FACULTAD DE INGENIERIA
Universidad de Buenos Aires

75.42 Taller de Programación I

Proyecto BitTorrent

Grupo Sitos:

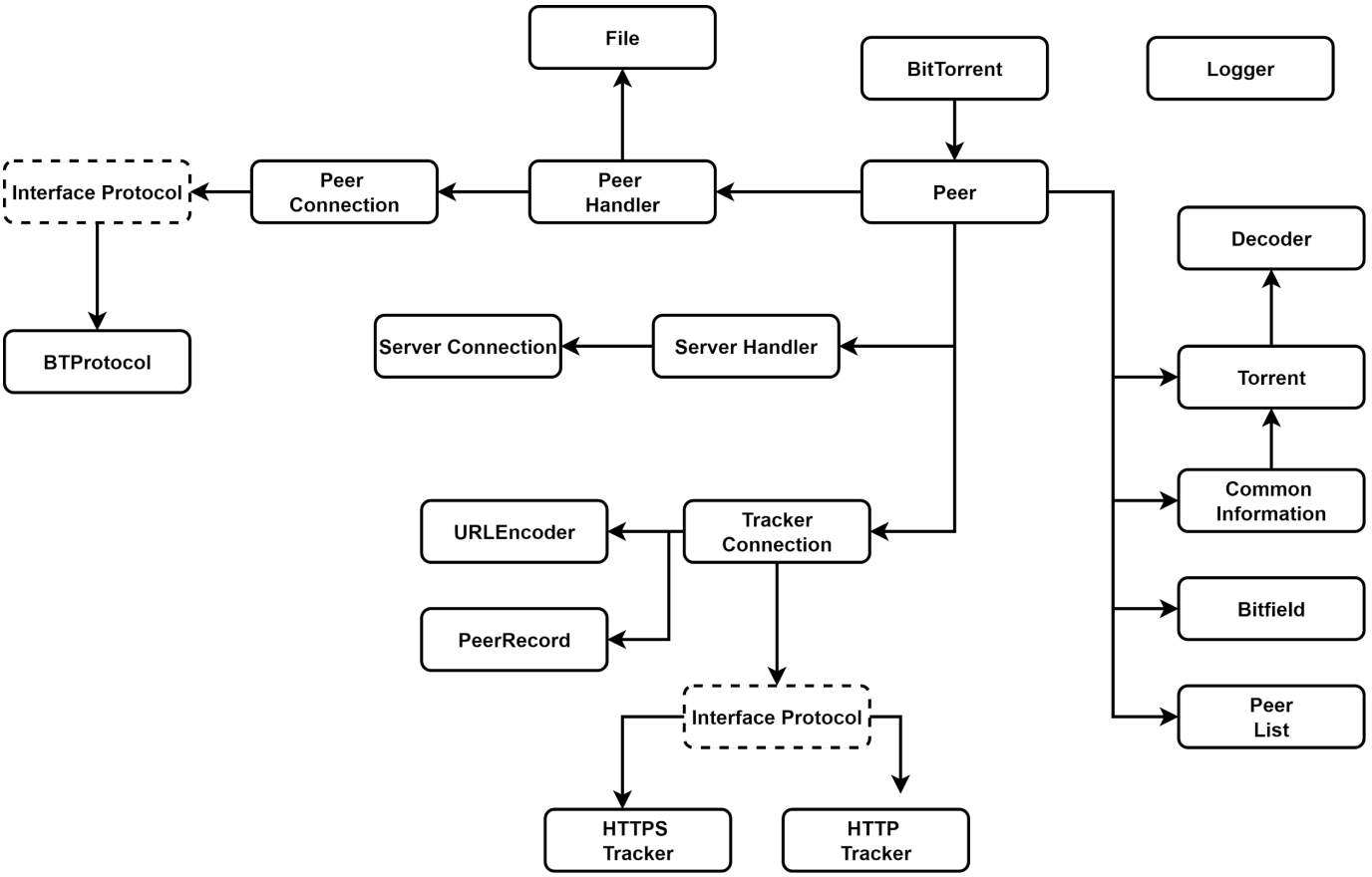

Lautaro Goijman


Sofia Carbon Posse

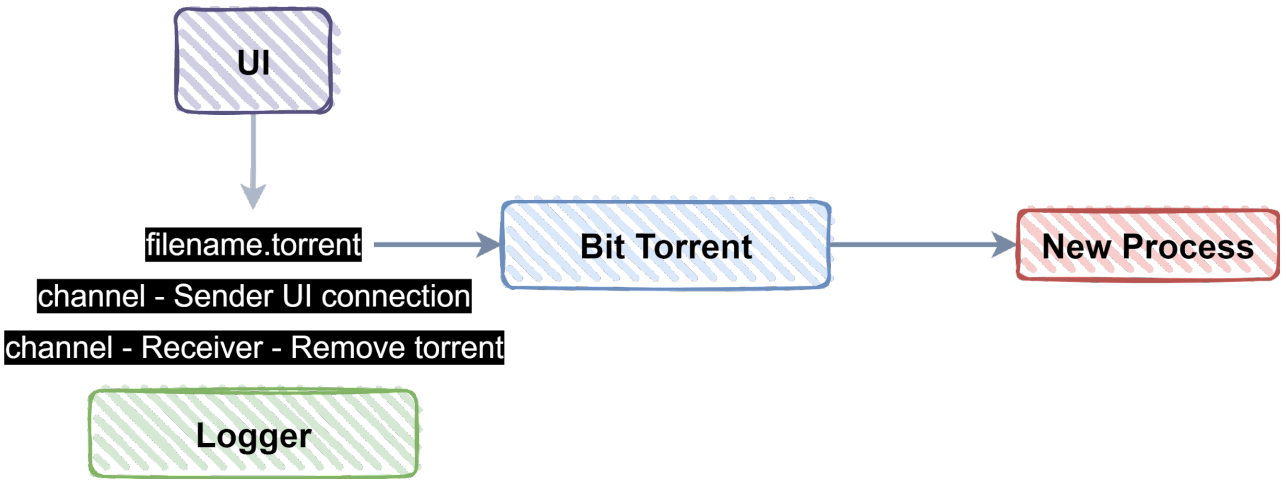

Martin Veiga


Julian Garcia

Arquitectura del Programa:



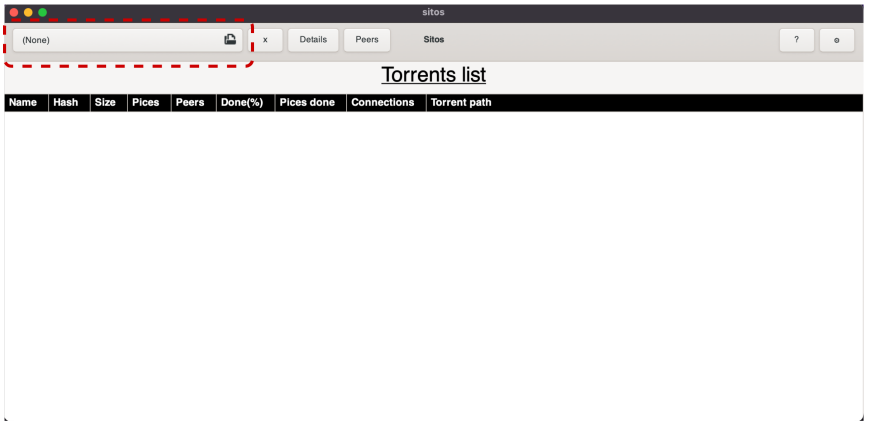
¿Sobre qué estructuras pasa el flujo de programa cuando queremos descargar un archivo?



Paso 1 - User Interface

Al inicializar el programa lo primero que podemos observar es la *User Interface.* Esta nos va a permitir seleccionar un archivo ".torrent" para poder empezar a descargarlo.

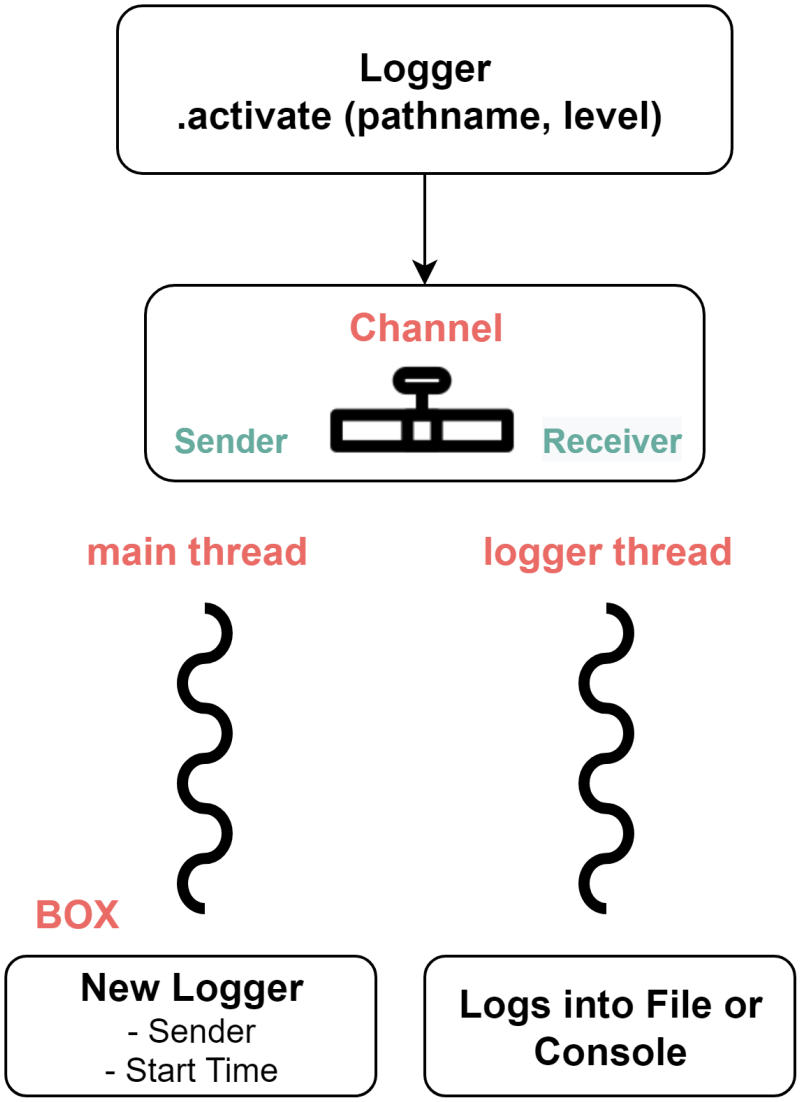
Podemos seleccionar el archivo a descargar desde el recuadro rojo punteado, como se puede observar en la imagen inferior.



El funcionamiento de la UI se puede ver en la demo, pero quedémonos con la idea de que la UI le pasa el pathname del archivo ".torrent", un channel (Sender) al programa por donde se van a actualizar los datos de UI y un channel (Receiver) para recibir una señal que nos indicará que debemos interrumpir el programa.

Paso 2 - Creación de las instancias Logger y BitTorrent

En el main del programa, primero se crea una instancia del Logger. Este nos va a permitir obtener un registro de todo lo que sucede durante la ejecución.
Implementa una función "*.activate(pathname, log_level)*", esta crea un canal de comunicación, spawnea un thread pasándole el Receiver del canal, el cual será el encargado de recibir la información que tiene que loggear.
A su vez, el thread principal continúa con la ejecución del programa instanciando el Logger que contiene el Sender del canal y un tiempo de inicio. El Sender nos va a permitir que el resto de las estructuras del programa puedan acceder al Logger para mandar la información necesaria a loggear. La instancia del Logger es un singleton, es decir, es única.



Luego, se crea una instancia de la estructura BitTorrent.

BitTorrent
processes: Vec<JoinHandle<()>>
new_process(torrent_pathname, sender, remove_rx)

BitTorrent contiene como único atributo un vector de Join Handles que nos va a permitir poder esperar a todos los threads a medida que van terminando.
El método *.new_process()* se encarga de instanciar un nuevo Peer, activarlo, y pushear su JoinHandle en el vector de procesos.

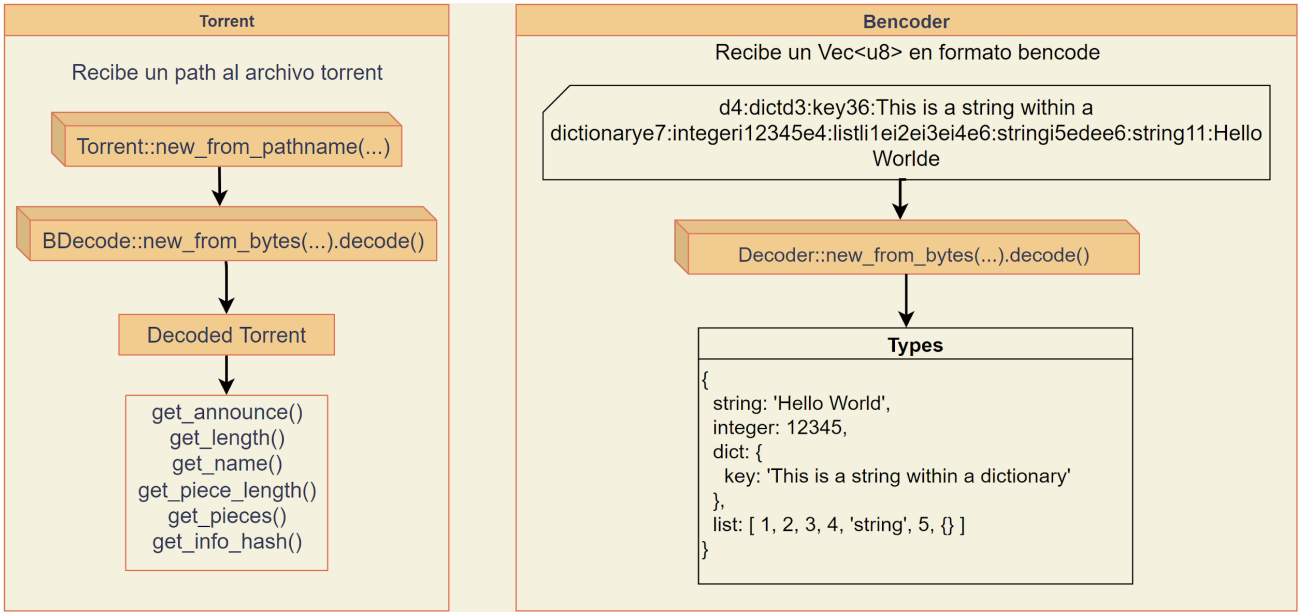
Paso 3 - Creación de Peer y su activación.

3.1 - Creación de Peer

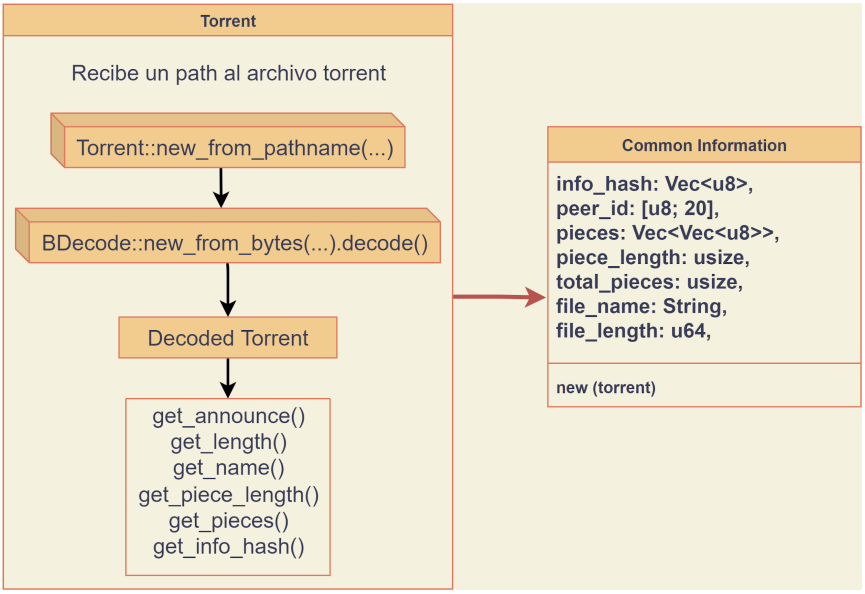
Peer
torrent: Torrent common_info: CommonInformation have: Arc<Mutex<Bitfield>> peers: Arc<Mutex<PeerList>> state: Arc<Mutex<PeerState>> handlers: Vec<JoinHandle>
new(torrent_pathname, temp_directory, download_directory, Sender) activate()

El peer contiene 6 atributos fundamentales para su funcionamiento:

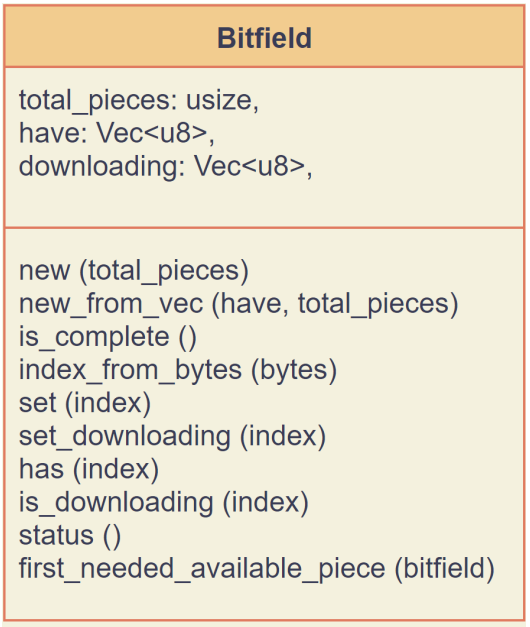
- **torrent**: este atributo contiene una instancia de *Torrent*, la cual se encarga de leer el archivo .torrent y desglosarlo para obtener toda la información necesaria. Torrent utiliza una estructura auxiliar *Bencoder*, que permite decodear el archivo ya que este viene con un formato especial.



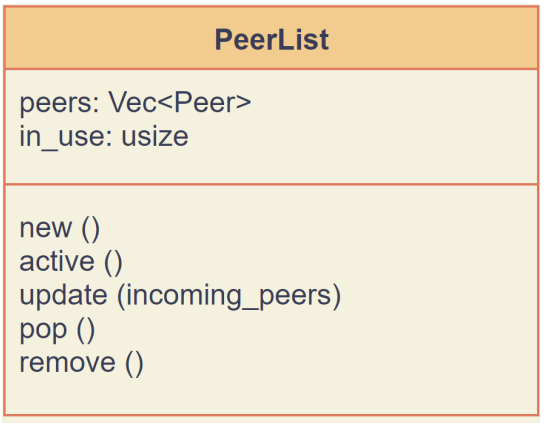
- **common_information**: este atributo contiene una instancia de *CommonInformation,* la cual contiene información necesaria que se comparte entre todos los procesos. CommonInformation recibe la instancia de Torrent mencionada arriba y obtiene toda la información de allí.



- **have**: este atributo contiene una instancia de *Bitfield* dentro de un Arc-Mutex debido a que esta información debe ser compartida entre los distintos threads. Bitfield contiene la cantidad de pieces totales que tiene el torrent y dos vectores: have, que indica con 1 las pieces obtenidas y con 0 las pieces faltantes análogamente pasa lo mismo con downloading, este indica con 1 las pieces que se están descargando y con 0 las pieces que no están siendo descargadas.



- **peers:** este atributo contiene una instancia de *PeerList*. Esta estructura es muy sencilla, contiene un vector con todos los peers disponibles que nos devuelve el Tracker.

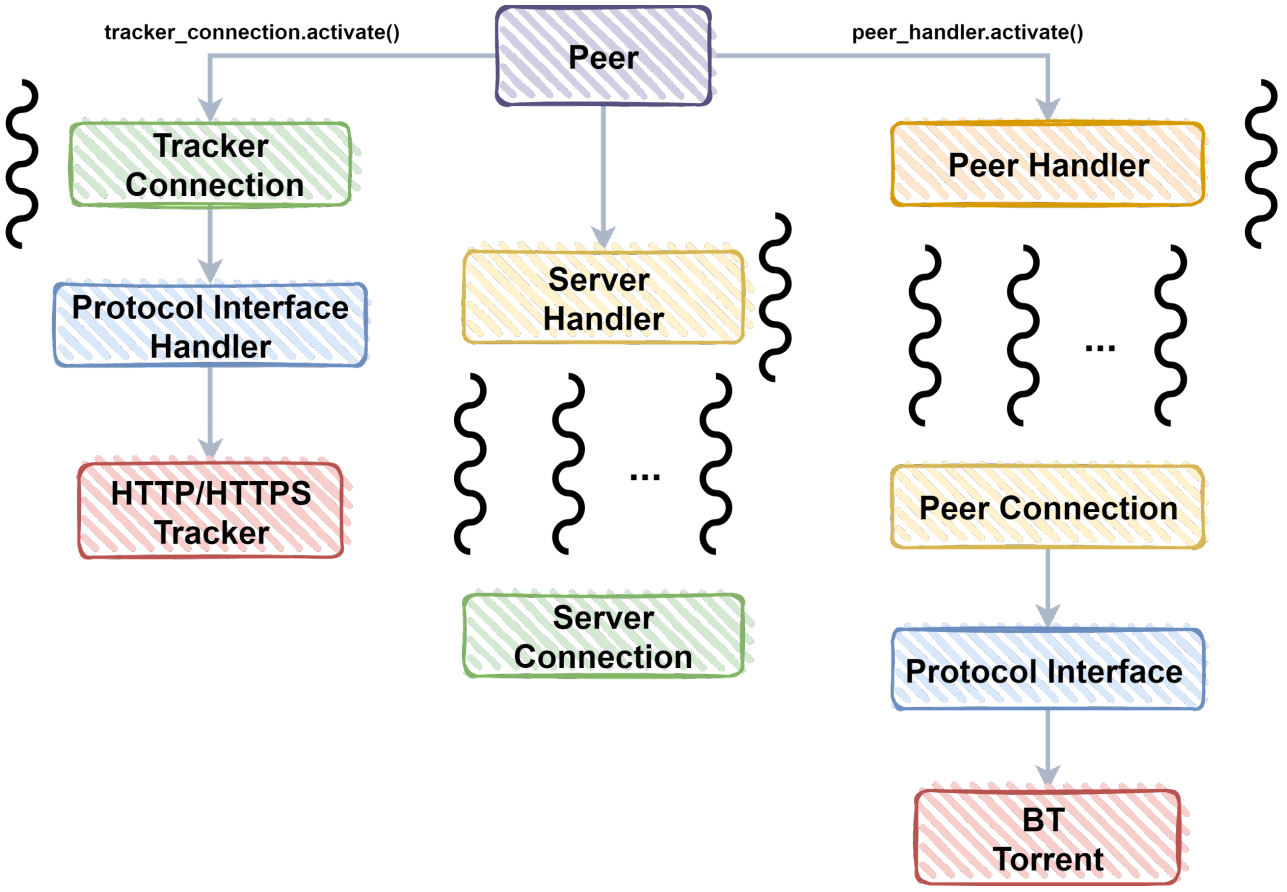


- **state:** este atributo es un enumerate, contiene 3 estados en el cual se puede encontrar el Peer: NoPieces, SomePieces y All Pieces. El peer puede pasar de un estado a otro mediante la función upgrade().

- **handlers:** este atributo al igual que en BitTorrent, contiene un vector de Join Handles que nos va a permitir poder esperar a todos los threads a medida que van terminando.

3.2 - Activar el Peer

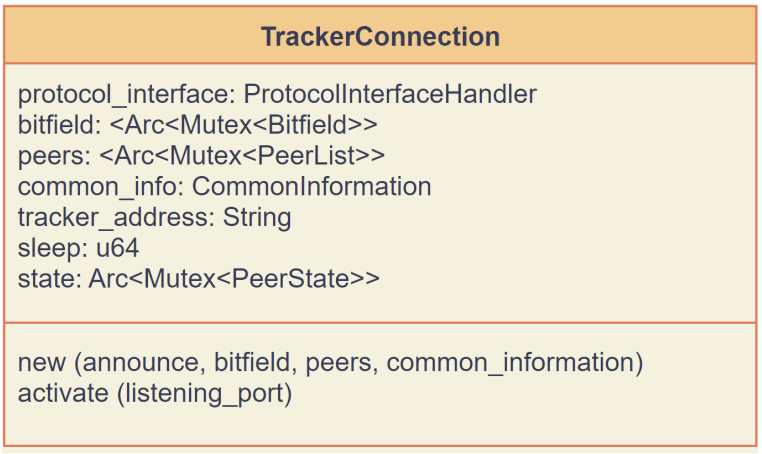
Ahora que ya sabemos que contiene cada atributo de nuestra estructura Peer veamos qué sucede cuando lo **activamos**:



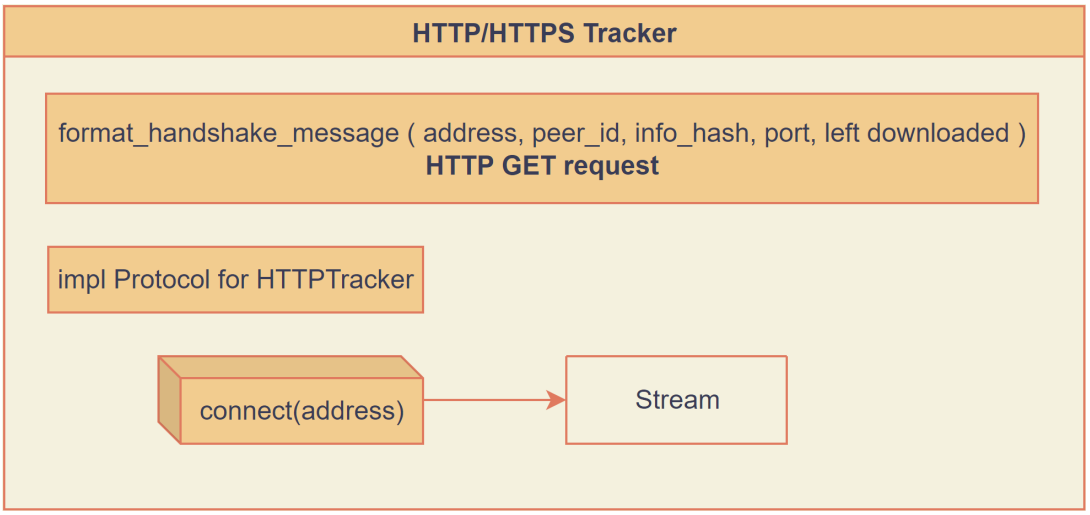
En este diagrama podemos observar que hay **3 flujos** distintos ocurriendo al mismo tiempo al mismo tiempo :

Conexión con el Tracker

Una de las actividades que sucede al activar el Peer es crear una instancia de *TrackerConnection* y activarla.



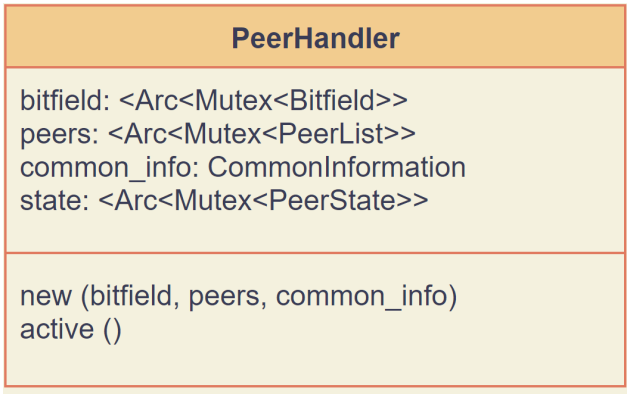
Esta estructura, como bien dice su nombre, es la encargada de conectarse con el Tracker. Cabe destacar que uno de los atributos que contiene, es el protocol_interface, este crea una instancia del enumerate *ProtocolInterfaceHandler*, el cual se va a encargar de manejar el request al Tracker dependiendo del tipo de protocolo (HTTP o HTTPS). El resto de los atributos ya los mencionamos previamente.



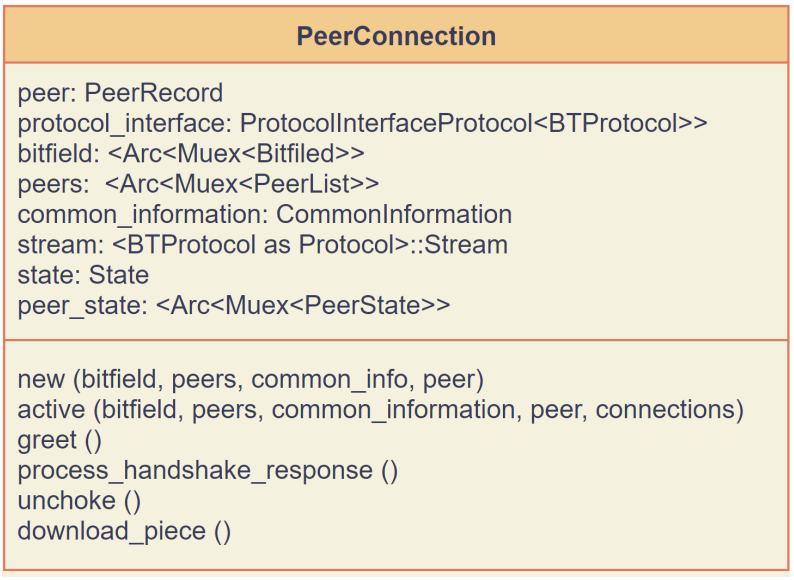
Luego, por cada peer obtenido se crea una instancia de *PeerRecord* que almacena la información importante de cada peer y actualiza la PeerList.

Conexión con Peer Handler

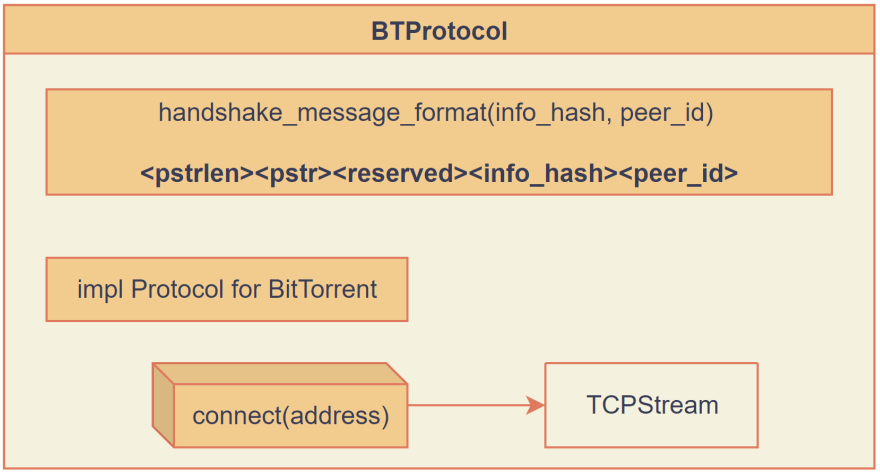
Otra de las actividades que sucede al activar el Peer es crear una instancia de *PeerHandler* y activarla.



Nuevamente conocemos el funcionamiento de todos los atributos de PeerHandler.
Al activar PeerHandler comenzamos a establecer conexiones con los distintos peers disponibles para poder bajar las distintas pieces.



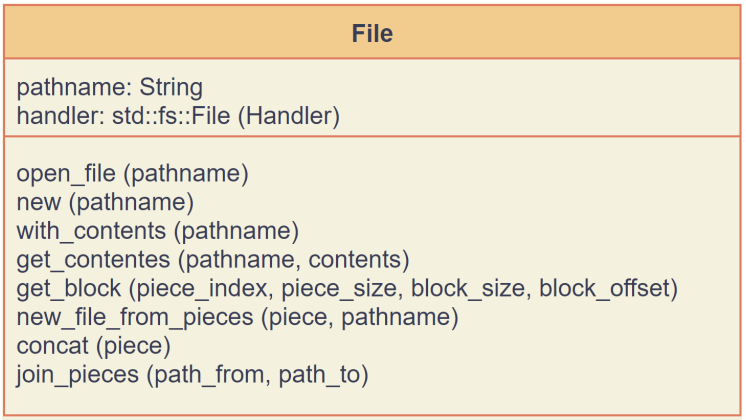
Estas conexiones las realizamos a través de *PeerConnection.*Esta es la encargada de establecer los distintos mensajes disponibles con cada peer.
*Luego,*al igual que el TrackerConnection, tenemos el protocol_interface que se encarga de manejar el request con los distintos peers a través del protocolo *BTProtocol*.



A medida que se establecen las conexiones con los distintos peers se va actualizando constantemente el StatePeer, Bitfield y se van descargando las distintas pieces a medida que las vamos obteniendo.

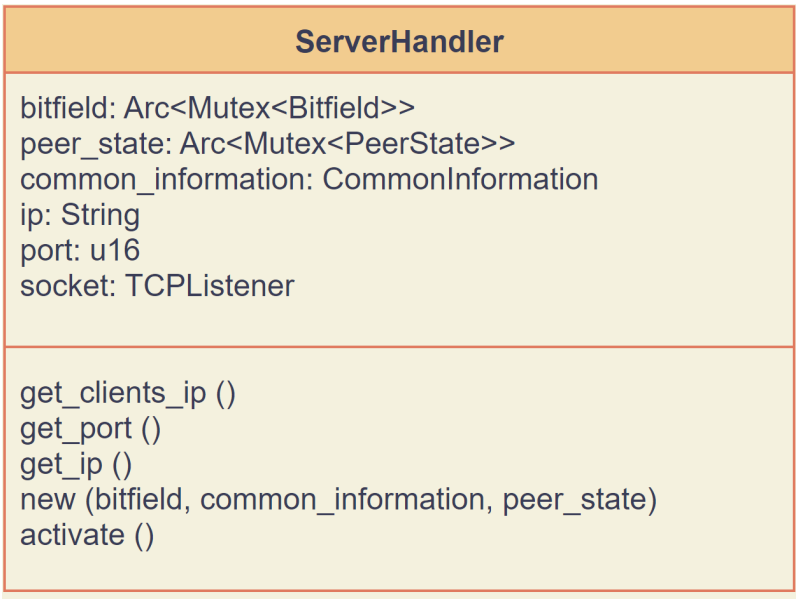
Dado el momento en el que nos encontramos con el Bitfield completo, procedemos a concatenar todas las pieces y obtener el archivo.

Para poder lograr unir las pieces y obtener el file completo utilizamos el struct *File*

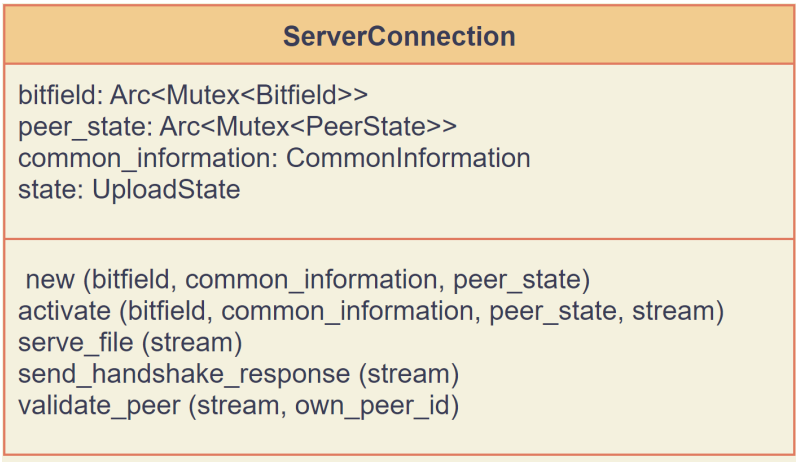


Conexión con Server Handler

Finalmente la última actividad que sucede al activar el Peer es crear una instancia de *ServerHandler* y activarla. Esta se comporta de una manera muy similar a PeerHandler.



ServerHandler es la encargada de unir a nuestra dirección de ip un listener que va a captar las conexiones entrantes. Cada vez que se encuentra con una conexión correcta, spawnea un thread para poder realizar la conexión, esto lo hacemos mediante ServerConnection.



Nuevamente análogo al PeerConnection, instanciarla nos permite abrir nuestros puertos para que otros peers puedan conectarse a nosotros y descargar las pieces que necesiten.
De esta manera realizamos el intercambio de mensajes pero de manera inversa a PeerConnection.

En conclusión, con todas estas estructuras mencionadas previamente y los distintos flujos, pudimos lograr crear un BitTorrent capaz de descargar y subir archivos entre múltiples dispositivos.