SISTEMAS OPERATIVOS II

Sustentantes
Alexis Ubri Penzo 2012-5122
Jason Parra 2012-5692

Entregable
Reporte Proyecto I

Profesor
Roberto Abreu

Conceptos del protocolo Torrent

Torrent: Archivo utilizado por BitTorrent, un programa de intercambio de archivos punto a punto (P2P), para descargar un archivo (a menudo desde varias ubicaciones al mismo tiempo). No contiene el contenido que se distribuye, sólo los metadatos, que incluyen información sobre el archivo o grupo de archivos que se descargarán, como sus nombres, tamaños y estructura de carpetas.

Peer: computador que está descargando y cargando el archivo en el enjambre. Los archivos se descargan en piezas. Cuando un usuario descarga algunas piezas, automáticamente comienza a cargarlas. Un archivo se descargará más rápido si hay más personas involucradas en el enjambre. Un Peer se convierte en un Seed cuando ha completado el 100% del archivo y desea continuar subiendo.

Seed: computador que tiene un archivo torrent abierto en su cliente y está compartiéndolo con otros computadores.

Tracker: computador que sirve archivos torrent para descargar desde un sitio web. El Tracker mantiene información sobre todos los clientes de BitTorrent que usan cada torrent. Específicamente, el Tracker identifica la ubicación de red de cada cliente que carga o descarga el archivo P2P asociado con un torrent. También realiza un seguimiento de los fragmentos de ese archivo que cada cliente posee para ayudar a compartir datos de manera eficiente entre los clientes.

Bloque: es un fragmento de los datos del archivo a descargar que un cliente puede solicitar a un Peer. Dos o más bloques forman una pieza completa, que luego puede ser verificada para ser descargada.

Pieza: conjunto de bloques.

Handshake: mensaje utilizado para inicializar la comunicación entre el cliente y un Peer. Este contiene un identificador único del computador cliente y un hash que contiene la información del archivo a descargar extraída del archivo torrent.

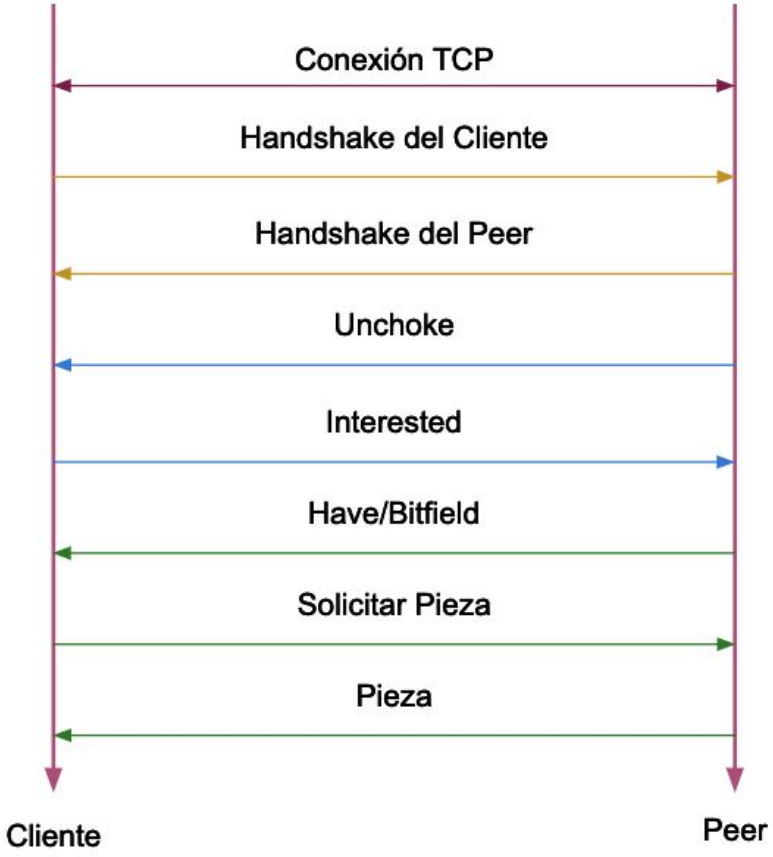
Comunicación entre el cliente y los Peers

Tras una respuesta exitosa de un mensaje de tipo Handshake entre un cliente y un Peer se realizan otros mensajes con el objetivo final de comenzar a intercambiar bloques de un archivo.

Cada cliente comienza en el estado *choked* y *not interested*. Eso significa que al cliente no se le permite solicitar piezas al Peer, ni tiene interés en esto. Durante la comunicación el cliente puede cambiar entre cuatro estados:

- **Choked:** Un Peer en estado choked no tiene permitido solicitar bloques a otro Peer.
- **Unchoked:** Un Peer en estado unchoked tiene permitido solicitar bloques a otro Peer.
- **Interested:** Indica que un Peer está interesado en solicitar bloques.
- **Not interested:** Indica que un Peer no está interesado en solicitar bloques.

Después del Handshake, el Peer envía al cliente un mensaje de *unchoke*, indicándole que ya puede comenzar a solicitar bloques, y este le devuelve con un mensaje *interested* confirmando que está interesado en comenzar a solicitar bloques, tal como se ilustra en la imagen de abajo.



Posteriormente, el Peer manda mensajes para informar al cliente sobre las piezas del archivo que tiene para luego el cliente proceder a solicitar la pieza que desea y el Peer responderle con los bloques deseados.

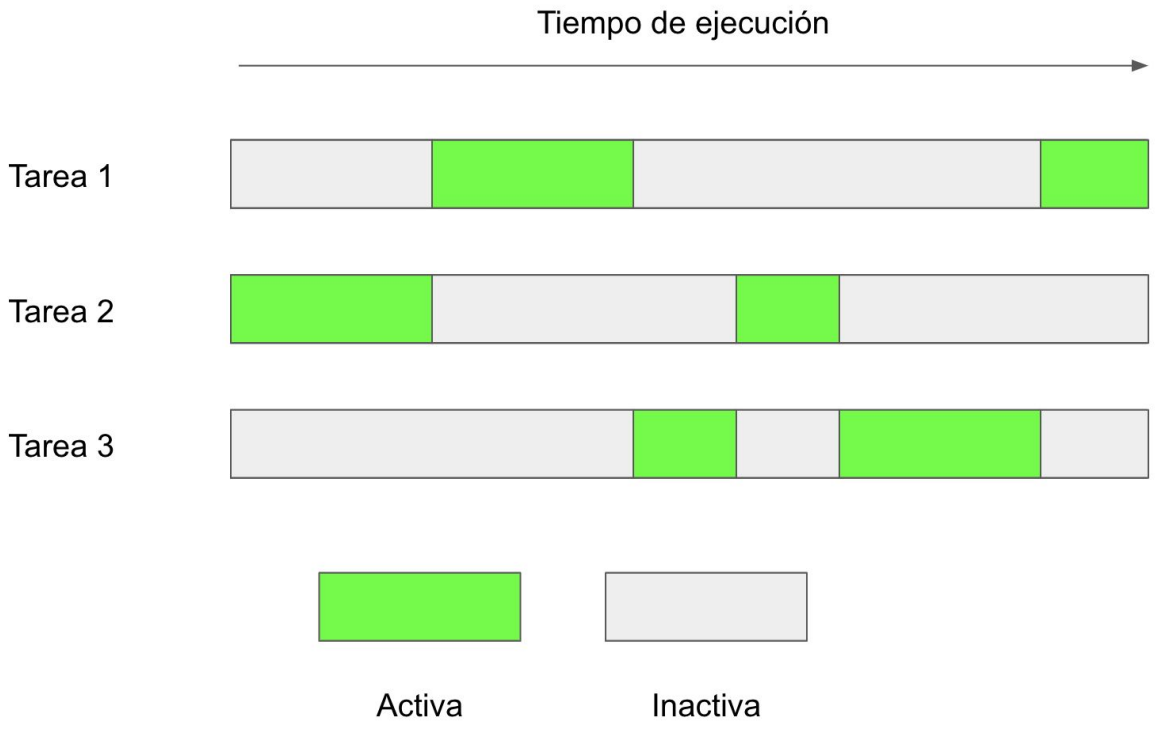
Arquitectura

Decidimos utilizar AsynchIO un librería asíncrona que nos permite mantener múltiples conexiones concurrentes con cada peer y como esta aplicación consiste en escribir y leer datos mediante red la utilizamos para esperar respuestas en vez de utilizar hilos.

Conceptos de asincronia:

1. Corrutina: Una función que puede ser pausada y resumida
2. Task: Básicamente planifica las corrutinas y les define que paso ejecutar.
3. Loop: Una cola de tasks a ser ejecutados.

De manera asíncrona conseguimos concurrencia al correr varias tareas bajo un solo proceso y utilizando un único hilo alternando entre estas mientras esperamos una respuesta y dándoles turnos de actividad para progresar en sus funciones.



Como vemos en la imagen arriba, tenemos tres tareas de ejemplo, que podrían estar haciendo lo mismo o cosas muy diferentes, corriendo bajo el mismo tiempo de ejecución, proceso e hilo pero se va alternando la actividad de cada una. Bajo esta lógica asíncrona diseñamos nuestra arquitectura donde cada conexión con cada *Peer* se maneja como una tarea diferente dentro de un ciclo.

Descarga de archivos

Utilizando lo explicado anteriormente. Dividimos el proceso de descarga en tareas. Cada tarea tiene asignada funciones de comunicarse e intercambiar mensajes con los *Peers* que se obtuvieron del *Tracker*, con estos mensajes, asumiendo que sean exitosos, obtenemos las piezas del archivo a descargar los cuales se acumulan en una cola y se procede a guardarse en el disco.

El proceso anterior se ve ilustrado en la imagen de arriba, cada tarea de *Peer* tiene el objetivo de extraer bloques del archivo a descargar para formar una pieza y luego proceder a su escritura en el archivo almacenado en el disco. Como vemos en el ejemplo, las tareas *Peer 1*, *Peer 2* y *Peer 3* obtienen las piezas deseadas y entonces alternan a la tarea *File Saver*, que procede a escribir dicha pieza en disco, y luego retorna a otra de las 3 tareas para seguir solicitando bloques a los *Peers*. Este proceso se repite hasta que se bajen todas las piezas necesarias del archivo a descargar.

Enviar archivos a otros Peers

Paralelamente a todo el proceso de conexión concurrentes asíncronas y de escritura en el disco, tenemos un *thread* que opera como servidor en espera de que Peers le soliciten piezas. En el momento de creación de la instancia de este servidor se realiza un Handshake con los Peers para que registren nuestra información en sus listas de Peers.

Cada vez que se graba una pieza en el disco duro, en el thread de descargas, se manda un mensaje de *HAVE* con el índice de la pieza a todos los peers que están conectados para dejarles saber que tenemos en disco y disponible esa pieza para su envío.

Cuando el servidor recibe un mensaje de *INTERESTED* envía uno de *UNCHOKED* al peer interesado, si este le responde uno de *REQUEST PIECE* pues procede a buscar en el archivo descargado la pieza deseada para enviarle por bloque las tramas de la pieza.

Es un proceso un tanto parecido al de descarga pero de manera inversa, actualmente logramos enviar sin problemas el mensaje de *HAVE* a los peers cuando descargamos las piezas pero aún seguimos trabajando en todo el proceso de mandar los bloques a los peers.

Clases

Tracker	
Parámetros	
torrent	Objeto con los datos del archivo torrent
Métodos	
_calculate_peer_id	Calcula un id único para el peer
_get_params	Obtiene los parámetros de la petición HTTP para un tracker HTTP
http_connect	Establece la conexión con el tracker HTTP mediante el url (announce url) del torrent y obtiene los peers
_parse_peers	Parsea los peers que responde el tracker a una lista de direcciones ip con sus puertos
_decode_port	Decodifica el puerte
parse_announce	Parseo del announce para obtener el nombre host y el puerto
udp_create_connection_request	Crea una conexión UDP
get_peers	Parsea los peers de la respuesta del tracker
udp_create_announce_request	Crea una los parámetros de una petición para un tracker UDP

get_connection_id	Obtiene el id de conexión de una petición UDP
announce_udp	Establece la conexión con el tracker UDP del torrent y obtiene los peers
startConnection	realizar un recorrido del *announce list* del torrent file para luego tomar la decisión si se debe realizar una conexión UDP o HTTP

Torrent	
Parámetros	
path	Dirección del archivo torrent
Métodos	
announce_url	Obtiene el *announce_url* del archivo torrent
announce_url_list	Obtiene la lista de los urls de los trackers incluidos en el archivo torrent
file_name	Obtiene el nombre del archivo torrent
piece_length	Obtiene el tamaño de una pieza del archivo torrent
number_of_pieces	Obtiene la cantidad de piezas del archivo torrent
get_piece_hash	Obtiene el hash de una pieza específica
info_hash	Obtiene un hash parseado en *SHA1* del directorio *info* del archivo torrent
size	Obtiene el tamaño del archivo a descargar
read_torrent_file	Lee el archivo torrent de una dirección y retorna un objeto parseado con los datos de dicho archivo

Peer	
Parámetros	
host	Dirección IP del peer
port	Puerto del peer
session	Instancia de la clase Pieces encargada de gestionar la descarga de piezas
torrent_session	Objeto con los datos del archivo torrent
file	Instancia del archivo descargado
Métodos	
_get_handshake_params	Obtiene los parámetros para realizar la petición de *handshake* al peer
download	Inicia el proceso de descarga de bloques desde un Peer
request_a_piece	Hace una petición para obtener una pieza del archivo en descarga
_calculate_peer_id	Genera un ID único para identificar al Peer
send_interested	Manda un mensaje de INTERESTED a un Peer
_consume	Parsea la data del buffer
_get_data	Obtiene y parsea el payload recibido del buffer
_handle_messages	Evalúa el ID de un mensaje y realiza una operación específica dependiendo de su tipo
send_have	Manda un mensaje de HAVE a un Peer
get_blocks_generator	Genera bloques
recv_piece_request	Procesa el mensaje al recibir una pieza
request_a_piece	Hace una petición de una pieza

FileSaver	
Parámetros	
dir	Dirección del directorio donde se escribirán los datos del archivo a descargar
torrent	Objeto con los datos del archivo torrent
Métodos	
get_file_path	Obtiene el *announce_url* del archivo torrent que es el enlace de Tracker
start	Obtiene los bloques de una cola y los escribe en un archivo
get_received_blocks_queue	Obtiene la cola de bloques recibidos

Block	
Parámetros	
piece_index	Dirección del directorio donde se escribirán los datos del archivo a descargar
begin	Inicio del bloque
length	Tamaño del bloque
data	Data del bloque
Métodos	
flush	Obtiene el *announce_url* del archivo torrent

Piece	
Parámetros	
dir	Dirección del directorio donde se escribirán los datos del archivo a descargar
torrent	Objeto con los datos del

	archivo torrent
Métodos	
get_file_path	Obtiene el *announce_url* del archivo torrent
start	Obtiene los bloques de una cola y los escribe en un archivo
get_received_blocks_queue	Obtiene la cola de bloques recibidos

Pieces	
Parámetros	
dir	Dirección del directorio donde se escribirán los datos del archivo a descargar
torrent	Objeto con los datos del archivo torrent
Métodos	
get_file_path	Obtiene el *announce_url* del archivo torrent
start	Obtiene los bloques de una cola y los escribe en un archivo
get_received_blocks_queue	Obtiene la cola de bloques recibidos

Upload	
Parámetros	
conn	Objeto con los datos de la conexión
addr	Dirección IP de la conexión
Métodos	
run	Imprime los mensajes de la conexión

File	
Parámetros	
torrent_file	Objeto con los datos del archivo torrent
file_name	Nombre del archivo en disco
fd	Instancia del archivo en disco
received_blocks_queue	Cola de bloques recibidos
Métodos	
get_received_blocks_queue	Obtiene la cola de bloques recibidos
get_file_path	Obtiene la dirección donde está alojado el archivo
get_piece_by_index	Obtiene una pieza del archivo descargado

Experimentos

Utilizando la herramienta WireShark para la captura de mensajes en la red, y utilizando el filtro de BitTorrent que trae instalado, podemos ver que se realiza sin problemas el Handshake con los peers y procedemos a decirles que estamos interesados en descargar sus bloques de piezas.

No.	Time	Source	Destination	Protocol	Length	Info
105	5.460912	10.0.0.8	188.165.205.116	BitTo…	134	Handshake
106	5.462071	10.0.0.8	80.56.114.141	BitTo…	122	Handshake
111	5.667882	80.56.114.141	10.0.0.8	BitTo…	160	Handshake Bitfield, Len:0x13 Have, Piece (Idx:0x3a)
113	5.668783	10.0.0.8	80.56.114.141	BitTo…	59	Interested
114	5.669219	10.0.0.8	80.56.114.141	BitTo…	59	Interested

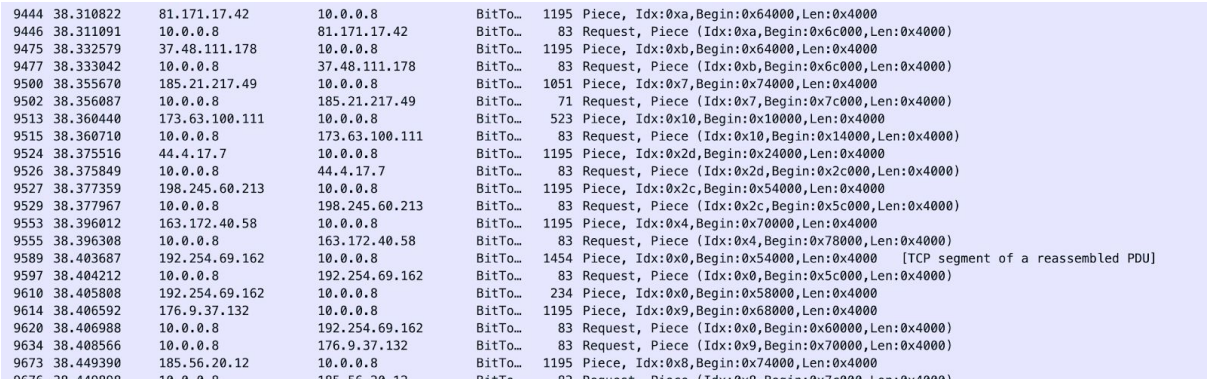
Sin mucho problemas vemos como la comunicación sigue fluida hasta el punto que entramos en el bucle de pedir y recibir piezas.

115	5.669341	10.0.0.8	80.56.114.141	BitTo…	71	Request, Piece (Idx:0x0,Begin:0x0,Len:0x4000)
116	5.669518	10.0.0.8	80.56.114.141	BitTo…	71	Request, Piece (Idx:0x0,Begin:0x4000,Len:0x4000)
117	5.908278	80.56.114.141	10.0.0.8	BitTo…	256	Have, Piece (Idx:0x88) Have, Piece (Idx:0x47) Have, Piece (Idx:0x1a) Have, Piece (Idx:0x14) Have, Piece (Idx:0x24) Have, Piece (
130	7.369643	188.165.205.116	10.0.0.8	BitTo…	139	Handshake
132	7.370094	10.0.0.8	188.165.205.116	BitTo…	71	Interested
133	7.616111	188.165.205.116	10.0.0.8	BitTo…	85	Bitfield, Len:0x13

167	8.920985	10.0.0.8	188.165.205.116	BitTo…	83	Request, Piece (Idx:0x1,Begin:0x8000,Len:0x4000)
239	10.929224	188.165.205.116	10.0.0.8	BitTo…	1454	Piece, Idx:0x1,Begin:0x4000,Len:0x4000 [TCP segment of a reassembled PDU]
258	10.930148	188.165.205.116	10.0.0.8	BitTo…	923	Piece, Idx:0x1,Begin:0x8000,Len:0x4000
259	10.930198	10.0.0.8	188.165.205.116	BitTo…	83	Request, Piece (Idx:0x1,Begin:0xc000,Len:0x4000)
263	10.930651	10.0.0.8	188.165.205.116	BitTo…	83	Request, Piece (Idx:0x2,Begin:0x0,Len:0x4000)
289	13.015585	188.165.205.116	10.0.0.8	BitTo…	1454	Piece, Idx:0x1,Begin:0xc000,Len:0x4000 [TCP segment of a reassembled PDU]
301	13.020903	10.0.0.8	188.165.205.116	BitTo…	83	Request, Piece (Idx:0x2,Begin:0x4000,Len:0x4000)

Sin embargo notamos algo particular en estas capturas, solo se estaba descargando bloques desde un solo Peer. Tras un buen tiempo de investigación y probar otro archivo torrent note que lo que pasaba era que justamente en ese tiempo solo estaba disponible dicho Peer.

Luego probé con otro archivo torrent y pude notar que si estaba comunicándome con varios peers concurrentemente.



9444	38.310822	81.171.17.42	10.0.0.8	BitTo…	1195	Piece, Idx:0xa,Begin:0x64000,Len:0x4000
9446	38.311091	10.0.0.8	81.171.17.42	BitTo…	83	Request, Piece (Idx:0xa,Begin:0x6c000,Len:0x4000)
9475	38.332579	37.48.111.178	10.0.0.8	BitTo…	1195	Piece, Idx:0xb,Begin:0x64000,Len:0x4000
9477	38.333042	10.0.0.8	37.48.111.178	BitTo…	83	Request, Piece (Idx:0xb,Begin:0x6c000,Len:0x4000)
9500	38.355670	185.21.217.49	10.0.0.8	BitTo…	1051	Piece, Idx:0x7,Begin:0x74000,Len:0x4000
9502	38.356087	10.0.0.8	185.21.217.49	BitTo…	71	Request, Piece (Idx:0x7,Begin:0x7c000,Len:0x4000)
9513	38.360440	173.63.100.111	10.0.0.8	BitTo…	523	Piece, Idx:0x10,Begin:0x10000,Len:0x4000
9515	38.360710	10.0.0.8	173.63.100.111	BitTo…	83	Request, Piece (Idx:0x10,Begin:0x14000,Len:0x4000)
9524	38.375516	44.4.17.7	10.0.0.8	BitTo…	1195	Piece, Idx:0x2d,Begin:0x24000,Len:0x4000
9526	38.375849	10.0.0.8	44.4.17.7	BitTo…	83	Request, Piece (Idx:0x2d,Begin:0x2c000,Len:0x4000)
9527	38.377359	198.245.60.213	10.0.0.8	BitTo…	1195	Piece, Idx:0x2c,Begin:0x54000,Len:0x4000
9529	38.377967	10.0.0.8	198.245.60.213	BitTo…	83	Request, Piece (Idx:0x2c,Begin:0x5c000,Len:0x4000)
9553	38.396012	163.172.40.58	10.0.0.8	BitTo…	1195	Piece, Idx:0x4,Begin:0x70000,Len:0x4000
9555	38.396308	10.0.0.8	163.172.40.58	BitTo…	83	Request, Piece (Idx:0x4,Begin:0x78000,Len:0x4000)
9589	38.403687	192.254.69.162	10.0.0.8	BitTo…	1454	Piece, Idx:0x0,Begin:0x54000,Len:0x4000 [TCP segment of a reassembled PDU]
9597	38.404212	10.0.0.8	192.254.69.162	BitTo…	83	Request, Piece (Idx:0x0,Begin:0x5c000,Len:0x4000)
9610	38.405808	192.254.69.162	10.0.0.8	BitTo…	234	Piece, Idx:0x0,Begin:0x58000,Len:0x4000
9614	38.406592	176.9.37.132	10.0.0.8	BitTo…	1195	Piece, Idx:0x9,Begin:0x68000,Len:0x4000
9620	38.406988	10.0.0.8	192.254.69.162	BitTo…	83	Request, Piece (Idx:0x0,Begin:0x60000,Len:0x4000)
9634	38.408566	10.0.0.8	176.9.37.132	BitTo…	83	Request, Piece (Idx:0x9,Begin:0x70000,Len:0x4000)
9673	38.449390	185.56.20.12	10.0.0.8	BitTo…	1195	Piece, Idx:0x8,Begin:0x74000,Len:0x4000

Resultados

Logramos descargar de manera exitosa archivos de poco tamaño pero al intentar descargar archivos de mayor tamaño se nos presentan errores de “bloques y piezas no validas”. Sospechamos a que esto se debe a que todavía no estamos mandando archivos de manera simultánea a como estamos descargando y los Peer en comunicación pierden el interés de compartir bloques con nuestro cliente.