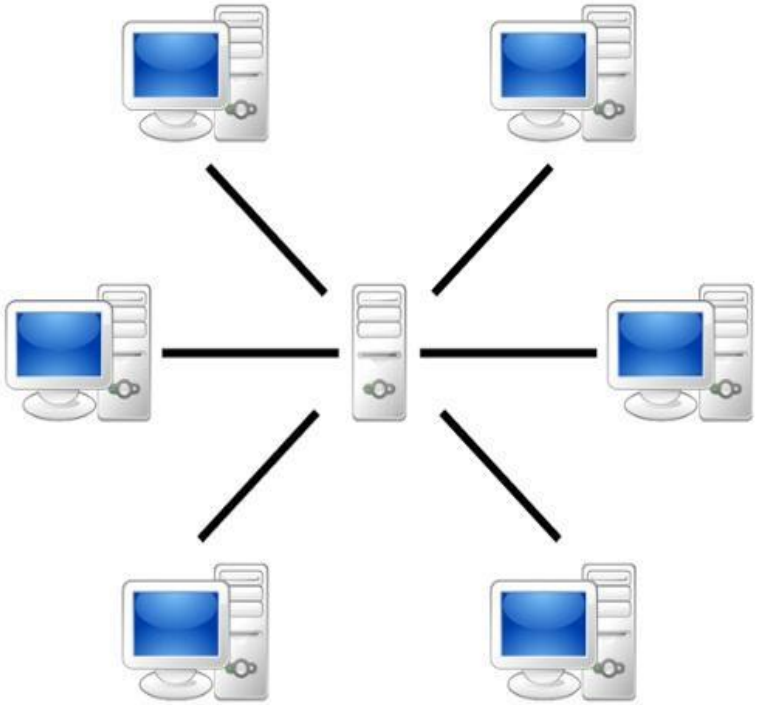
BitTorrent

Por Lisandro “La matraca” Garcia, Bruno “Enzo” Malvasio y Santino “Petaca” García

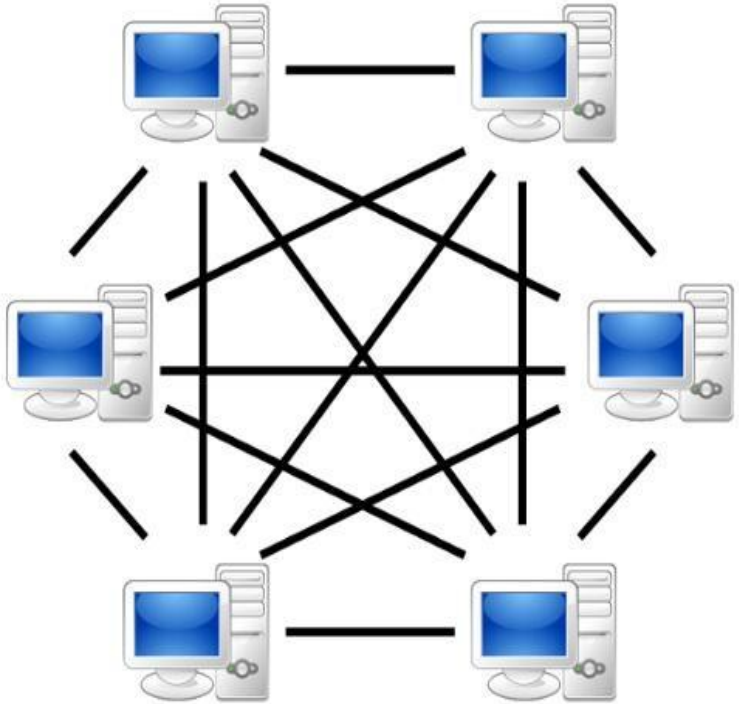
¿Qué es BitTorrent?

BitTorrent es un protocolo de transferencia de datos orientado en una red de procedimientos peer-to-peer.

Optimizado para una alta velocidad de descarga gracias a su descentralización del proceso de descarga.



Server-based



P2P-network

Sistemas Peer-to-Peer (P2P)

Los elementos que lo forman comparten sus recursos para proveer el servicio que el sistema está diseñado para proveer

Los elementos del sistema deben:

- Proveen servicios a otros elementos
- Piden servicios a otros elementos

Se debe cumplir con esto para ser considerado un sistema P2P, pero hay excepciones

Funciones de los sistemas P2P

- Función de enrolamiento
- Función de descubrimiento de peers
- Función de guardado de datos

La función de enrolamiento no se ve en BitTorrent a menos que sea un torrent privado.

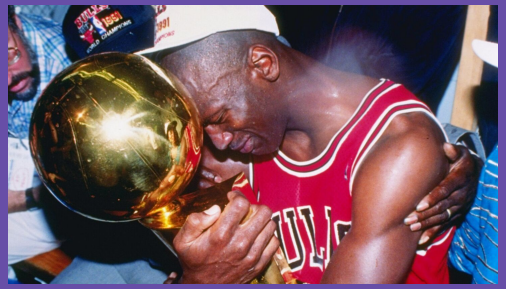
El descubrimiento de peers se realiza en BitTorrent de forma centralizada con un tracker, o de forma descentralizada con DHT o PEX (se verá más adelante).

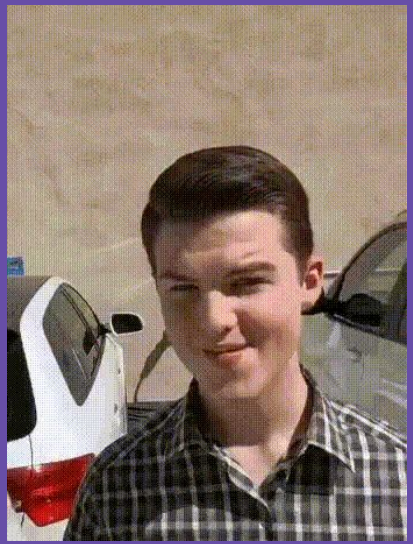
¿Por qué se creó BitTorrent?

Bram Cohen desarrolló BitTorrent en 2001 para resolver un problema importante de internet en ese momento: Distribuir archivos grandes a muchas personas sin saturar los servidores.

¿Por qué usar BitTorrent?

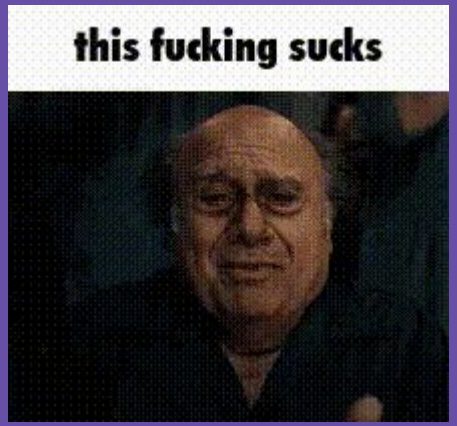
- Más rápido (la mayoría de veces)
- Mismo ancho de banda, mejor usado
- No satura los servidores
- No depende de un sólo servidor
- Simple reanudación



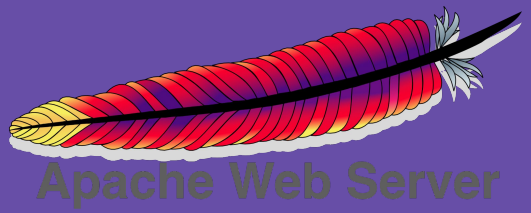


¿Por qué NO usar BitTorrent?

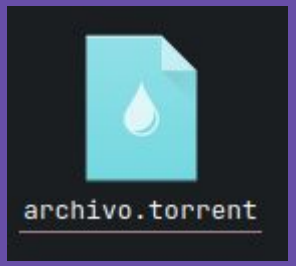
- Riesgos de seguridad
- Contenido ilegal/pirata (*I am not from the US!*)
- Archivos infectados
- Si hay pocos peers, puede ser mejor la descarga directa



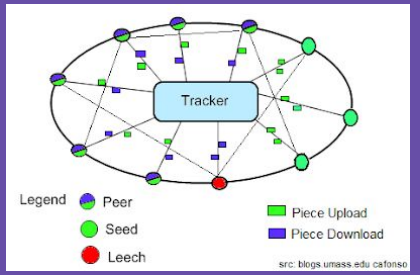
Entidades Involucradas



Servidor Web



Archivo Torrent



Tracker

Descargador Original

Buscador web de los usuarios finales

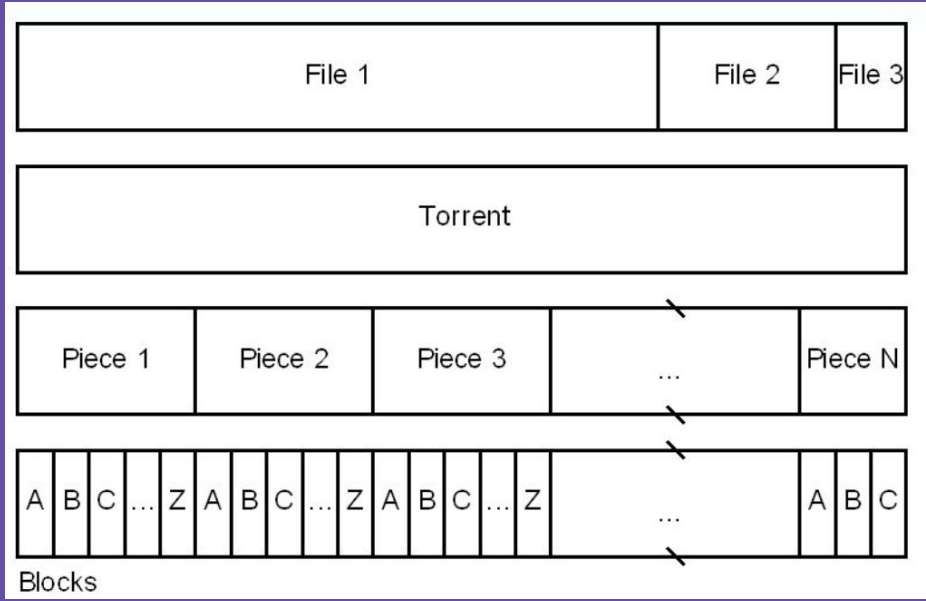
Descargador de los usuarios finales

Funcionamiento general

BitTorrent divide el/los archivo/s en múltiples piezas (*pieces*)

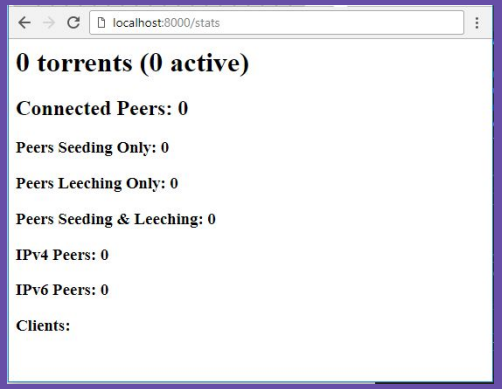
Un endpoint (peer) interesado en los datos necesita descargar todas las pieces a partir de otros peers en el swarm

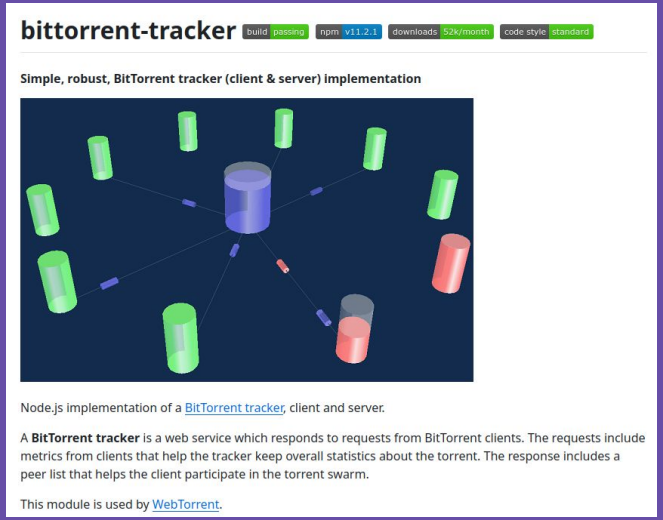
A su vez, estas pieces se dividen en bloques que se transmitirán a través de mensajes entre peers



¿Qué debe hacer un host para comenzar a servir datos?

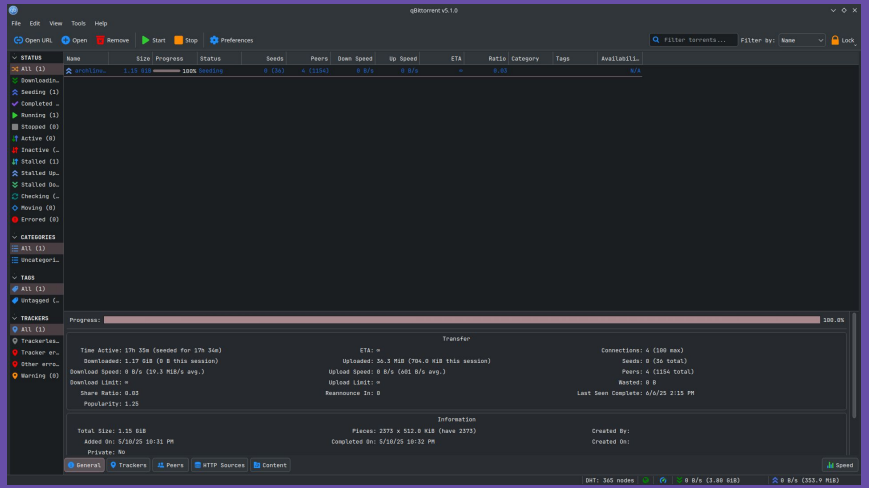
1. Se necesita iniciar un Tracker
2. Se debe iniciar un servidor web ordinario
3. Se asocia la extensión .torrent con el mimetype *application/x-bittorrent*
4. Se genera un archivo metainfo (*.torrent)*
5. Se pone el archivo metainfo en el servidor web
6. Se realiza un enlace al archivo metainfo en otra página web
7. Se inicia un descargador que ya tenga el archivo completo





¿Qué debe hacer un usuario para comenzar a descargar datos?

1. Instalar un cliente BitTorrent
2. Investigar la web en busca de un torrent
3. Clickear en un link que lo lleve a un archivo metainfo
4. Seleccionar donde guardarlo o reanudar una descarga parcial que ya tenía
5. Esperar a que la descarga termine
6. Cerrar el cliente o seguir subiendo el archivo



Archivo Metainfo (.torrent)

d8:**announce**35:http://tracker.example.com/announce10:**created by**13:mktorrent 1.113:**creation date**i1748799493e4:**info**d6:**length**i65703e4:**name**9:perro.jpg12:**piece length**i262144e6:**pieces**20:D%’ÓÞ$_h’éf/Ød¿ee

- **announce**: Incluye la URL del tracker.
- **announce-list** (opcional): Es una extensión de la especificación de BitTorrent oficial. Es una llave que refiere a una lista de listas de trackers.
- **creation-date** (opcional): Es el tiempo de creación del torrent en el formato standard UNIX.
- **comment** (opcional): Es una forma libre de comentarios textuales del autor (string).
- **created by** (opcional): Es el nombre y la versión del programa usado para crear el archivo metainfo (string).
- **encoding** (opcional): Es la string que especifica el formato de encoding usado para generar las piezas que son parte del diccionario info en el *.torrent* (string).
- **info**: Es un diccionario que describe el o los archivos del torrent. Hay dos formas posibles:
  - Una para el caso de un torrent de un solo archivo, que no tiene estructura de directorios.
  - Otra para el caso del torrent de múltiples archivos.

Diccionario info

- **piece length**: Mapea el número de bytes en cada pieza que el archivo se divide.
- **pieces**: Cadena que consiste en la concatenación de los valores de hash SHA1 de 20 bytes, uno por cada pieza.
- **private** (opcional): Es un integer que si se setea a 1, el cliente debe publicar su presencia para conseguir otros peers únicamente a través de los trackers explícitamente descritos en el archivo metainfo. Si este campo está seteado a 0 o no está presente, el cliente debe tener peers por otros medios (como PEX, DHT). Private se interpreta como “no hay fuente de peers externa”.

En el caso que esté en modo single-file (un solo archivo):

- **name**: Mapea a una string encodeada en UTF-8 que es el nombre sugerido para guardar el archivo.
- **length**: Es el largo del archivo en bytes (integer).

En el caso que esté en modo multiple-file (muchos archivos):

- **name**: Es el nombre del directorio en el que se guardarán los archivos. Es puramente en forma de consejo. (string).
- **files**: Una lista de diccionarios, uno por cada archivo. Cada diccionario en esta lista contiene las siguientes llaves:
    - **length**: El largo del archivo en bytes. (integer).
    - **path**: Una lista que contiene una o más strings que juntas representan el path y el filename. Cada elemento en la lista representa un nombre de directorio o (siendo el caso del último elemento) el filename.

Comunicación Tracker-Peer

Asiste en la comunicación entre peers

Mantiene registro de donde residen las copias de los archivos en las computadoras de los peers, y coordina la transmisión y reensamblamiento de los datos

Actualmente se usan las tablas distribuidas de hashes (DHT) y el peer exchange (PEX) para descubrir peers sin trackers, pero se siguen incluyendo para velocidad de descubrimiento

Tracker protocol

- communicates with clients via HTTP/HTTPS
- client GET request
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - stats: bytes uploaded, downloaded, left
- tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - stats: complete, incomplete

Comunicación Tracker-Peer: GET Requests

- **info_hash:** Hash de 20 bytes de la llave *info* del archivo metainfo.
- **peer_id:** String para identificar al usuario/cliente.
- **ip:** Dirección IP del usuario/cliente.
- **port:** Puerto en el que el cliente está escuchando (6881-6889).
- **uploaded:** Cantidad total de paquetes subidos (desde el evento started).
- **downloaded:** Cantidad total de paquetes descargados (desde el evento started).
- **left:** Número de bytes restantes por descargar.
- **compact:** Si es cliente acepta una string de peers (es un formato específico).
- **no_peer_id:** Si el tracker puede omitir el campo peer_id en el diccionario de peers.
- **event:** Puede ser started, completed o stopped.
- **numwant:** Cantidad de peers que el cliente desea obtener (default 50).

Comunicación Tracker-Peer: GET Requests

- **failure reason**: Mensaje de error.
- **warning message**: Advertencia mostrada como error.
- **interval**: Intervalo en el cual el cliente debe esperar antes de seguir enviando requests al tracker.
- **min interval**: Opcional. Intervalo mínimo de anuncio o transferencia, si este está presente los clientes no deberían anunciarse más veces que el mismo.
- **tracker id**: Un valor que el cliente devuelve en sus anuncios próximos. Siempre es necesario tener al menos un valor de este campo, aunque sea de un anuncio previo.
- **complete**: Peers con el archivo completo, es decir seeders, en formato de integer.
- **incomplete**: Número de peers no-seeders, AKA leechers.
- **peers (modelo de diccionario)**: Es una lista de diccionarios que incluyen las siguientes llaves:
    - **peer id**: La ID autoseleccionada del peer, como se describe más arriba para la **request** para el tracker.
    - **ip**: La IP del peer, ya sea IPv6 o IPv4.
    - **port**: El número del puerto del peer.
- **peers (modelo binario)**: En vez de usar el modelo del diccionario, el valor de *peers* puede ser una string que consiste en múltiples de 6 bytes. Los primeros 4 siendo la dirección IP y los últimos 2 el puerto.

Comunicación Peer-Peer

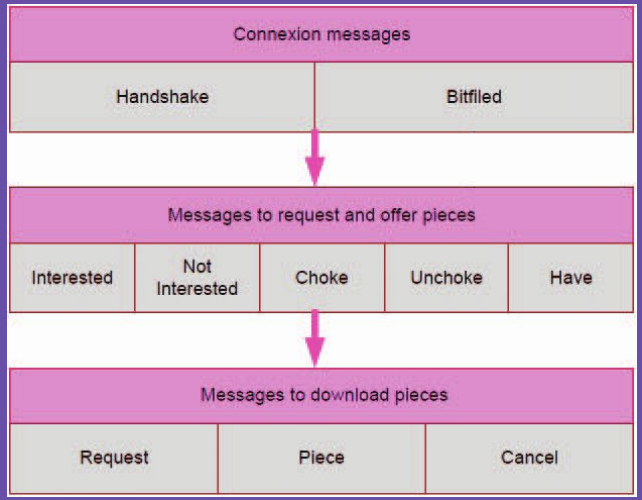
Operan sobre TCP (L4)

Son conexiones simétricas, es decir, los peers deben actualizarse su estado mutuamente

Intercambian *pieces* por index igual que como están en el .torrent (archivo metainfo)

Consiste en dos estados: *choked* e *interested*. Un cliente debe mantener información de estado con cada conexión con un peer remoto

- Peer states
  - (Un)Choked – Is the remote peer allowed to send data?
  - (Un)Interested – Is the peer interested in receiving data?
  - Data Transfer takes place when one side is interested, and the other is not choking.

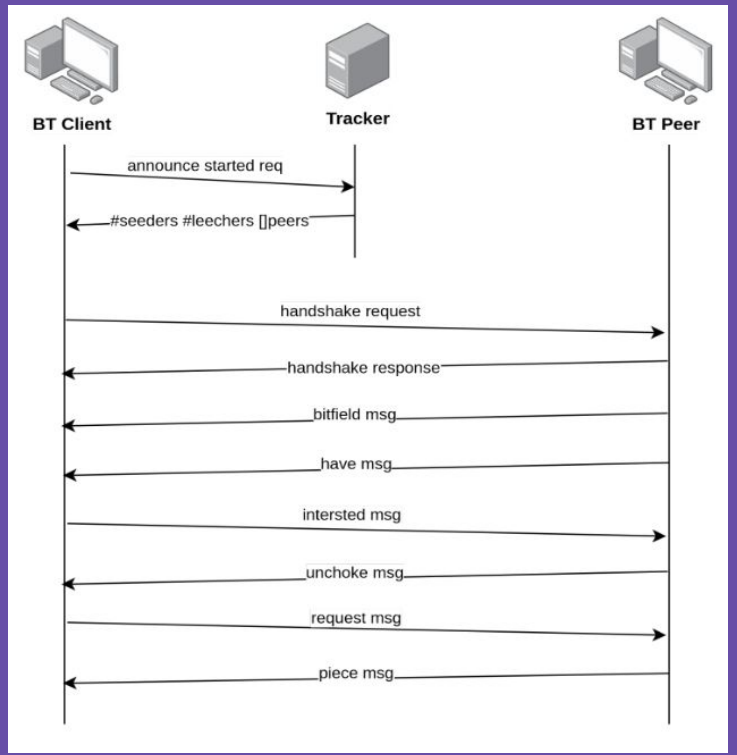


Comunicación Peer-Peer: Mensajes

- **handshake:** Conexión inicial entre Peers. Contiene **info_hash** y **peer_id**.
- **bitfield (opcional):** Representa las *pieces* descargadas de forma exitosa.
- **keep-alive:** Para que los Peers no cierren la conexión.
- **choke:** sin payload.
- **unchoke:** sin payload.
- **interested:** sin payload.
- **not interested:** sin payload.

- **have:** La carga es el index de una *piece* descargada exitosamente y verificada por hash.
- **request:** Es usado para pedir un bloque con el **index del *piece***.
- **piece:** Contiene de payload el bloque que se pidió con el **request**.
- **cancel:** Se usa para cancelar requests de bloques.
- **port:** Para clientes que usan **DHT** es el puerto que el Peer está escuchando.

Ejemplo de comunicación entre Peers



Extensiones para el protocolo original aceptadas

- **DHT (Distributed hash table):** Tabla de enrutamiento disponible para cada nodo, eliminando la dependencia al Tracker.
- **PEX (Peer exchange):** Mecanismo que permite a los peers intercambiar información sobre otros peers conectados a la misma swarm.
- **Magnet link:** Link web que contiene toda la metadata y datos necesarios para que un peer pueda unirse a una swarm. Elimina la necesidad del archivo metainfo.
- **Holepunch:** Ofrece una forma de conectarse saltando restricciones de filtradores NAT o firewall.

Implementación de BitTorrent (Python): Estructura general del programa

- download_torrent(magnet/torrent, ruta)

- session

Implementación de BitTorrent (Python): Componentes clave del programa

- bdecode()
- torrent_info()
- session()
- add_torrent(params)

```python
def download_torrent(torrent_path_or_magnet, download_dir):
    ses = lt.session({'listen_interfaces': '0.0.0.0:6881'})

    print("Configurando la sesión libtorrent...")

    params = lt.add_torrent_params()
    params.save_path = download_dir

    if download_dir == '.':
        print("Los archivos se descargarán en el directorio actual.")
    else:
        print(f"Los archivos se descargarán en '{download_dir}'.")

    if torrent_path_or_magnet.startswith("magnet:"):
        print(f"Agregando magnet link: {torrent_path_or_magnet}")
        params.url = torrent_path_or_magnet
    elif os.path.exists(torrent_path_or_magnet):
        print(f"Agregando archivo .torrent: {torrent_path_or_magnet}")
        try:
            e = lt.bdecode(open(torrent_path_or_magnet, 'rb').read())
            info = lt.torrent_info(e)
            params.ti = info
        except Exception as e:
            print(f"Error al leer el archivo torrent: {e}")
            return
```

Implementación de BitTorrent (Python): Seguimiento de la descarga

torrent_handle.status() devuelve:

- status
- progress
- download_rate
- num_seeds / num_peers
- has_metadata
- is_finished

```
while not h.status().has_metadata:
    print("Esperando metadatos...")
    time.sleep(1)
```

```
print("\n--- Estado Actual ---")
print(f"Estado: {state_str[s.state]}")
print(f"Progreso: {s.progress * 100:.2f}%")
print(
    f"Descargado: {s.total_payload_download / 1000:.2f} kB")
print(f"Subido: {s.total_payload_upload / 1000:.2f} kB")
print(f"Vel. Descarga: {s.download_rate / 1000:.2f} kB/s")
print(f"Vel. Subida: {s.upload_rate / 1000:.2f} kB/s")
print(f"Seeds: {s.num_seeds}")
print(f"Pares: {s.num_peers}")
print("----------------------")
```

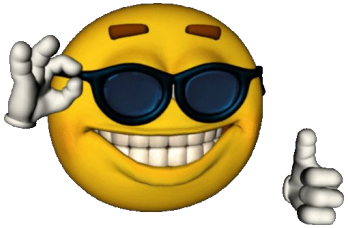
```
print(f"\rEstado: {state_str[s.state]:<22} | Progreso: {s.progress * 100:5.2f}% | "
      f"Descargado: {s.total_payload_download / 1000:8.2f} kB | Vel. Descarga: {s.download_rate / 1000:7.2f} kB/s | "
      f"Seeds: {s.num_seeds:3d} | Pares: {s.num_peers:3d}  ", end='')
```

Implementación de BitTorrent (Python): Seguimiento de la descarga

Se leen los inputs ingresados con getch(), lo que permite:

- p: Pausa (handle.pause())
- r: Reanuda (handle.resume())
- c: Cancela (session.remove_torrent())
- s: Imprime estado actual
- q: Sale limpiamente

```
char = getch()
if char:
    char = char.lower()
    if char == 'p':
        h.pause()
        print("\nDescarga pausada.")
        time.sleep(2)
    elif char == 'r':
        h.resume()
        print("\nDescarga reanudada.")
        time.sleep(2)
    elif char == 'c':
        print("\nCancelando descarga...")
        ses.remove_torrent(h)
        print("Descarga cancelada.")
        break
    elif char == 's':
        print("\n--- Estado Actual ---")
        print(f"Estado: {state_str[s.state]}")
        print(f"Progreso: {s.progress * 100:.2f}%")
        print(
            f"Descargado: {s.total_payload_download / 1000:.2f} kB")
        print(f"Subido: {s.total_payload_upload / 1000:.2f} kB")
        print(f"Vel. Descarga: {s.download_rate / 1000:.2f} kB/s")
        print(f"Vel. Subida: {s.upload_rate / 1000:.2f} kB/s")
        print(f"Seeds: {s.num_seeds}")
        print(f"Pares: {s.num_peers}")
        print("----------------------")
        time.sleep(2)  # Pausa para que el usuario pueda leer
    elif char == 'q':
        print("\nSaliendo...")
        break
```



Opinión

¿Recomendarías BitTorrent?
¡Mándanos tus comentarios!