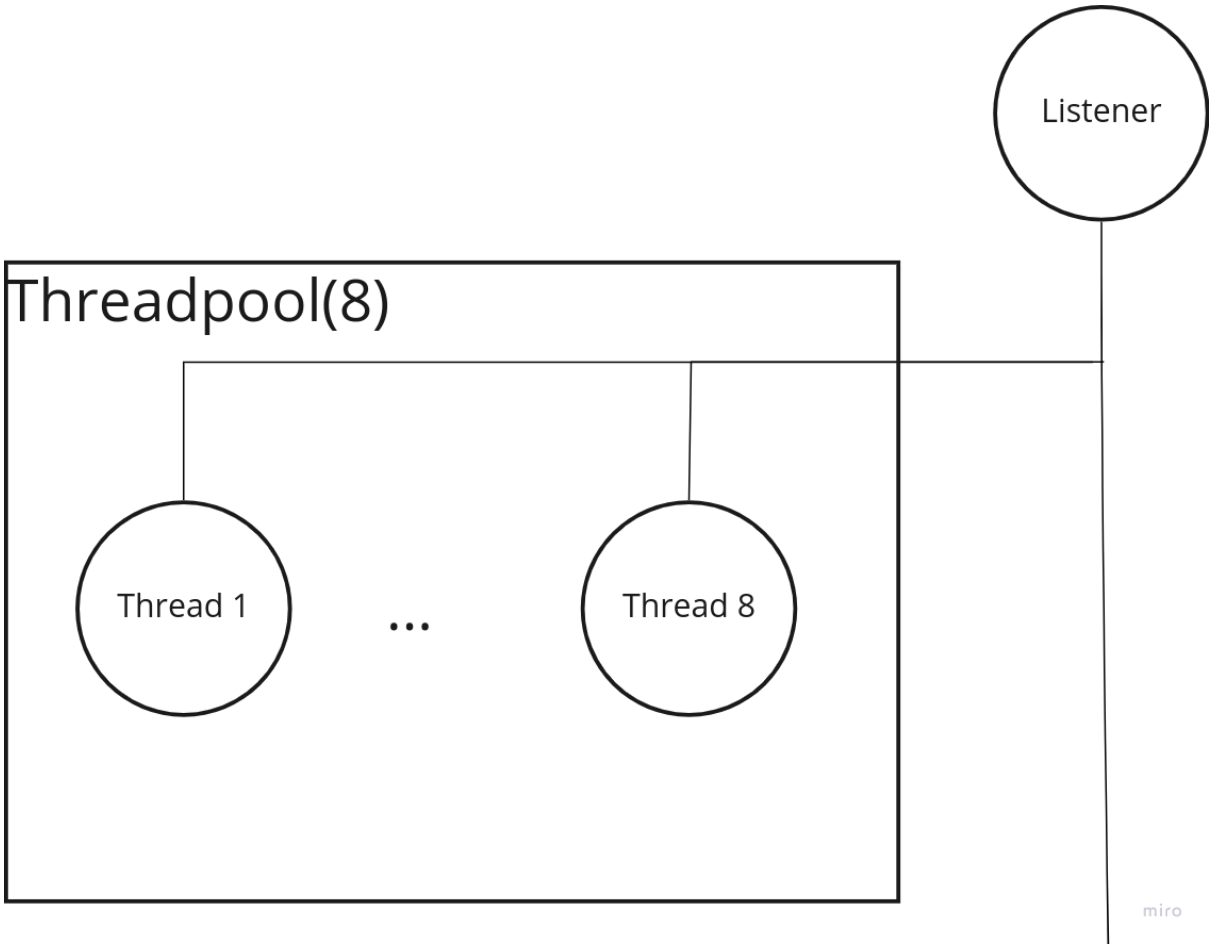

Imagen 4 - listener escuchando

DataManager

El encargado de mantener de forma persistente los datos del *Tracker* actualizados en disco, guardándolos en un *.json* para que en una futura sesión se puedan cargar toda la información sin perderla.

Logger

Otra entidad importante es el *Logger*, que nos ayudará a registrar las actualizaciones que vayan ocurriendo durante las llamadas al *Tracker*.

El *Logger* se crea durante la inicialización del programa y este a su vez crea un archivo .txt donde se van a escribir los mensajes y luego comenzará a escuchar los mensajes que se reciban.

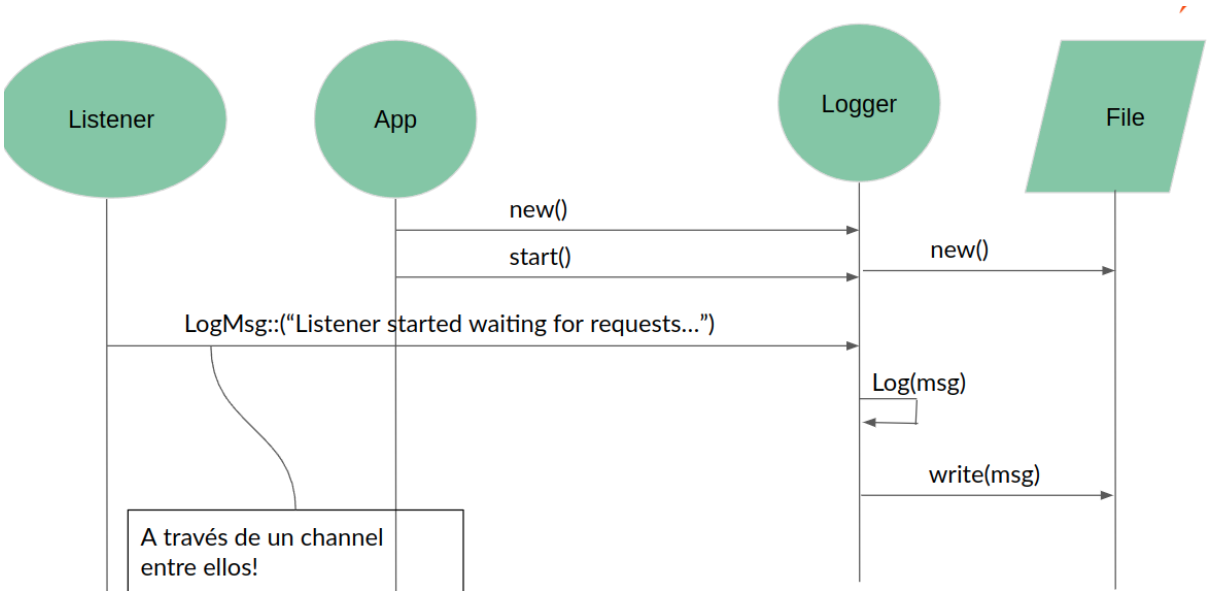

Imagen 5 - Logger

TrackerController

Es la entidad encargada de terminar el programa cuando el usuario presione Enter. Una vez se ingresa la tecla para finalizar el programa, éste joinea los *JoinHandle* recibidos en la inicialización del programa, con la ayuda de un *AtomicBool* para comunicarle a las otras entidades cuando el programa debe finalizar.

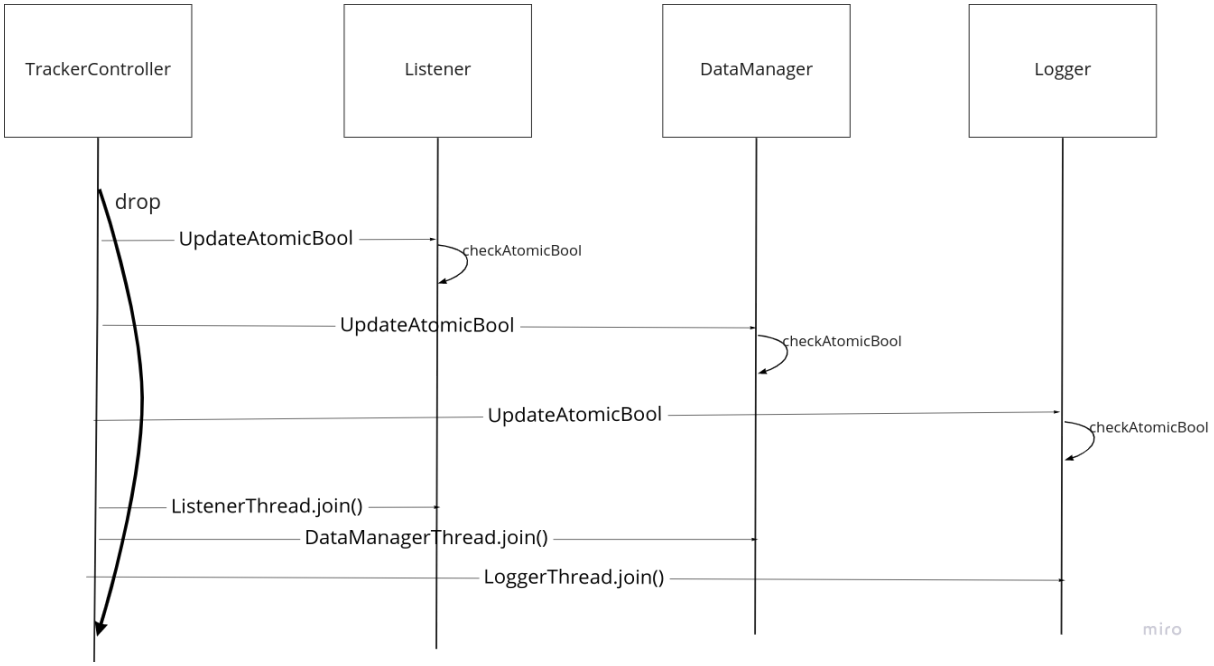

Imagen 6 - Join de threads

Tracker

El struct *Tracker* está compuesto por un diccionario que almacena todos los torrents, otro diccionario llamado *historical_peers()* que almacena la cantidad de peers conectados y la cantidad de peers con descarga completa según cada torrent y por último está compuesto por un vector llamado *historical_torrents* que contiene los tiempos en los que se agrega un nuevo torrent al *Tracker*. Estos dos últimos nos ayudarán para luego crear los gráficos mostrados en el *FrontEnd*.

A su vez, es quien recibe el *announce_dic* que vino del *Listener*, filtrando la información y encargado de actualizar las cantidades y tiempos históricos que están en *historical_torrents* e *historical_peers*.

Además detecta de que torrent es el peer a actualizar y delega en *Torrent* la actualización del mismo.

Cuando se accede al endpoint */stats*, el *Listener* le pedirá al *Tracker* toda la data histórica que recopiló hasta el momento.

Finalmente, es el encargado de devolverle la response al cliente que hizo el *announce*, delegando en *Torrent* la obtención de la lista de peers.
Al cliente se le devolverá la lista de peers que son parte de su torrent, y los valores interval, complete e incomplete.

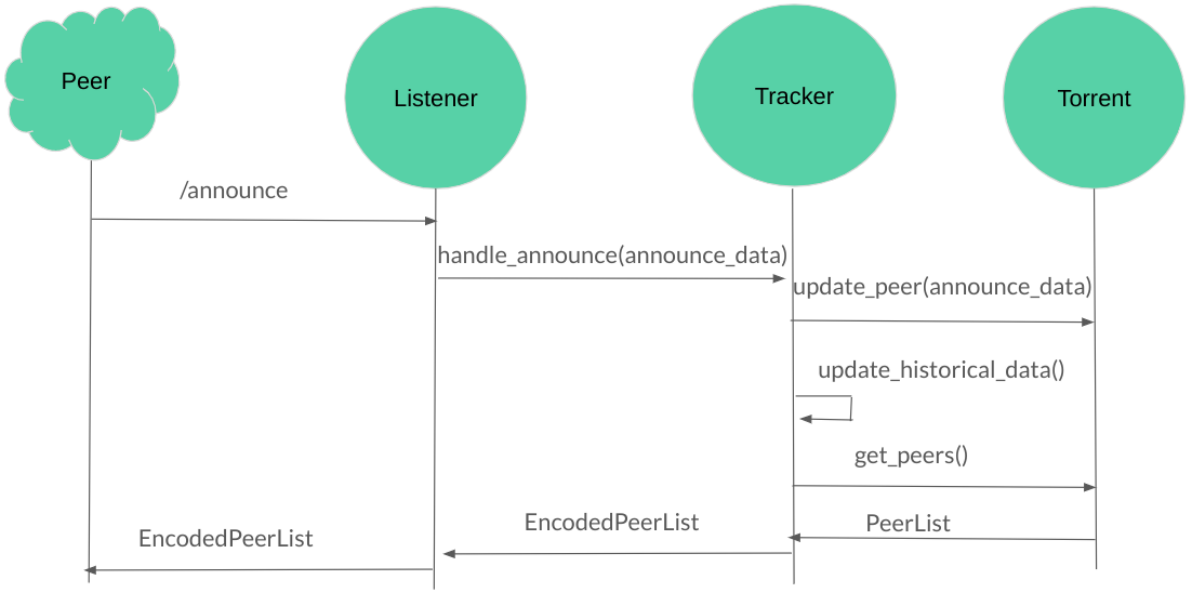

Imagen 7 - Gráfico tracker + torrent

FrontEnd

El desarrollo del *FrontEnd* requirió de cierto grado de investigación de las herramientas *HTML*, la cual nos permitió estructurar la página, *CSS*, la cual nos permitió darle estilos y *Javascript* para darle funcionalidad. En este caso optamos por usar “vanilla” javascript ya que al ser una sola vista no valía la pena usar un framework como React ya que complejizaría el proyecto de forma innecesaria.
Para los gráficos estadísticos se utilizó una librería llamada *Chart Js* la cual fue importada a través de su CDN y nos permitió simplificar la forma de mostrar las estadísticas del *Tracker*.
Además se hizo una página especial para las demás rutas (casos en que habría 404 NOT FOUND) la cual manda el mismo mensaje y además sugiere un link que te redirige a /stats.

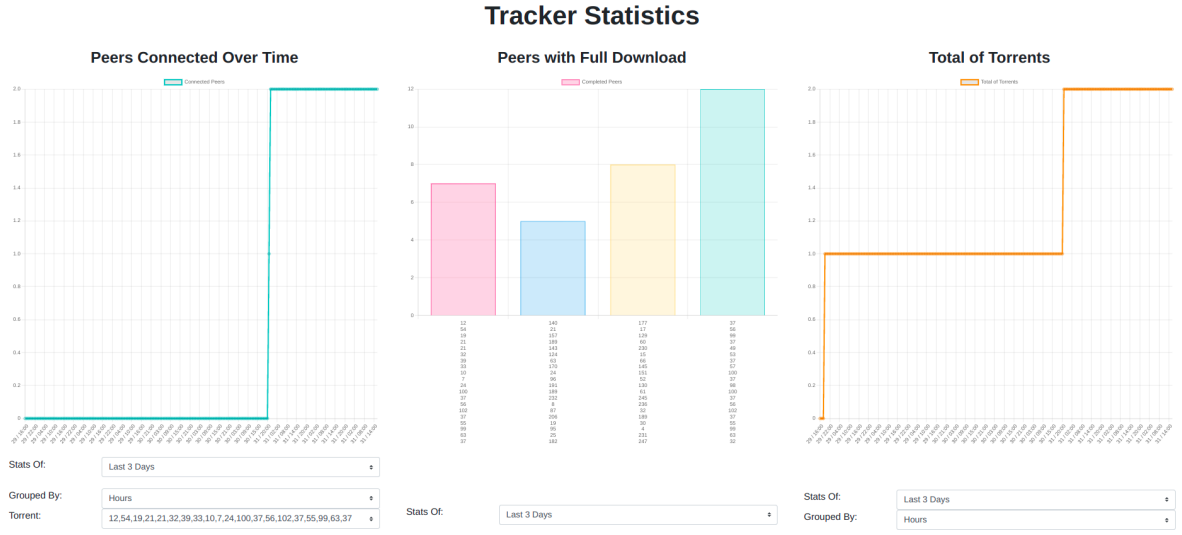

Imagen 8 - Gráfico

Conclusiones

El desarrollo de este proyecto nos permitió enfrentarnos a las problemáticas de un servidor, como por ejemplo:

- Mantener los datos de forma persistente para no perder información al prender y apagar el servidor.
- El manejo de conexiones entrantes de forma escalable para no sobrecargar el sistema.
- El manejo de la información almacenada para generar estadísticas y comprender mejor el estado de cada torrent a través del tiempo.
- La forma de mostrar las estadísticas para facilitar su compresión y posterior análisis.

Con el agregado del FrontEnd nos familiarizamos con herramientas como HTML y JavaScript, también profundizamos el uso de endpoints para conectarlo con el BackEnd. Además obtuvimos nuevo conocimiento acerca del funcionamiento de las promesas, funciones asíncronas y la librería Chart Js.

Para esta entrega final, ya contábamos con experiencia en *Rust* y nos fue más fácil la implementación de las diferentes entidades y sobre todo el manejo de los distintos threads.

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/
- Chart Js: https://www.chartjs.org/