BitTorrent协议规范

维基教科书，自由的教学读本

BitTorrent是由布莱姆·科恩设计的一个点对点（P2P）文件共享协议，此协议使多个客户端通过不可信任的网络的文件传输变得更容易。

目录

目的

此规范的目的是详细介绍 BitTorrent 协议规范 v1.0 。Bram 的协议规范网站 http://www.bittorrent.com/protocol.html 简要地叙述了此协议，在部分范围缺少详细行为阐述。希望此文档能成为 一个正式的规范，明确的条款，将来能作为讨论和执行的基础。

此文档规定由 BitTorrent 开发者维持和使用。欢迎大家为它做贡献，其中的内容代表当前协议，它仍由许多客户使用。

这里不是提出特性请求的地方。如果有请求，请见邮箱列表。

应用范围

本文档适用于 BitTorrent 协议规范的第一版（v1.0）。目前，这份文档应用于 torrent 文件结构、用户线路协议和服务器（Tracker）HTTP/HTTPS 协议规范。如果某个协议有了新的修订，请到对应页面查看，而不在这里。

约定

在本文档中，使用了许多约定来简明和明确地表达信息。

- 用户（peer）v/s 客户端（client）：在本文档中，一个用户可以是任何参与下载的 BitTorrent 客户端。客户端也是一个用户，尽管 BitTorrent 客户端运行在本地机器上。本规范的读者可能会认为自己是连接了许多用户的客户端。
- 片断（piece）v/s 块（block）：在本文档中，片断是指在元信息文件中描述的一部分已下载的数据，它可通过 SHA-1 hash 来校验。而块是指客户端向用户请求的一部分数据。两块或更多块组成一个完整的片断，它能被校验。
- 实际标准：大的斜体字文本指出普通的准则在不同客户端 BitTorrent 的执行，它被当作为实际标准。(对照英文原文，common应该翻译成通用或者常见，这句话的大概意思是一个规范由于被许多不同的BitTorrent客户端实现所通用，以至于被当做是实际标准)

为了帮助其他人找到本文档最近的修改，请填写改变日志（最后一段）。它应包含一个简短的项目（如：一行），用来记录你每次对此文档的主要改动。

B编码

B编码是一种以简洁格式指定和组织数据的方法。支持下列类型：字节串、整数、表和字典。

字节串

字节串按如下编码：<以十进制 ASCII 编码的串长度>：<串数据>
注意没有开始和结束的分隔符。 例：“4：spam” 代表字符串“spam”

整数

整数按如下编码：i<以十进制 ASCII 编码的整数>e
开始的“i”与结尾的“e”分别是开始和结束分隔符。可以使用如“i-3e”之类的负数。但不能把“0”放到数字的前面，如“i04e”。另外，“i0e”是有效的。
例：“i3e”代表整数“3”

表

表按如下编码：l<编码值>e
开始的“l”与结尾的“e”分别是开始和结束分隔符。 表可以包含任何已编码的类型，包括整数、串、字典和其他的表。
例：l4:spam4:eggse 代表两个串的表“spam”、“eggs”

字典

字典按如下编码：d<编码串><编码元素>e
开始的“d”与结尾的“e”分别是开始和结束分隔符。 注意关键字必须被编码为串。值可以是任何已编码类型，包括整数、串、表和其他字典。关键字必须是串，以分类的顺序出现（以原始串排列，而不是以字母数字）
例1：d3:cow3:moo4:spam4:eggse 代表字典 { "cow" => "moo", "spam" => "eggs" }
例2：d4:spaml1:a1:bee 代表字典 { "spam" => ["a", "b"] }

元信息文件结构

所有在元信息文件中的数据都要编码。编码规则如上所述。

元信息文件（以 .torrent 结尾的文件）的内容是一个编码的字典，包含以下列表中的各项。所有字符串值都以 UTF-8 编码。没有“可选”标记的键值是必需的字段：

- **信息**

一个描述 torrent 文件的字典。有两种可能的形式：一种是没有目录结构的“单一文件”，另一种是包含子目录树的“多文件”

对于“单一文件”来说，信息字典包含以下的结构：

长度：文件字节数长度（整数）

md5和:（可选）一个 32 位的 16 进制字符串，它对应于文件的 MD5和。不被 BitTorrent 所使用，但被一些程序包含，以提供更大的兼容性。

名称：文件的名称。建议使用（字节串）。

片断长度：每个片断的字节数（整数）。

片断：包含所有 20 字节 SHA-1 散列值的字符串，每个片断都有唯一的值。（字节串）

对于“多文件”来说，信息字典包含以下的结构：

文件：字典列表，每个文件都有一个。每个在表中的字典包含以下键值：

长度：文件长度的字节数（整数）

md5和:（可选）一个 32 位的 16 进制字符串，它对应于文件的 MD5和。不被 BitTorrent 所使用，但被一些程序包含，以提供更大的兼容性。

路径：一个包含着一个或多个字符串元素的，它包含路径和文件名。每个表中元素对应于一个目录名或（在最后的元素的情况下）文件名。
例：文件名“dir1/dir2/file.ext”将包含三种串元素：“dir1”、“dir2”和“file.ext”。编码为串表的例子“l4:dir14:dir28:file.exte”

名称：结构中根目录的名称——包含上述文件列表中所有文件的目录（字符串）

片断长度：每个片断的字节数（整数）。

片断：包含所有 20 字节 SHA-1 散列值的字符串，每个片断都有唯一的值。（字节串）

- **发布**：服务器的发布 URL（字符串）
- **发布列表**：（可选）这是官方规范的一个扩展，它是向后兼容的。此键值用来执行备份服务器的列表。完整的规范可在 http://home.elp.rr.com/tur/multitracker-spec.txt 找到。

- **创建日期**：（可选）torrent 文件的创建时间，使用标准 Unix 时间格式（从 UTC 1970年1月1日 00：00：00 开始，整数秒）
- **评论**：（可选）发布者的自由评论（字符串）
- **由……创建**：（可选）创建 torrent 文件的名字和程序版本（字符串）

注意：

- **片断长度**指定了标准的片断大小，通常是 2 的 n 次方。片断长度的一般是根据 torrent 文件中所有数据的数量来决定的，如果片断太大，会导致效率低，出错概率增加；而如果太小，则会使生成的 torrent 元数据文件过大。常识决定使用最小的片断大小，这样就会使生成的 torrent 文件不大于 50－75 KB（可以减轻存储 torrent 文件服务器的负担）。但是，由于没有严格限制存储和带宽，即使为了高效率的共享文件可能导致生成更大的 torrent 文件，也建议将小于 8－10 GB 文件的片断大小设为小于或等于 512 KB。通常大小是 256 KB,512 KB 和 1 MB。除了最后的片断大小不定以外，其余片断大小是相等的。因此片断的数目由总大小决定。对于多文件模式下的片断边界，将文件数据设想为一个长的连续流，由文件有序列表中的每个文件相互连接而成。片断数目和其边界的决定方式与单一文件相同。片断可能由两个文件的边界组成。
- 每个片断都有相应的 SHA-1 hash 数据校验码。这些校验码相互连接形成上述的信息字典的片断值。注意这不是一个表，而是一个字符串。其长度必须是 20 字节的整数倍。

服务器 HTTP/HTTPS 协议

服务器是用类响应 HTTP GET 请求的一种 HTTP/HTTPS 服务。该请求包括客户端的度量标准，这个标准可以帮助服务器全面统计 torrent 文件。基本的 URL 包括元数据文件（torrent）中定义的“发布 URL”。再将那些参数通过标准 CGI 方法添加到此 URL 中（如：“？”在发布 URL 之后，紧接着“参数=值”的序列，分隔符“&”）

注意所有在 URL 中的二进制数据（特别是 info_hash 和 peer_id）必须使用转义符。这意味着除 0-9，a-z，A-Z和$-_.+!*'() 外，其余字节需要采用“%nn”格式的编码，其中的“nn”是字节的 16 进制数值。（详细见 RFC1738）

客户端向服务器的 GET 请求的参数如下：

- **info_hash**：元信息文件中 20 字节的 SHA-1 散列值。注意此值会进入编码字典中，如上述的信息关键字的定义所述。与不需编码的 peer_id 相比，它总是被 URL 编码。
- **peer_id**：客户端 ID ，客户端用来唯一标识自己 ID 的 20 字节的串，它在客户端启动时生成。允许为任何值，包括二进制数据。目前没有特定的算法来生成客户端 ID。但是，人们会认为它至少对于自己的本地机器是唯一的，从而应该像进程 ID 一样合并数据，也可能在启动时由时标记录。见本区域下面的一般客户端编码的 peer_id。
- **端口**：客户端监听的端口号。BitTorrent 所使用的典型端口是 6881-6889。如果此范围的端口都无效，可以选择其他的。
- **已上传的**：从客户端发送“已开始”事件到服务器算起的上传总量，数值采用 10 进制的 ASCII。对于没有在官方规范明确指出的，该值应为已上传的字节总数。
- **已下载的**：从客户端发送“已开始”事件到服务器算起的下载总量，数值采用 10 进制的 ASCII。对于没有在官方规范明确指出的，该值应为已下载的字节总数。
- **剩下的**：客户端需要下载的字节数，以 10 进制 ASCII 编码。
- **紧密的**：客户端接受一个紧密的响应。客户端列表由客户端串代替，此串中每个客户端都编码成 6 字节。前 4 字节是主机名（以网络的字节顺序），后两个字节是端口号（同样以网络字节的顺序）。
- **事件**：如果被指定，则是**已开始**，**已完成**，**已停止**中的一个，或者为空（表示未指定）。如果未指定，此请求为常规时间间隔中的一次运行。
  - **已开始**：向服务器发送的第一个请求，必须包含开始值的事件关键字。
  - **已停止**：如果客户端关机则须发送到服务器上。

  - **已完成**：完成下载时必须发送到服务器上。但是，当客户端启动时下载完成度为 100%（即：做种中）则不会发送。可能这是允许服务器增加“已完成下载”的方法。
- **ip**：可选。客户端的真实 IP 地址，以点分四元组格式或 RFC3513 中定义的 16 进制 IPv6 地址。注意：大体上此参数没有客户端地址重要，它能由 IP 地址决定，HTTP 请求也来自该处。仅在请求参与的 IP 地址不是客户端的 IP 地址的情况下才需要。这种情况发生在客户端通过代理服务器与服务器进行通信的情形。当客户端和服务器同时处在本地 NAT 网关时也需要。原因是服务器会发出客户端的内部地址（RFC1918），这是不可到达的。所以客户端必须清楚地把自己的外部可到达的 IP 地址发送到其他客户端中。不同的服务器对此参数的解释有所不同。某些只有当请求参与的 IP 地址属于 RFC1918 时才允许。有些无条件允许，但有些则完全忽略。如果使用 IPv6 地址（如：2001:db8:1:2::100），则表示客户端能通过 IPv6 进行通信。
- **需求数目**：可选。客户端想从服务器接收的用户数目。允许此值为“0”。如果不用此项，则默认值为 50 个用户。
- **关键字**：可选。一个不与任何用户共享的另外的标识。当 IP 地址改变后，允许客户端证明它们的标识。
- **服务器id**：可选。如果先前发布包含服务器的 id，它应放在这里。

服务器作出“text/plain”文档的响应包括以下编码字典的关键字：

- **失败原因**：如果当前使用此值，则其余关键字不会使用。该值是可读的错误消息，包括请求失败的原因。（字符串）
- **警告消息**：（新）与失败原因相似，但响应仍然会被正常处理。警告消息看起来像错误。
- **时间间隔**：以秒计算，是客户端发送规则请求到服务器之后等待的时间。（强制）
- **最小时间间隔**：最小发布时间间隔。当前客户重发间隔不能小于此值。
- **服务器 id**：一个客户端应在下一个通告发回的字符串。如果没有该值，先前通告会发出一个服务器 id ，不要丢弃旧的值，一直使用它。
- **完成**：拥有完整文件的用户数，即做种者（整数）
- **未完成**：非种子用户的数目，也叫“吸血者”（整数）
- **用户**：是字典的列表，每个值都有如下的关键字：
  - **用户 id**：用户的自选择 ID，如上述用来发送服务器请求的（字符串）
  - **ip**：用户的 IP 地址（IPv4 或 IPv6 格式）或域名（字符串）
  - **端口**：用户的端口号（整数）

如上所述，用户列表长度默认值为 50。如果连接的用户少于该值，列表会更小。另外，服务器随机选择用户及其响应。服务器在响应请求时可能使用一个更智能的机构来选择用户。例如，应避免向其他做种者报告种子。

在事件发生（即：已停止或已完成）或客户端需要连接更多的用户时，客户端向服务器发送请求的间隔可以低于指定的时间间隔。但是，为了获得更多的用户而向服务器频繁地请求会被认为是错误的行为。如果客户端想在回应中得到许多用户，则需要在“需求数目”参数中设定。

使用者注意：30 个用户就算丰富的源了，官方客户端版本 3（v3）实际上在连接数少于 30 时会尝试增加新的连接，当连接数大于或等于 55 时会拒绝连接多余的用户。这个值对性能很重要。当完成下载一个新的片断时，“已拥有”消息（见下面）将会发送到最活动的用户。结果广播通信量与用户数目成正比例增加。大于 25 时，新用户不太可能会增加下载速度。有人强烈建议用户界面设计者使该项模糊和很难修改，因为那样做几乎没有用。

服务器“刮”约定

根据惯例，多数服务器支持请求的另一种形式，这种方式询问给定的服务器正在处理的 torrent（或所有的 torrent）。通常叫做“刮页”，因为它自动处理“刮屏”（服务器统计页）冗长的部分。刮 URL 也是一种类似于上面描述的 HTTP GET 方法。但基本 URL 不同。用以下步骤来得到刮 URL：从发布 URL 开始寻找其中最后一个“／”。如果在文本之后的“／”不是“announce”，它将被作为一个符号，此符号不支持刮约定。如果是，则以“scrape”代替“announce”来找到刮页。

例：（发布 URL -> 刮 URL）

```
http://example.com/announce          -> http://example.com/scrape
http://example.com/x/announce        -> http://example.com/x/scrape
http://example.com/announce.php      -> http://example.com/scrape.php
http://example.com/a                 -> (不支持刮)
http://example.com/announce?x=2%0644 -> http://example.com/scrape?x=2%0644
http://example.com/announce?x=2/4    -> (不支持刮)
http://example.com/x%064announce     -> (不支持刮)
```

特别注意：结束引语没有完成。此标准是由 Bram 在 BitTorrent 开发列表文件 (http://groups.yahoo.com/group/BitTorrent/message/3275)中说明的。

刮 URL 可以作为可选参数“info_hash”的一个补充，是一个 20 字节的值。这限制了服务器向特殊种子汇报。另外，服务器正在处理的所有种子的统计也被发回。为了降低服务器的负载和带宽，有人强烈建议软件作者尽可能使用“info_hash”参数。

HTTP GET 方法的响应是一个由编码字典组成的“text/plain”文档，包括以下关键字：

- **文件**：每个 torrent 都包含一对关键字的字典，内容是统计数据。如果添加了有效的“info_hash”，此字典将包含单个关键字。每个关键字由一个 20 字节的 info_hash 值组成。该关键字的值是另一个包含下列名称的嵌套字典：
  - **已完成**：拥有整个文件的用户数目，即做种者（整数）
  - **已下载**：服务器注册编号完成的总数（“事件＝完成”，即客户端完成下载 torrent）
  - **未完成**：非做种者用户的数目，也叫“吸血者”（整数）
  - **名称**：（可选）torrent 的内部名称，由 torrent 文件中信息字段指定

注意此响应有三层字典嵌套。例如：

d5:*files*d20:.....d8:*complete*i5e10:*downloaded*i50e10:*incomplete*i10eeee

其中“.....”是 20 字节的 info_hash，以上表明有 5 个做种者，10 个吸血者和 50 个完成下载的用户。

用户线路协议（TCP）

概述

用户线路协议使元信息文件中片断的交换变得更容易。

注意原始规范在描述用户协议时也使用术语“片断”，但与元信息文件中的术语“片断”不同。由于该原因，术语“块”将在本规范中用来描述用户之间通过线路交换的数据。

客户端必须为每个远程用户的连接保持状态信息：

- **被阻塞**：远程用户是否阻塞此客户端。当用户阻塞客户端时，不会响应任何请求。客户端不应尝试发送请求块，所有未响应的请求会被远程用户丢弃。
- **感兴趣**： 远程用户是否对此客户端感兴趣。当客户端未阻塞时，远程用户将开始发送请求块。

注意这也意味着客户端需要记住自己是否对远程用户感兴趣和阻塞它。因此，真正的列表看起来像这样：

- **am_choking**：此客户端阻塞远程用户
- **am_interseted**：此客户端对远程用户感兴趣
- **peer_choking**：远程用户阻塞此客户端
- **peer_interested**：远程用户对此客户端感兴趣

客户端的连接以“被阻塞”和“不感兴趣”开始。也就是：

- **am_choking** = 1
- **am_interested** = 0
- **peer_choking** = 1
- **peer_interested** = 0

当客户端对远程用户感兴趣并且远程用户未阻塞该客户端时，客户端开始下载块。当客户端没有阻塞远程用户并且远程用户对该客户端感兴趣时，客户端开始上传块。

客户端保持通知远程用户自己是否对它们感兴趣，这是很重要的。与每个远程用户连接的状态信息应保持最新，直到该客户端被阻塞。这允许远程用户知道当自己未阻塞时，客户端是否会开始下载；反之亦然。

数据类型

如果不特别指定，在用户线路协议中的所有整数都会编码成 4 字节 big endian 值。这包括所有消息中的长前缀，它在握手之后。

消息流

用户线路协议包括初始握手。之后，用户通过长前缀消息的交换来进行通信。长前缀是上述的一个整数。

握手

握手是必需的消息，它一定是由客户端发送的第一条消息。

- **握手**：<pstrlen><pstr><reserved><info_hash><peer_id>
  - **pstrlen**：<pstr>串的长度，作为单个原始字节
  - **pstr**：协议的串标识符
  - **reserved**：8 个保留字节。当前的执行使用全〇。字节中的每位可以用来改变协议的行为。一封 Bram 的电子邮件建议首先使用末位，以使首位可用来改变末位的含义。
  - **info_hash**：元信息文件信息关键字的 20 字节 SHA-1 散列值。这与服务器请求中发送的 info_hash 意义相同。
  - **peer_id**：每个客户端用来作为唯一标识的 20 个字节的串。这与服务器请求中发送的 peer_id 意义相同。

在 BitTorrent 协议 v1.0 中：pstrlen=19，pstr="BitTorrent protocol"

连接的发起者应该立即发送彼此的握手信息。即使接收者能同时提供多个 torrent（torrent 通过自己的 info_hash 来唯一标识），它也应等待发起者的握手信息。但是，接收者在看到握手的 info_hash 部分后必须迅速回应。服务器的 NAT 检查特性不会发送握手的 peer_id 字段。

如果客户端收到一个当前不能处理的握手 info_hash，该客户端就会断开那个连接。

如果连接的发起者收到一个握手信息，其中的 peer_id 与预期的不同，那么发起者就会断开该连接。注意发起者可能会收到来自服务器的远程用户信息，它包括远程用户注册的 peer_id。来自服务器的 peer_id 与握手信息中的应该相同。

peer_id

主要有两种将客户端及其版本信息编码到 peer_id 的方法：Azureus 型和 Shadow 型。

Azureus 型使用如下编码：“-”，一个客户端标识使用两个字符，版本号用 4 个 ASCII 数字表示，“-”紧跟在随机数字之后。

例如：'-AZ2060-'...

已知采用这种方法编码的客户端是：

- 'AR' - Arctic (http://dev.int64.org/arctic.html)
- 'AZ' - Azureus (http://azureus.sf.net/)
- 'BB' - BitBuddy (http://www.btvampire.com/)
- 'BC' - BitComet (http://www.bitcomet.com/)
- 'BS' - BTSlave (http://btslave.sourceforge.net/)
- 'BX' - Bittorrent X
- 'CD' - Enhanced CTorrent (http://www.rahul.net/dholmes/ctorrent/)
- 'CT' - CTorrent (http://ctorrent.sourceforge.net/)
- 'LP' - Lphant (http://www.lphant.com/)
- 'LT' - libtorrent (http://libtorrent.sf.net/)
- 'lt' - libTorrent (http://libtorrent.rakshasa.no/)
- 'MP' - MooPolice (http://www.moopolice.de/)
- 'MT' - MoonlightTorrent (http://www.moonlighttorrent.com/)
- 'QT' - Qt 4 Torrent example
- 'RT' - Retriever (http://www.halogenware.com/software/retriever.html)
- 'SB' - Swiftbit
- 'SS' - SwarmScope
- 'SZ' - Shareaza (http://shareaza.sourceforge.net/)
- 'TN' - TorrentDotNET
- 'TR' - Transmission (http://transmission.m0k.org/)
- 'TS' - Torrentstorm (http://www.torrentstorm.com/)
- 'UT' - µTorrent (http://www.utorrent.com/)
- 'XT' - XanTorrent (http://www.xantorrent.pwp.blueyonder.co.uk/xantorrent.zip)
- 'ZT' - ZipTorrent (http://www.ziptorrent.com/)

Shadow 型采用如下编码：用 1 个 ASCII 字母或数字标识客户端，3 个 ASCII 数字标识版本号，“——”紧跟在随机数字之后。

例如：'S587——'...

已知采用此种类型编码的客户端为：

- 'A' - ABC (http://pingpong-abc.sourceforge.net/)
- 'O' - Osprey Permaseed (http://osprey.ibiblio.org/)
- 'R' - Tribler (http://www.tribler.org/)
- 'S' - Shadow's client (http://bt.degreez.net/)
- 'T' - BitTornado (http://bittornado.com/)
- 'U' - UPnP NAT Bit Torrent

Bram 的客户端现在采用的形式： 'M3-4-2--'

BitComet则不同。它的 peer_id 由四个 ASCII 字符组成“exbc”，后面是两个字节的“x”和“y”，之后才是随机字符。版本号“x”是在小数点前的十进制数值，“y”是小数点之后的两个十进制数值。BitLord (http://www.bitlord.com/) 使用相同的结构，但在版本号后添加“LORD”字符。一个 BitComet 的非官方补丁 (http://solidox.org/bc/)将“exbc”替换成了“FUTB”。BitComet 用户标识的编码在 0.59 及其以前的版本都采用的 Azureus 型。

XBT Client (http://xbtt.sourceforge.net/client/)也有自己的风格。其 peer_id 由三个大写字母“XBT”紧跟三个代表版本号的 ASCII 数字。如果客户端处在调试阶段，第七字节则是小写字母“d”，其他情况下是“-”。之后是“-”和随机数字，大写和小写字母。例：'XBT054d-'表示 0.5.4 版的开始调试阶段。

Opera 8 previews (http://snapshot.opera.com/)采用以下结构：前面两个字符“OP”紧跟四个相等的构建号。之后所有字符都是随机小写 16 进制的数字。

Bits on Wheels (http://www.bitsonwheels.com/)采用格式“-BOWAxx-yyyyyyyyyyyy”，其中“y”是随机大写字母，x 取决于版本。版本 1.0.6 中 xx＝0C。

许多客户端使用随机数字或 12 个〇紧跟着随机数字（如以前旧版本 (http://www.bittorrent.com/)的 Bram 客户端）。

消息

协议中所有发送的消息均采用“<长前缀><消息标识符><有效负载>”的形式。长前缀是一个 4 字节 big endian 值。消息标识符是一个十进制字符。有效负载是消息依赖的。

keep-alive: <len=0000>

“保持活动”消息是由〇组成的字节串，将长前缀设为〇可指定。没有消息标识符和有效负载。如果在一段时间内，用户没有收到消息，它会断开连接，所以需要发送活动消息来维持连接。一条保持活动消息大概每两分钟发送一次。

choke: <len=0001><id=0>

阻塞消息是定长的，无有效负载。

unchoke: <len=0001><id=1>

未阻塞消息是定长的，无有效负载。

interested: <len=0001><id=2>

感兴趣消息是定长的，无有效负载。

not interested: <len=0001><id=3>

不感兴趣消息是定长的，无有效负载。

have: <len=0005><id=4><piece index>

拥有消息是定长的。有效负载是刚成功下载和通过散列值校验的〇基片段的索引。

使用者注意：这是严格的定义，实际上会用到某些游戏程序。特别地，因为用户极不可能下载已经拥有的片断，用户可能不会选择向另一个用户宣布已拥有那个片断的消息。少量“拥有抑制”会导致拥有消息减少一半，在协议前面将转化到 25-35%。

恶意用户可能会选择宣布拥有别人永远不会下载的片断。因此，向普通用户发送此信息是坏主意。

bitfield: <len=0001+X><id=5><bitfield>

位字段消息可能在握手序列发送完成后，在其他任何消息发送之前立即发送。它是可选的，如果客户端没有此片断则不必发送。

位字段消息是变长的，其中的“X”是位字段的长度。有效负载是代表成功下载片断的位字段。首字节的高位对应片断索引 0。已清除的位则指出缺少的片断，通过一个有效可用的片断来设置位。末位剩下的位设置为〇。

一个错误长度的位字段被认为是错误。客户端在收到不正确大小的位字段或已有设置为剩下位的位字段时应断开连接。

request: <len=0013><id=6><索引><开始><长度>

请求消息是定长的，用来请求块。有效负载包含以下信息：

索引：指定〇基片断索引的整数。

开始：指定片断内〇基字节的偏移量整数。

长度：指定被请求长度的整数。

根据官方规范有关主要版本3，“所有当前执行应使用 2^15（32 KB），请求数量大于 2^17（128 KB）时应断开连接。”在主要版本4中，此反应修改到了 2^14（16 KB），超过该值的用户会强迫拒绝。注意到块请求小于片断大小（>=2^18 字节），所以为下载一个完整片断需要多次请求。

由于新版本将限制定在 16 KB，尝试使用 32 KB 的块就好比用 4 发子弹来玩俄式轮盘——会遇到困难。更小的请求会导致更大的系统时间和空间开销，因为要跟踪很多请求。结果应使用所有客户端都允许的 16 KB。

请求块大小的限制执行的选择没有减少一部分清楚。在主要版本 4 中，强制使用 16 KB 的请求，许多客户端会使用该值，只有一个严格客户端组不会使用。大多数旧客户端使用 32 KB 请求，不允许明显减少可能用户的批次。同时 16 KB 是现在部分官方的限制（“部分”是因为官方协议文档没有更新），所以强制使用没有错。另外，允许更大的请求增大了可能用户的批次，除在非常低的带宽连接（小于 256 kbps）中，多个块会在一个阻塞周期内完成下载，从而强迫使用旧的限制仅会降低很少的性能。因此，推荐仅在旧的 128 KB 下才强行限制。

piece: <len=0009+X><id=7><索引><开始><块>

片断消息是变长的，其中的“X”是块长度。有效负载包含以下信息：

索引：由〇基片断索引指定的整数

开始：由片断内〇基字节偏移量指定的整数

块：数据块，是索引指定片断的子集。

cancel: <len=0013><id=8><索引><开始><长度>

取消信息是定长的，用来取消块的请求。有效负载与“请求”消息相同。典型用在“最后阶段”中。（见下面的算法段）

port: <len=0003><id=9><listen-port>

端口消息是由运行 DHT 服务器的新版本的主要部分。监听端口是用户 DHT 节点正在监听的端口。如果 DHT 服务器支持，则应把此用户加入本地的路由表中。

算法

排队

通常建议用户在每个连接上保持一些未完成的请求。因为从一块的下载到开始下载另一块需要一个完全的往返程（往返程在片断消息和下一个请求消息之间）。一旦与高 BDP（Bandwidth Delay Product，高延迟或带宽）相连，会降低很多性能。官方客户端未完成请求的默认值是 5。

用户注意：这是最严格的性能条款。一个 5 请求的静态队列对于具有 50 ms 延迟 5 Mbps 中的 32 KB 块是合理的。连接更大的带宽越来越常见，所以用户界面设计者被催促使其对于改变更适用。特别地，线缆调制解调器以调整通信量和增加其可能缓和部分由此导致的问题，这是众所周知的。

自动调整：调整此参数的一个合理的方法是连续测量单个连接的带宽。如果该用户增加带宽而队列不够，则尝试增加队列长度。使用相同标志如果减少队列长度没有减少带宽和延迟，可能是队列长度太大了。

超级种子

（该项不是原始规范的一部分）

在 S-5.5 中的超级种子特性是一种新的做种算法，用来帮助只有有限带宽的做种者发布很大的文件，减少为了产生新的种子而上传的数据总量。

当做种者使用“超级种子模式”时，它不会作为标准种子，而是伪装成一个没有数据的普通客户端。当客户端连接时，它会通知它们自己收到一块从未发送或源很少的片断。这将促使客户端仅尝试下载那个片断。

当客户端完成下载该片断时，做种者看到它以前发送的片断在其他用户中至少有一个拥有后，才会继续发送另外的片断。在那之前，客户端下载不到做种者的其他片断，这样不会浪费做种者的带宽。

这种方法会有更高的效率，同时促使用户只下载源最少的数据，降低了多余数据的发送，限制了没有为该群传输数据的用户而发送的数据量。在这之前，做种者可能需要上传文件总大小的 1.5 到 2 倍，其他用户才可能成为种子。但是，使用超级种子模式的单个客户端发布大的文件只需上传文件大小的 1.05 倍就能成为种子。这是标准做种效率的 1.5 到 2 倍。

不推荐一般用户采用超级种子模式。虽然它有助于稀少数据的扩散，但是它限制了客户端下载片断的选择，同时限制了客户端下载自己已得到部分的片断。所以，超级种子模式只推荐原始做种者使用。

综上，“原始做种者模式”或“发布者模式”是更恰当的名称。

片断下载策略

客户端可以随机顺序下载片断。

更好的方法是首先下载源最少的片断。客户端可以从每个其他用户保存的原始位字段来决定，通过拥有消息来更新。然后，客户端可以下载出现在其他用户位字段中频率最低的片断。

最后阶段

下载接近完成时，最后几块的速度有变慢的趋势。为了加速，客户端向其他所有拥有自己缺少块的用户发送请求。为防止变成无效，客户端在每个块完成后就向其他用户发送一个取消的消息。

没有已定的界限，推荐百分比或能用来作为指导的块计数。

何时进入最后阶段模式有待讨论。一些客户端以请求了所有块来进入最后阶段。其他的等到剩余块的数目少于传输中块的数目或不超过 20 来进入该阶段。保持很少的等待块（1 或 2 块）来将允许值减到最少，如果随机选择块的请求，可能会重复下载到已有的块。更多的协议说明见：http://hal.inria.fr/inria-00000156/en

阻塞与最佳畅通

阻塞有几种原因。TCP 拥塞控制在同时发出许多连接时表现很差。同时，阻塞使每个用户使用 tit-for-tat-ish 算法来确定自己得到一致的下载速度。

下面描述的阻塞算法是现在使用的。所有新算法同时在整个包括它们的网络中工作正常，这是很重要的。

一个好的阻塞算法应有几个标准。它应为更高的 TCP 性能改进并发上传数。它应避免被叫做“原纤化作用”的快速阻塞和未阻塞。它应互换到让自己下载的用户。最后，它应偶尔测试未使用连接来发现是否比当前使用的更好，这叫做最佳畅通。

当前使用的阻塞算法通过每 10 秒钟改变被阻塞用户来避免原纤化作用。

互换和上传数目的改进是由拥有最佳上传速度和感兴趣的 4 个未阻塞用户来控制的。这将使客户端的下载速度变得最大。这 4 个用户被称为下载者，因为它们对从客户端的下载感兴趣。

与下载者相比，具有较高上传速度的用户对未阻塞不感兴趣。如果它们感兴趣，具有最低上传速度的下载者将被阻塞。如果客户端拥有完成的文件，它使用上传速度而不是下载速度来决定哪一个用户未阻塞。

对于最佳畅通，在任何时候只有一个未阻塞用户，而不管它的上传速度（如果感兴趣，它会成为 4 个允许的下载者之一）。最佳畅通的用户 30 秒循环一次。最新连接的用户有 3 次可能作为循环中当前的最佳畅通。这给它们得到一个完成块就上传的机会。

反冷落

（扩展不在官方协议中）

偶尔一个 BitTorrent 用户会被其他用户冷落，它先前从那些用户中下载了数据。在这种情况下，它通常得到很低的下载速度，直到最佳畅通找到更好的用户。为了缓和此问题，当过了 1 分钟而没有从某个用户得到一个片断，BitTorrent 认为它被那个用户“冷落”了，不上传给那个用户（除了最佳畅通以外）。这会频繁导致超过 1 个的用户同时变成最佳畅通（一个例外是最佳畅通，规则如上所述），它会使波动的下载速度回复得更快。

相关文档

- BitTorrent Specification (http://wiki.theory.org/BitTorrentSpecification) - 本文参考英文原文
- BitTorrent WishList (http://wiki.theory.org/BitTorrentWishList) - 开发者愿望表，与最终用户相似。
- BitTorrent Tracker Extensions (http://wiki.theory.org/BitTorrentTrackerExtensions) - 描述了正在使用的服务器协议的不同扩展。

请参阅

- BitTorrent
- P2P