BitTorrent Client
Taller de programacion I
Primer Cuatrimestre 2022

July 11, 2022

Integrante	Padrón	Correo electrónico
Marczewski Santiago	106404	smarczewski@fi.uba.ar
Montecalvo Ignacio	105555	imontecalvo@fi.uba.ar
Arillo Luis Anibal	56078	larillo@fi.uba.ar
Sabao Tomás	99437	tsabao@fi.uba.ar

Contents

1 Introducción

El objetivo del presente informe es presentar en detalle el desarrollo del proyecto de BitTorrent Client, recorriendo los puntos mas importantes con el fin de facilitar el entendimiento de dicho proyecto y comprender ciertas decisiones tomadas durante el mismo.

2 Ejecución del programa

2.1 Archivos requeridos

Para la ejecución del programa se debe contar con:

- Un archivo .torrent o un directorio con archivos .torrent
- Un archivo de configuracion especificando:
  - Puerto utilizado para la conexion
  - Path valido donde se desean guardar los logs
  - Path valido donde se desean guardar los archivos descargados

2.2 Comando de ejecución

Comando de ejecución

```
 cargo run [ruta al archivo .torrent/directorio] [ruta al
archivo de configuración]
```

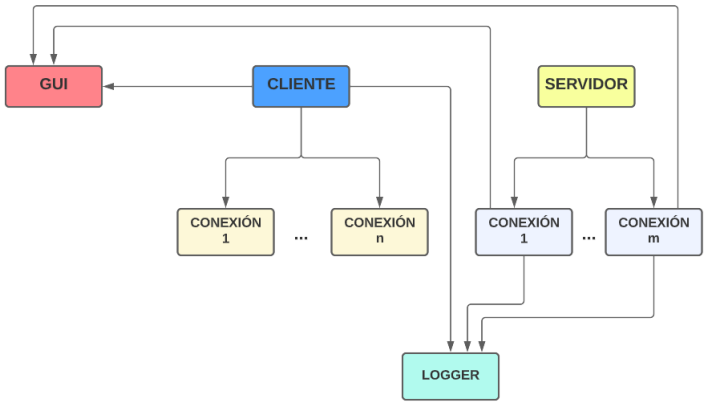
3 Información general

3.1 Flujo general del programa

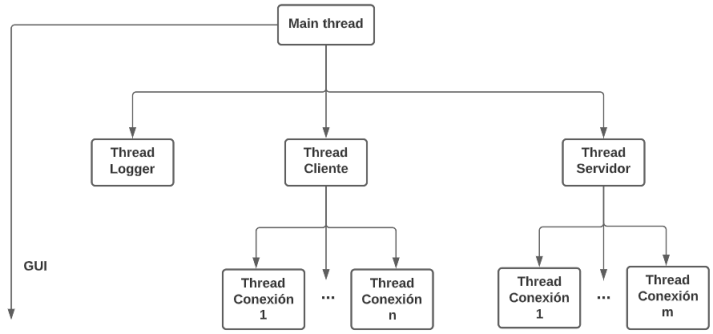
- Parseo de la configuración.
- Escaneo de torrents (parsear torrents y detectar si ya hay piezas descargadas)
- Ejecución del servidor
- Ejecución de el/los clientes (se intancia uno por cada torrent, como máximo pueden haber tres clientes en paralelo).
- Logging de eventos
- Interfaz gráfica

A continuación se presentarán una serie de diagramas generales con el fin de dar un primer acercamiento al modelo creado.

3.2 Diagrama de Interacción general



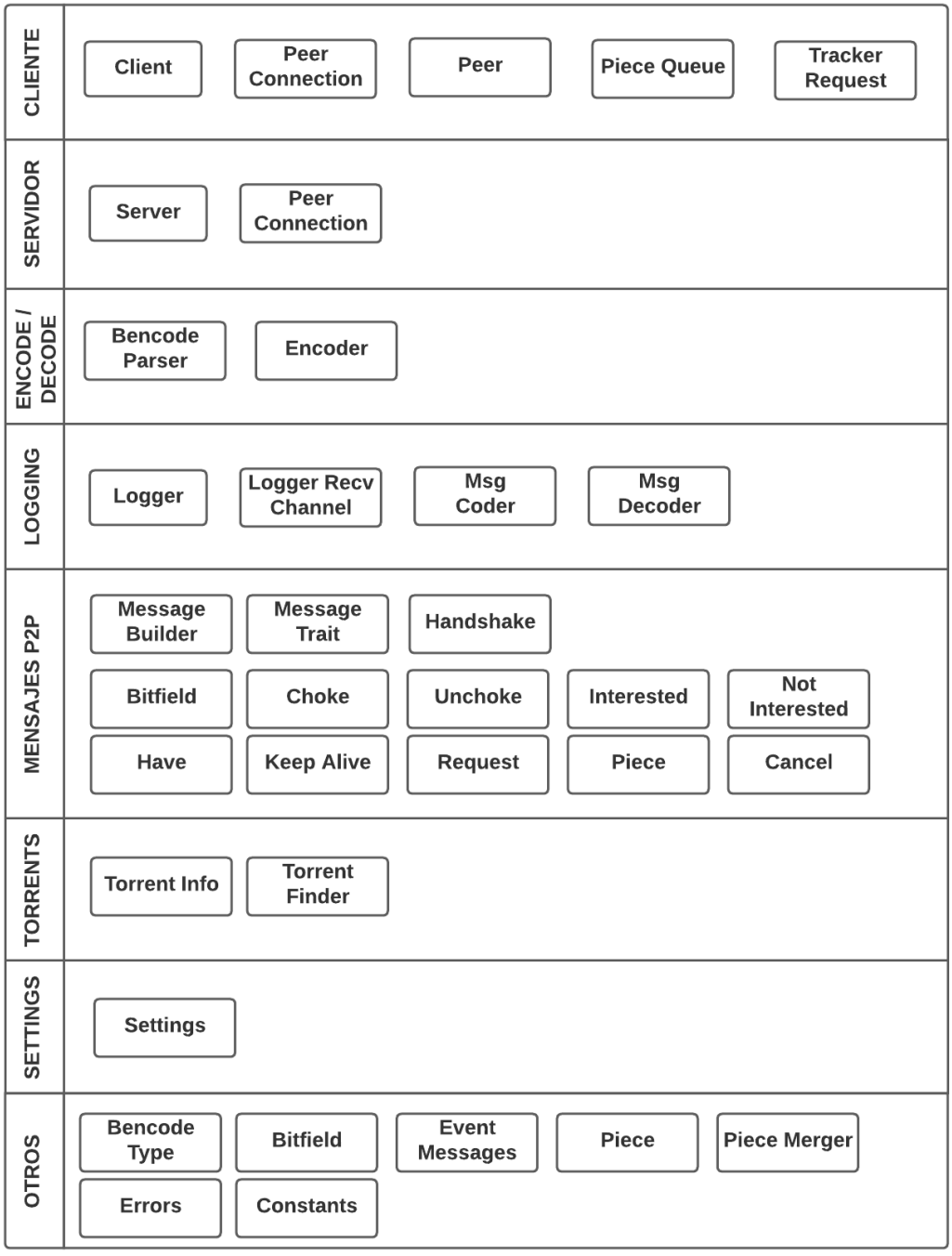
3.3 Diagrama de Threads



En el hilo principal se encuentra la GUI. A su vez, a partir de este se crean los siguientes hilos:

- Logger: Se encuentra en un thread a parte recibiendo mensajes y loggeando eventos.
- Cliente: Se crea un thread de cliente por cada torrent a descargar. Se limita el número de threads a tres, es decir, como máximo se pueden descargar tres torrents en paralelo. A su vez, por cada thread del cliente se lanza un thread por cada peer que podemos establecer conexión.
- Servidor: A diferencia del cliente, existe un único thread para el servidor. A partir de este thread, también se lanza un thread por cada conexión que recibimos.

3.4 Módulos del programa



A continuación, una breve descripción de las partes más importantes:

- Cliente:
  - Client: Contiene toda la lógica del cliente. Aquí se encuentra la creación, inicialización y ejecución de un cliente el cual se encarga de la descarga de un torrent.
  - Peer Connection: Abstracción utilizada para representar una conexión con un peer. Se encarga de establecer la conexión y del intercambio de mensajes resultando en la descarga de una pieza.
  - Peer: Representa un peer.
  - Piece Queue: Abstracción utilizada para representar una cola con las piezas pendientes por descargar.
  - Tracker Request: Se maneja la conexión con el tracker, envío de la petición y posterior parseo de la misma.
- Servidor:
  - Server: Contiene toda la lógica del servidor. Aquí se encuentra la creación, inicialización y ejecución de un servidor, el cual estará escuchando conexiones y manejándolas.
  - Peer Connection: Abstracción utilizada para representar una conexión con un peer. Se encarga de realizar el intercambio de mensajes, y por lo tanto de servir las piezas solicitadas.
- Logging:
  - Logger: Se encarga de loggear los distintos eventos en un archivo de texto.
  - Logger Recv Channel: Abstracción que actúa de canal, utilizado para loggear eventos.
  - MsgCoder: Codifica un mensaje que será enviado al logger.
  - MsgDecoder: Decodifica un mensaje que recibirá el logger.
- Torrents:
  - Torrent Info: Abstracción utilizada para representar toda la información de un archivo .torrent.
  - Torrent Finder: Dada la ruta de un archivo/directorio y la ruta de descargas, se encarga de obtener un Torrent Info por cada torrent y además, crea un bitfield de piezas descargadas para tener conocimiento si ya tenemos piezas descargadas.

4 Inicio del programa

4.1 Parseo de la configuración

Se recibe por linea de comando la ruta del archivo de configuración y posteriormente se lo parsea extrayendo los datos relevantes: Puerto, Ruta del directorio de descargas y Ruta del directorio donde se guardarán los logs.

4.2 Escaneo de archivos

En caso de recibir por linea de comando un único torrent, se lo parsea. En caso de recibir un directorio, se lo recorre y se parean todos los torrents.
Luego se recorre el directorio en el cual se van a almacenar los archivos descargados. En base a dichos archivos se actualiza un bitfield que se genera para llevar cuenta de cuales piezas se van a descargar. Esto permite tener conocimiento acerca de cuáles piezas faltan descargar y evitar descargar piezas que ya se tienen. De esa manera se puede reanudar la descarga en el caso de que esta sea interrumpida en el futuro, ya que el cliente va a buscar solo aquellas piezas faltantes en el bitfield.
Por último, es importante destacar que estos bitfields generados son compartidos por el servidor y el cliente, permitiendo que desde un comienzo el servidor sepa qué piezas puede compartir, y con el transcurso de la descarga, el cliente irá actualizando dicho bitfield de manera que el servidor (que accede al bitfield sólo en forma de lectura) sabe si tiene la pieza que le solicitaron o no.

5 Cliente

5.1 Inicialización

Se instancia un cliente por cada torrent y se encarga de la descarga total del archivo en cuestión.
Por otro lado, como máximo pueden descargarse tres torrents en paralelo, es decir, pueden exitir como mucho tres threads ejecutándo un cliente cada uno. Este número fue seleccionado de forma arbitraria, pero con el objetivo de limitar la cantidad de threads dado a que no limitamos la cantidad de conexiones con otros peers.
Para que pueda ser inicializado correctamente, cada cliente recibirá el torrent parseado (como TorrentInfo) y el bitfield de piezas (PieceBitfield) para saber cuáles son las que restan por descargar.

5.2 Comunicación con el tracker

Lo siguiente es establecer la conexión con el tracker. Para ello, hacemos una distinción según que tipo de url tenga el tracker:

- URL con https: Se utilza un TlsStream haciendo uso del crate tls-native.
- URL sin https (http, udp, etc): Se utiliza directamente el TcpStream.

Una vez realizada la conexión con el tracker se envía la petición HTTP, este proceso se realiza a través de la abstracción TrackerRequest.
Con la respuesta del tracker parseada, podemos obtener la lista de peers, la cual será utilizada para poder conectarnos a los mismos.

5.3 Conexión con peers

Una vez ya conseguida la lista de peers, la misma se recorre y se intenta establecer conexión con todos los peers presentes en ella.
Por cada una de estas conexiones se lanza un nuevo thread y se crea una instancia de la abstracción PeerConnection, la cual representa la conexión. La misma guarda el estado de dicha conexión y se encarga de toda la lógica referida a la interacción entre el cliente y el peer:

- Conectarse al peer
- Enviar y recibir handshake
- Intentar descargar piezas hasta que la descarga del torrent finalice u ocurra algún error en el transcurso.

Por último, es importante mencionar que el thread del cliente y los threads de cada conexión están conectados por un canal de tipo MPSC. Mientras la descarga sucede en los múltiples threads de las conexiones, el cliente se queda recibiendo mensajes hasta que la descarga esté lista o las conexiones activas lleguen a cero. Cada vez que el cliente sea notificado sobre la descarga de una nueva pieza, éste actualizará el bitfield de piezas descargadas.

Se tomó esta decisión para que el cliente sea el único responsable de modificar el bitfield de piezas descargadas y así evitar numerosos bloqueos, los cuales ocurrirían si esta responsabilidad la tendría cada una de las PeerConnections.

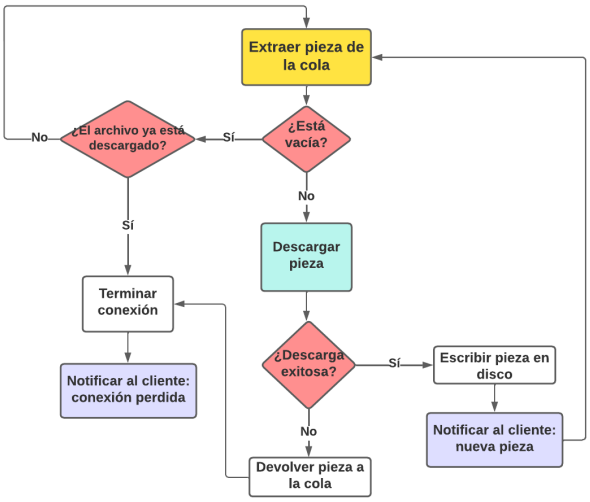
5.4 Descarga de piezas

La descarga de piezas sucede de forma simultánea (concurrente) en los threads de cada PeerConnection.
Estos threads comparten la cola con las piezas restantes, por lo que pedirán el lock e intentarán retirar una pieza de la misma.

Puede darse la situación de que un thread intenta tomar una pieza de la cola y esta se encuentra vacía. Que la cola se encuentre vacía no es indicador de que todas las piezas hayan sido descargadas, dado que puede ocurrir que algún thread que tomó una de las piezas para descargar no pueda hacerlo. Debido a eso es necesario un mecanismo de control para saber que piezas fueron correctamente descargadas. Para ello se utiliza el mismo bitfield representativo de las piezas del torrent, donde se distinguen las descargadas de las pendientes. En caso de que la cola esté vacía:

- Se chequea si la descarga finalizó fijándose si el bitfield de piezas descargadas posee todas las piezas en 1. En este caso, la conexión se termina.
- Si el peer no posee ninguna pieza de las restantes, también finalizará la conexión.
- En caso contrario, intentará nuevamente retirar una pieza de la cola llamando previamente a "yield now".

Si la cola no estaba vacía, se intenta descargar la pieza obtenida siguiendo el protocolo de BitTorrent. En caso de algún error, ya sea que la pieza es inválida, el peer nos bloqueó, no tiene la pieza o simplemente no nos contesta, la pieza extraída se devuelve a la cola de piezas pendientes. Por otro lado, si la pieza se descargó con éxito, la misma se escribe en disco y se notifica al cliente para que actualice el bitfield.



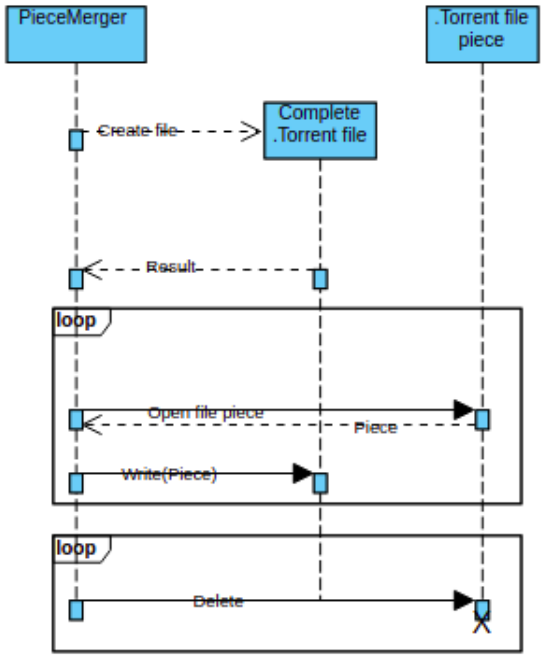
5.5 Almacenamiento de piezas

Se decidió mantener cada pieza en un vector, al cual se van agregando cada uno de los bloques que envía el peer. Una vez que se completa la pieza, se valida la misma contra el valor de sha1 que debería producir. En el caso de que sea valida, esta se escribe a un archivo y se libera el vector. Puede garantizarse entonces que el programa consumirá una cantidad de memoria (en almacenamiento de piezas) menor o igual a:

$$Memoria\ max = (\#Torrents) \times (\#of\ peers\ ) \times (Size\ biggest\ piece) \quad (1)$$

5.6 Unión de piezas

La combinación de las piezas para la generación del archivo se realiza una vez todas las piezas fueron descargadas. Se genera un archivo vacio, al cual se le va appendeando el contenido de cada pieza en orden secuencial de piezas. Al tener solo el contenido de una sola pieza en memoria, no es intensiva computacionalmente Una vez que las piezas son unidas, se eliminan las piezas descargadas



6 Servidor

6.1 Inicialización

En el caso del servidor, se instancia un único servidor el cual se encargará de compartir piezas de todos los torrents que tenga disponibles, a diferencia del cliente que se instancia uno por cada torrent.
Para su inicialización, el servidor recibe todos los torrents parseados junto con el bitfield de piezas de cada torrent, los cuales comparte con los clientes. El servidor accederá a dichos bitfield sólo en forma de lectura.

6.2 Conexiones

Posteriormente, el servidor comienza a escuchar las conexiones que lleguen y crea un thread por cada una de estas. Nuevamente, cada conexión se representa mediante la abtracción de Peer Connection (ligeramente diferente a la utilizada del lado del cliente). Para la creación de estas, cada una recibe toda la información de todos los torrents que posee el server y se queda únicamente con la información del torrent solicitado por el peer.
Cada una de estas Peer Connection, guarda el estado de la conexión, se encarga de responder el handshake y se encarga de manejar los mensajes que le lleguen, entre ellos las peticiones de piezas.

6.2.1 Servir piezas

Las piezas se sirven de a bloques mediante el mensaje de tipo Piece como respuesta a uno de tipo Request. Para ello, cada vez que se reciba la petición de un bloque de una determinada pieza se tomó la decisión de leer y cargar en memoria la pieza entera, debido a que es probable que el peer nos siga pidiendo bloques de la misma pieza y de esta manera se evita realizar múltiples accesos a disco para leer la misma pieza. En cuanto se reciba una petición de una pieza distinta a la que se tiene cargada en memoria, la misma se sobreescribirá por la pieza nueva.
Por último, resulta conveniente mencionar que se pueden presentar dos escenarios durante la lectura de una pieza, y cada uno es manejado de distinta forma:

- La pieza se encuentra en un único archivo dado a que el torrent al cual pertenece aún no fue descargado por completo. En este caso, simplemente se lee la totalidad del archivo.
- El torrent fue descargado por completo, y en consecuencia todas las piezas fueron unidas formando el archivo en cuestión. En este caso, se debe utilizar un offset para situar el puntero a la posición donde comienza la pieza que deseamos leer, y realizar la lectura a partir de allí.

7 Logging de mensajes

7.1 Generacion de archivos

Se generan dos archivos de log por ejecución del archivo, uno para logs del lado del servidor y uno para el lado cliente

7.2 Formato de logs

Los logs son generados con el siguiente formato:

Formato de logs

```
[Tipo de log] [Momento que se realizo el log] Mensaje
```

7.3 Manejo de multiples hilos

Dado que múltiples hilos pueden bajar archivos en simultaneo, se requiere de un mecanismo para que los logs se escriban en un mismo archivo. Se decidió utilizar channels para transmitir los mensajes que deben ser logeados al logger. El uso de channels evita el cuello de botella que se puede generar al compartir elementos entre varios hilos, ya que cada hilo manda el mensaje para loggear y no detiene su ejecución, siendo el encargado del otro extremo del canal el encargado de manejar dichos mensajes enviados

8 Conclusion

El presente proyecto permitió poner en práctica y afianzar conocimiento en materia de concurrencia, presentándonos múltiples desafíos a la hora de trabajar con threads y su sincronización, dandonos la posibilidad de experimentar y comprender mejor el funcionamiento de locks y channels. Además, nos aportó bastante conocimiento en todo lo que respecta al modelo cliente-servidor, uso de sockets y conexiones tcp.
Por otro lado, la realización del proyecto nos enseñó la importancia de mantener un orden de versiones al momento de desarrollar . Se sufrieron numerosos retrasos producto de trabajar con versiones desactualizadas en diversas ramas que poseian cada una funcionalidades distintas.