

BitTorrent

Patricia Córdoba Hidalgo y David Cabezas Berrido

Índice

1. Introducción	2
2. Arquitectura Peer-to-Peer (P2P)	2
2.1. Escalabilidad en arquitecturas P2P para la distribución de archivos	2
3. BitTorrent	3
3.1. Otros mecanismos de funcionamiento:	3
4. Capturas con Wireshark	4
4.1. Conexión con Tracker	4
4.2. Conexión con Peers	6
5. Referencias	8

1. Introducción

BitTorrent es un protocolo para descargar y compartir archivos a través de Internet, especialmente archivos grandes como videos y audios. Este protocolo se aprovecha de las ventajas ofrecidas por la arquitectura Peer-to-Peer. Fue desarrollado por Bram Cohen en abril de 2001 y la primera implementación se llevó a cabo en junio de ese mismo año. En Noviembre de 2004, BitTorrent fue responsable del 25 % de todo el tráfico de Internet y en Febrero de 2013 se dedicó el 6 % de la banda ancha mundial a la distribución de archivos, del cual BitTorrent fue responsable del 3.35 % total. En 2013, el número de usuarios concurrentes en BitTorrent oscilaba entre 15 y 27 millones. En Junio de 2012 se alcanzó un pico de 150 millones de usuarios activos simultáneamente. A continuación explicaremos el funcionamiento de este protocolo.

También existe BitTorrent como cliente para descargar archivos a través de este protocolo. Otros clientes de BitTorrent son: qBittorrent, Tixati, Transmission, μ Torrent, y para streaming: Butter Project, Popcorn Time, Torrents-Time, BitTorrent Live...

2. Arquitectura Peer-to-Peer (P2P)

La arquitectura Peer-to-Peer es una alternativa a la arquitectura Cliente-Servidor, donde se minimiza la dependencia de un servidor que tiene que estar siempre en correcto funcionamiento. En lugar de ser una arquitectura centralizada, los distintos hosts (llamados *peers*) intermitentemente conectados se comunican directamente entre sí, actuando a la vez como clientes y como servidores. Estos peers son máquinas controladas por usuarios.

Una de las principales aplicaciones de esta arquitectura es la distribución de archivos.

Para comprar un archivo en una arquitectura Cliente-Servidor el servidor debe enviar una copia del archivo a cada cliente. Sin embargo, en una arquitectura P2P cada peer puede redistribuir una porción del archivo a otros peers, asistiendo así al servidor en el proceso de distribución. Esto presenta una mayor escalabilidad.

2.1. Escalabilidad en arquitecturas P2P para la distribución de archivos

Si queremos compartir un archivo con N clientes y la velocidad de subida del servidor es u_s (bytes/s), el tiempo de distribución en una arquitectura CS es, al menos, $\frac{NF}{u_s}$ segundos, donde F es el tamaño del archivo en bytes. Por lo tanto el tiempo de distribución crece linealmente con el número de clientes.

Por otra parte, en una arquitectura P2P los peers pueden difundir los trozos del archivo que ya hayan descargado. Sea u_i la velocidad de subida del peer i -ésimo en bytes/s, el tiempo de transferencia del archivo a todos los peers es, al menos, $\frac{NF}{u_s + \sum_{i=1}^N u_i}$ segundos.

En la [Figura 2.25](#) del Kurose, Ross 2013 se comparan los tiempos de difusión de ambas arquitecturas suponiendo $u_i = u \forall i$, $\frac{F}{u} = 1h$ y $u_s = 10u$. No solo el tiempo de difusión es menor en la arquitectura P2P, sino que además está mayorado por 1h.

Kurose, Ross 2013 (Capítulo 2, página 148)

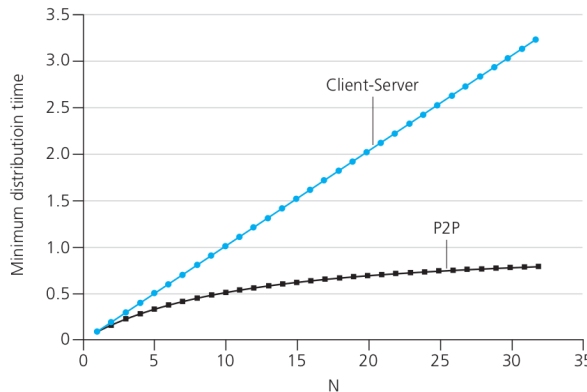






Figure 2.25 ♦ Distribution time for P2P and client-server architectures

3. BitTorrent

BitTorrent es un protocolo P2P para distribución de archivos. El conjunto de peers que participan en la comunicación se llama *torrent*. Los peers en un torrent descargan chunks de igual tamaño del archivo (típicamente 256 Kbytes). Al principio, un peer no tiene ningún chunk y va acumulándolos con el paso del tiempo a la vez que va difundiendo los que ya tiene al resto de peers. Un peer puede, en cualquier momento, abandonar el torrent con un subconjunto de chunks del archivo y volver a conectarse en otro momento para reanudar la comunicación. Una vez descargado el archivo, un peer puede seguir en el torrent compartiéndolo o abandonar la red.

Tamaño	Progreso	%	Piezas	Piezas	Prioridad	Valoración	Resolución	Duraci	Retransmisión
302 B	302 B	100.0 %	1		Omitir				No
7.61 GB	2.16 GB	28.3 %	1950		Normal	624.1 KB/s	1920x802	1h 41m	Si
111 B	0 B	0.0 %	1		Omitir				No
196 B	0 B	0.0 %	1		Omitir				No

Captura de μ Torrent: Se observa un archivo con 1950 chunk, de los cuales sólo se han descargado algunos.

El principal mecanismo de BitTorrent es el siguiente:

Un archivo con la extensión **.torrent**, que contiene información sobre el archivo a descargar (nombre, longitud, información de hashing) y la url de un *tracker* (un servidor que asiste a la comunicación entre peers en el torrent), se sube a un servidor web. Un usuario ejecuta el **.torrent** en un cliente para unirse al torrent. Cuando un peer se une, a través de un protocolo basado en HTTP, manda información al tracker sobre qué archivo está descargando y cuál es su puerto de escucha, entre otras cosas. Periódicamente los peers le informan de que siguen activos para que el tracker lleve un registro de los peers participantes. Cuando se une un peer, el tracker le envía la IP de un subconjunto de peers elegido aleatoriamente, con los que intenta establecer comunicación TCP. Los peers con los que establece comunicación TCP (“vecinos”) pueden dejar el torrent, y también pueden conectarse con él nuevos peers. Periódicamente, cada peer solicita a sus vecinos la lista de chunks que tienen (por conexión TCP) y luego solicita chunks que no tiene.

Para determinar qué chunk solicitar primero se usa la **heurística del más raro**. De entre los chunks que no tiene, el peer solicita aquel que menos se repita entre las listas de chunks de sus vecinos, con el fin de equilibrar el número de copias de cada chunk en el torrent. De esta forma se reduce el riesgo de que todos los peers que tienen un chunk estén desconectados simultáneamente.

Para decidir qué solicitudes atender, un peer da prioridad a los vecinos que actualmente le proporcionan más datos. Cada peer determina los cuatro peers que más datos le comparten, basándose en la media de los últimos 20 segundos, y es a éstos a los que atiende (se les llama *unchoked peers*). Este top cuatro se recalcula cada 10 segundos. Además, cada peer escoge adicionalmente uno de sus vecinos al azar y atiende sus peticiones (**Optimistic Unchoking**). Éste vecino se recalcula cada 30 segundos. De esta forma, los nuevos peers en el torrent consiguen chunks para intercambiar, y así ascender a los cuatro vecinos favoritos de algunos peers y permite encontrar mejores conexiones. A largo plazo, este criterio provoca que los peers que comparten información a velocidades similares se encuentren mutuamente.

Este sistema presenta una gran escalabilidad ya que su cuello de botella es la sobrecarga del ancho de banda del tracker, pero la función de éste requiere muy poco ancho de banda.

3.1. Otros mecanismos de funcionamiento:

- **Pipelining y mini-chunks:** En una conexión TCP, es importante tener siempre varias solicitudes pendientes para evitar el retraso entre chunks enviados. BitTorrent facilita esto subdividiendo los chunks en piezas más pequeñas (normalmente 16 Kbytes) y manteniendo algunas solicitudes (normalmente cinco) en un pipeline. Cada vez que una sub-pieza (mini-chunk) llega a su destinatario, éste manda una nueva solicitud.
- Se da **Prioridad Estricta** a los mini-chunks que forman parte de un chunk del que ya se han descargado algunos mini-chunks. Esto ayuda a que se consigan chunks completos lo más rápido posible.

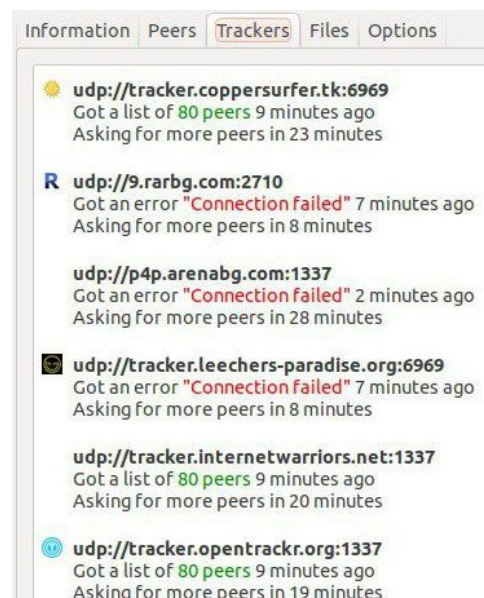
- **Primera Pieza Aleatoria:** Cuando un peer se une al torrent por primera vez, no tiene nada que compartir. Su prioridad debe ser conseguir un chunk completo lo antes posible. Si solicitase el chunk más raro, tendría pocas fuentes de las cuales descargar mini-chunks y la descarga sería más lenta. Es por eso que las piezas a descargar se solicitan aleatoriamente hasta que se consiga el primer chunk completo. Una vez conseguido, se cambia a la heurística del más raro.
- **Endgame Mode:** Cuando un peer tiene solicitadas todas las sub-piezas que le faltan, solo le queda esperar a que se las envíen. Si una de ellas se envía a baja velocidad, el peer desperdicia ancho de banda. Para evitar esto, el peer solicita todas las mini-piezas restantes a todos los peers y va cancelando solicitudes conforme las vaya recibiendo. De esta manera, a cambio de un desperdicio del ancho de banda por un corto periodo de tiempo, el archivo se termina de descargar rápidamente.
- **Anti-Snubbing:** Cuando un peer deja de recibir chunks de sus vecinos, empieza a tener una velocidad de descarga baja hasta que encuentre nuevos peers adecuados, cosa que sólo puede hacer por medio de Optimistic Unchoking. BitTorrent considera que un peer que lleva más de un minuto sin descargar un chunk desde un peer particular está siendo despreciado (*"snubbed"*) por éste y deja de enviarle piezas a no ser que lo encuentre con Optimistic Unchoking. Esto frecuentemente resulta en más de un Optimistic Unchoke simultáneo (una excepción). De esta manera se recuperan más rápido los ratios de descarga.
- **Sólo subida:** Una vez que un peer ha descargado el archivo completo, ya no hace uso de su velocidad de bajada. Entonces pasa a enviar chunks a los peers que mejor aprovechan su capacidad de subida, priorizando aquellos a los que nadie más les está compartiendo.
- **UDP Tracker:** Una extensión reciente de BitTorrent es el UDP Tracker Protocol. El tracker usa el protocolo UDP para la transmisión de datos, en vez del protocolo HTTP (sobre TCP). Esto tiene mejor rendimiento y menor sobrecarga en el tracker, pero no todos los clientes de BitTorrent lo soportan. Aun así, usa TCP como protocolo de transporte.

4. Capturas con Wireshark

En esta sección usaremos la herramienta Wireshark para ver el intercambio de paquetes al utilizar BitTorrent. Descargamos un torrent con el cliente Transmission. BitTorrent es un protocolo que no está encriptado por defecto, pero la mayoría de clientes permite encriptar el tráfico. Veremos que en nuestro caso lo encripta.

4.1. Conexión con Tracker

Nos centraremos en la conexión con los trackers.



Captura de Transmission: Lista de los trackers a los que nos conectamos para descargar el archivo.

No.	Time	Source	Destination	Protocol	Length	Info
17	7.472996655	172.20.60.104	150.214.204.10	DNS	82	Standard query 0x4631 A arenabg.com OPT
18	7.473211608	172.20.60.104	150.214.204.10	DNS	82	Standard query 0x525d AAAA arenabg.com OPT
19	7.473387137	172.20.60.104	150.214.204.10	DNS	91	Standard query 0x479d A internetwarriors.net OPT
20	7.473550722	172.20.60.104	150.214.204.10	DNS	91	Standard query 0x5c3a AAAA internetwarriors.net OPT
21	7.579674427	172.20.60.104	150.214.204.10	DNS	94	Standard query 0x2b12 A traCker.coPPeRsurFER.tk OPT
22	7.579896461	172.20.60.104	150.214.204.10	DNS	94	Standard query 0x54d6 AAAA traCker.COppErSurFER.tk OPT

Captura de Wireshark: Conexión con trackers.

F1.....arenabg.com.....).....F1.....arenabg.com.....+....).....

Captura de Wireshark: Intercambio de mensajes para la conexión al tracker “arenabg”. Estos mensajes se hacen con protocolo DNS, que usa UDP internamente.

```
GET /favicon.jpg HTTP/1.1
Host: arenabg.com
User-Agent: Transmission/2.92
Accept: */*
Accept-Encoding: gzip;q=1.0, deflate, identity

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Thu, 08 Nov 2018 09:35:28 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Keep-Alive: timeout=30
Location: https://arenabg.com/favicon.jpg

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

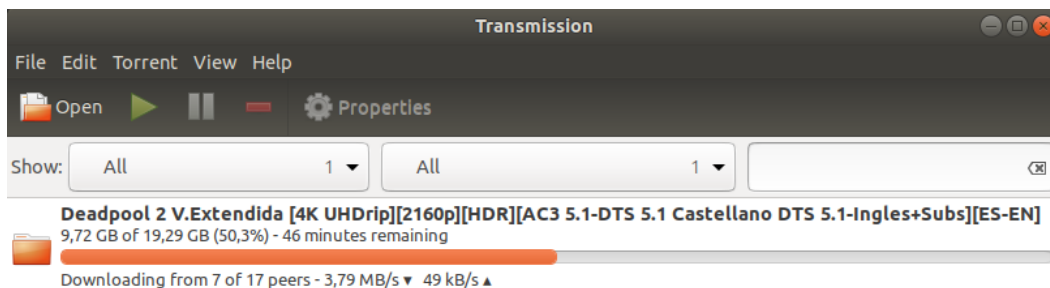
Captura de Wireshark: Usando el protocolo HTTP, nos comunicamos con el tracker.

```
$.s.+... .#f...0.....(w./...'.v.....{...5.=.....z.../...<.A...
...}...9.k.....|...3.g.E.....i.....arenabg.com.....
...
...
...http/1.1...p...l...N7.=.V..S..M5.O..Q.m.p.pc.. Xr`.2.0.f...KfZ..Z.[...a`_-...0..
$.http/1.1...
...
...0...0...0G..Q.g+6q...!..H.0
...*.H..
...0J1.0...U...US1.0...U..
...
Let's Encrypt1#0!..U...Let's Encrypt Authority X30..
1809270957472..
18122609574720.1.0...U...arenabg.com0..0
...*.H..
...0..
...!V..ys...
.vZ...p2>...+...
t4lm...>.eM.....g...&P..E..m...n..).C.q:...1.....}..."p0..7...
...f...4.F.N[....F6...!X...a..u..".TWt6-.c1.F..U.....<...Fv..j<...+...><)b...d...e5F+...a.7.....W...).
%1H.7.!M..A...R.....H.....0...0...U.....0...U...%.0...+.....0...U.....0...0...U.....m.2...[...".kk.].6.[
20...U.#.0...j3c.j)...9..Ee...00...+.....c0a0...0...0...0...http://ocsp.int-x3.letsencrypt.org0/...+.....0...http://cert.int-
x3.letsencrypt.org/0...U...0
..arenabg.com0...U...0...0...g...0...+.....0...0&...+.....http://cps.letsencrypt.org0...+.....0...This Certificate
may only be relied upon by Relying Parties and only in accordance with the Certificate Policy found at https://letsencrypt.org/
repository/0...
+.....y...V..t...t)...>qm...6..q..].07...d...f..w.....G0E...Ev...@K'*.({.....r.?.).>...!..EJ...4..4m..).K...|5W'..
%.8W...w.)<Q.T.9e..P.X...o.Xz)r.....EG.x...f..x.....H0F.!.....#W...ba;...5H[...6..^!.....6..d.....%..=#(y...>p.t...&0
...*.H..
...C...12...3.....*pu.....L.7.k0...
X...0...0...y...U.../b...e.....!..L...L...>.uH..V...{.Yci.
0.(rrl...(.Z...7.E..S.ND...p10n.4S...h...a.1.Yx...({.pJ...n.K...$...0<...q.q...4.b.Z.z...cX..h.....G..*.g..]
m=...4\5...0...0...Z...
..AB...S.sj...0
...*.H..
...071$0"...U..
..Digital Signature Trust Co.1.0...U...DST Root CA X30..
1603171640462..
2103171640462031.0...U...US1.0...U..
...
Let's Encrypt1#0!..U...Let's Encrypt Authority X30..0
...*.H..
...0..
...Z..G.r.]7..hc0..58%...5.p./...KA...5.X...*h...U...bq.y.'.....xqg.i.....<H..-..Mw.$..G.Z...7.....{...J..A..
6...m<h.#*B...tg...Ra..7e...V.....?.....k
...)+e...6u.k.J...IX/.0'%)..t..1..18...3.C...0...y1.-6...3j.91...
..d.3...))...0...y0...U...0...+.....s0q02...+.....0...&http://
isrg.trustid.ocsp.identrust.com0;...+.....0.../http://apps.identrust.com/roots/dstrootca3.p7c0...U.#.0...({...K..U...`...0T..U..
```

Captura de Wireshark: Vemos como, tras el saludo, el tracker nos envía “Let’s Encrypt”, para empezar a encriptar los mensajes y “..arenabg.com0...U. ...0..0...g...0...+.....0...0&...+.....http://cps.letsencrypt.org0...+.....0...This Certificate may only be relied upon by Relying Parties and only in accordance with the Certificate Policy found at https://letsencrypt.org/repository/0...”, donde le manda el nombre del tracker (arenabg) y su política. Esto se hace con TCP. Tras esto empieza la conversación.

4.2. Conexión con Peers

En las siguientes imágenes mostraremos la conexión con los peers y el intercambio de chunks con estos.



Captura de Transmission: Descargamos paquetes de 7 de los 17 peers disponibles. La velocidad de subida es 49 KBytes/s y la de bajada 3,79 MBytes/s.

Subida	Descarga	%	Indicadores	Dirección	Cliente
	2 kB/s	100 %	TDEH	90.173.145.63	µTorrent 5.3.3
		0 %	T?E	91.116.153.161	µTorrent 3.5.4
	59 kB/s	100 %	DuEX	91.117.83.127	BitSpirit 3.6.0
	122 kB/s	100 %	DE	92.185.94.130	µTorrent 3.5.4
	11 kB/s	100 %	TDE	92.190.8.150	µTorrent 3.5.4
40 kB/s	8 kB/s	17 %	TDUE	95.16.87.17	µTorrent 3.5.4
	3 kB/s	99 %	TDUEX	95.16.231.93	qBittorrent 4.1.3
		100 %	TDE	95.17.119.248	µTorrent 3.5.4
	11 kB/s	100 %	TDE	95.21.90.137	µTorrent 3.5.4
	11 kB/s	100 %	TDE	95.21.157.249	µTorrent 3.5.4
	9 kB/s	100 %	TDE	95.23.226.237	-BT7a4S-

Captura de Transmission: Los peer con los que nos estamos comunicando, nuestros “vecinos”. Por ejemplo, del peer 92.185.94.130 estamos descargando paquetes a una velocidad de 122 KBytes/s. Además estamos enviando chunks al peer 95.16.87.17 a 40 KBytes/s.

No.	Time	Source	Destination	Protocol	Length	Info
10761	13.298116	92.185.94.130	10.0.2.15	TCP	174	36373 > 52983 [PSH, ACK] Seq=40742 Ack=1
10762	13.298120	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=40862 Ack=1396 W
10763	13.298123	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=42282 Ack=1396 W
10764	13.298126	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=43702 Ack=1396 W
10765	13.298129	92.185.94.130	10.0.2.15	TCP	174	36373 > 52983 [PSH, ACK] Seq=45122 Ack=1
10766	13.326829	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=45242 Ack=1396 W
10767	13.326860	10.0.2.15	92.185.94.130	TCP	54	52983 > 36373 [ACK] Seq=1396 Ack=46662 W
10768	13.326932	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=46662 Ack=1396 W
10769	13.326942	92.185.94.130	10.0.2.15	TCP	134	36373 > 52983 [PSH, ACK] Seq=48082 Ack=1
10770	13.328191	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=48162 Ack=1396 W
10771	13.328212	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=49582 Ack=1396 W
10772	13.328217	92.185.94.130	10.0.2.15	TCP	1474	36373 > 52983 [ACK] Seq=51002 Ack=1396 W

Captura Wireshark: El peer del ejemplo anterior (el 92.185.94.130) nos está pasando paquetes TCP con muchos datos y nosotros le respondemos con paquetes de menor longitud.

A continuación vemos un trozo de una conversación de tamaño 1413129 bytes. Al principio hay un intercambio de información entre ambos peers y luego la fuente empieza a enviar chunks. Hay que tener en cuenta que el archivo que estamos descargando es un archivo de vídeo y el intercambio de mensajes está encriptado.

```
j...A.F.O...b...v.v...<.0.zR.0um...L./6..J.C4V..8QT...M13...)Y...<[...1...g.
%.&...".8;)+.J.16V...06.<
...S...[W.]}
$...8.o_-'...#...27?...19...|"h.6.....#...S...<
D.G..S.i.S..Y'.kI..
<...2..D7...E...{...u.W.95...0..!
o...|...V...6E..h.AV.7...;M...Tm;..E..C...Y;...].$"...09V...d
.w7...If...b...R..7y...C6..B9.g.$).7d
.g.w...;...S..UY.8...3T./*.q+..jBM.d...*l...8...h}')}
r..hTH...#...6.y...I...F...=dy...Au
l...=
q!...6'...K'...B.?.%.u.$
3.g..m.?.
...$.a.61...[...p.\.<z...m.
+.6~...*.C.g.#...|.TP...W.ro7-u6M...E...#R..k!...Mr6K...Y...g0.lu)...
$6..Y..SW.r>A.J...[k..tXU.V...o=T.D0...<...{jk7..
+&.q...!Y.J#...].S]2S.!...D\=u.2..7.sR...Y.%...#...%...@+
+eX..A...S...P...#..I..u.o...[Lu.%
ou.34)...X..W.G.F.FD].I...8J...J]~..(bxd..s...Q...P..
%.9..|vhl...B.
...p...K...b+a...G.f.*...Jl...4.*...7X...=...R.X...[...
.ug.h.Gp.^lO...Q...C.q.TN.q0o.4.7.+...eu...o.?.?@K..9..w...[...x<x-
s.V.J]*rz.YW.g...v...[...g...L...+sm...r..lw...fu...k...
[F.ng...X...<E'W...)...9H..HS.*...W.v...<.A.8..Q..5..L.N...J.
J..v.MsV...Y;2%V...
!...G..x..lb...=k):lA..z.7..4...u...\.0...q...|.../...=12.l...
+...bz).&.0.X.s3..@.l.../...0.t...
(.J.F...f.<...#0.N...t...Z...F...!P...J.v...J.(.b')...b0_?..
...i...:G..jP.^).L...%m.Y
/-..J[-HR..B.TG...ht...@>.(...".1z4...s.FMFx...
\...3vP.j...L...1.(.e..9H...ey..q...>.A*.da.o:...u.M2.hi...It.)S.7a...x.
\...4C.G.g...(.W.8].V.o.i..F...<...%|[6./8\B.1..u;E..d...>...X.../..L.V)%
```

Vemos el intercambio de información entre los peers. Lo que envía la fuente aparece en azul y lo que envía nuestro host en rojo.

```
...01.. fe..%..r7.I2.....j}wu3.EV(-n$P.....
{L=.S..a0)...Cr..>.7.LRl.l..3a.....i..a>...1...7.^...2.
Fv.....F...A.)..d...4..v9U.
..w...<oH.v...Y..C...u...q...$.4..=..S.C.'Vk..$...)%?.k..
..*y..$.Xt..5Lb.(./r.
...P...$.../..oP4...fu...gb..S\*/t...Jzo.....H.../?..?
9;...k.8.Ot-D...^...b.v...t87x.w...qU.LU..DF...U;.....0
[I...Z..|w.d7.q6g..F.C.7.?..+d.s!..0.?A...P...V..jN"...j...F...;..h
{rR...8...Q..h2..."..7-B.45...q...H*...).X5..C<(\.b...|d.t)*
{...@Y.6^@r..aA..*9%..Fg...$.+..qK'F7.
{6'..9<...b..m...Z..jB..5..Z...}|.o8.....AC.....4...|.'w.v6P.W.Y.Y>Jz
...R.Z...2...t...@..6m.&..\C6p..]s+c...*Ot.X.A.N...).l..hvk:U...^..0...63Y..
})...
i...sv>...e...s.R.;IKU...d+...9-*...X.c:0"...-k.+!
yn..A.#...C..h...MIV...9...+n.mu0...'.1..1.K.
R...7...00.Z...3..g..h]nw...*#u.Q|.5...}R.
p00...p:p:z>5...@...E..k..
$.N...b.i7.z.f...q...o...&.v...P.x0.*>.RV;t.Kq...1r...ey..i...m~.
[.6%...].M..).M./...d...{...gN<)<...W.....
..
h*~p...L...K..>...c+;W..N...i..j..~.DT.?[...s.%sw;...>...
RR.2...%...YT...s...;...'d.p.*.f.=...0..X..7.cj.DhU.);WA.rl
\...w...F...m..d..RO'g..k..$.>..E.y5..d..0../...n_vb...E|.+.++..
$.<...6..*R..#...[q.P.p.<..U...
$.%x;s...GbK./fbz...].b...0.R...\.=...od7.../..HT..%K...&.jkh.
|.l..MLy..V...M..m/II..o]
661C..S...0...A...5).3...0V..w7..*..@...eN^..?..^480..."6.....u.8..."P^
K..5A0.*.=.Rsk.D{...C.Z...G.UV>..!^
...3:7...<k.J...n...\.On.#..T.Ua.5!..bN.r..*..eR...Z.f]PfK
%.BL.SW...08Lc.>..H4...f...L.../...1...W...fOD.K.&..;H..0.g
S.N..N..Y...J.X.A.P..ETx...3^...7.I.Q...N.#.Kj.i.i...>.v.KLR..=
\...e..7...wA.N...QM./I..)D]
```

El resto de la conversación aparece en azul hasta completar los 1413129 bytes.

Para obtener un análisis secuencial de las conexiones TCP, consultamos el gráfico de flujo TCP:

Time	10.0.2.15	89.29.134.226	Comment
16,154	(33226)	SYN → (10637)	Seq = 0
16,186	(33226)	← SYN ACK (10637)	Seq = 0 Ack = 1
16,186	(33226)	ACK → (10637)	Seq = 1 Ack = 1
16,653	(33226)	PSH,ACK-Len: 547 → (10637)	Seq = 1 Ack = 1
16,653	(33226)	← ACK (10637)	Seq = 1 Ack = 548
16,687	(33226)	PSH,ACK-Len: 93 → (10637)	Seq = 1 Ack = 548
16,687	(33226)	← ACK (10637)	Seq = 548 Ack = 94
16,718	(33226)	PSH,ACK-Len: 85 → (10637)	Seq = 94 Ack = 548
16,718	(33226)	← ACK (10637)	Seq = 548 Ack = 179
17,156	(33226)	PSH,ACK-Len: 124 → (10637)	Seq = 548 Ack = 179
17,157	(33226)	← ACK (10637)	Seq = 179 Ack = 672
17,281	(33226)	PSH,ACK-Len: 98 → (10637)	Seq = 179 Ack = 672
17,281	(33226)	← ACK (10637)	Seq = 672 Ack = 277
17,311	(33226)	PSH,ACK-Len: 383 → (10637)	Seq = 277 Ack = 672
17,311	(33226)	← ACK (10637)	Seq = 672 Ack = 660
20,164	(33226)	PSH,ACK-Len: 780 → (10637)	Seq = 672 Ack = 660
20,164	(33226)	← ACK (10637)	Seq = 660 Ack = 1452
22,146	(33226)	PSH,ACK-Len: 7 → (10637)	Seq = 660 Ack = 1452

Nótese que las tres primeras líneas muestran el establecimiento de la conexión: “SYN”, “SYN ACK”, “ACK”, lo que es llamado 3-Way Handshake

Time	10.0.2.15	89.29.134.226	Comment
34,729	(33226)	ACK - Len: 1420 (10637)	Seq = 819448 Ack = 3619
34,729	(33226)	ACK - Len: 1420 (10637)	Seq = 820868 Ack = 3619
34,729	(33226)	ACK - Len: 1420 (10637)	Seq = 822288 Ack = 3619
34,729	(33226)	ACK → (10637)	Seq = 3619 Ack = 823708
34,729	(33226)	ACK - Len: 1420 (10637)	Seq = 823708 Ack = 3619
34,729	(33226)	ACK - Len: 1420 (10637)	Seq = 825128 Ack = 3619
34,731	(33226)	ACK - Len: 1420 (10637)	Seq = 854948 Ack = 3619
34,731	(33226)	ACK → (10637)	Seq = 3619 Ack = 857788
34,732	(33226)	ACK → (10637)	Seq = 3619 Ack = 857788
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 857788 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 859208 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 860628 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 862048 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 863468 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 864888 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 866308 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 867728 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 869148 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 870568 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 871988 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 873408 Ack = 3619
34,732	(33226)	ACK → (10637)	Seq = 3619 Ack = 874828
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 874828 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 876248 Ack = 3619
34,732	(33226)	ACK - Len: 1420 (10637)	Seq = 877668 Ack = 3619

La fuente envía chunks al host y el host confirma que los recibe

5. Referencias

- <https://en.wikipedia.org/wiki/BitTorrent>
- Jim Kurose, Keith Ross, “Computer Networking: A Top-Down Approach”, Pearson, 2013, pp. 144-151.
- Bram Cohen, “Incentives Build Robustness in BitTorrent”, 2003.
- <https://wiki.wireshark.org/BitTorrent>
- https://en.wikipedia.org/wiki/UDP_tracker