**BT笔记**

## BT种子文件解析

### BEP-3协议

announce:Tracker的主服务器  
  
announce-list(可选)：备用tracker服务器列表  
  
comment(可选):种子文件的注释  
  
comment.utf-8(可选)：种子文件注释的utf-8编码  
  
creation date：种子文件建立的时间，是从1970年1月1日00:00:00到现在的秒数。  
  
encoding:种子文件的默认编码，比如GB2312，Big5，utf-8等  
  
info：所有关于下载的文件的信息都在这个字段里，它包括多个子字段，而且根据下载的是单个文件还是多个文件，子字段的项目会不同。  
  
当种子里包含多个文件时，info字段包括如下子字段：  
  
files：表示文件的名字，大小，该字段包含如下三个子字段：  
  
 lenghth：文件的大小，用byte计算  
  
 path：文件的名字，在下载时不可更改  
  
 path.utf-8：文件名的UTF-8编码，同上  
  
以上的三个字段每个文件都有一组值。  
  
name：推荐的文件夹名，此项可于下载时更改。  
  
name.utf-8:推荐的文件夹名的utf-8编码，同上。  
  
piece length：每个文件块的大小，用Byte计算  
  
pieces：文件的特征信息，该字段比较大，实际上是种子内包含所有的文件段的SHA1的校验值的连接，即将所有文件按照piece length的字节大小分成块，每块计算一个SHA1值，然后将这些值连接起来就形成了pieces字段，由于SHA1的校验值为20Byte，所以该字段的大小始终为20的整数倍字节。该字段是Torrent文件中体积最大的部分，可见如果大文件分块很小，会造成Torrent文件体积庞大。  
  
publisher：文件发布者的名字  
  
publisher.utf-8：文件发布者的名字的utf-8编码  
  
publisher-url：文件发布者的网址  
  
publisher-url.utf-8：文件发布者网址的utf-8编码。  
  
另外，当发布单文件时，files字段是没有的，而  
  
  
lenghth:  
name：  
name.utf-8:  
  
这三个字段负责描述单文件的属性：大小，名字，名字的utf-8编码。其他项目和多文件相同。  
  
以上的项目即为info字段的全部。  
  
说到info就不得不说INFO\_HASH，这个值是info字段的HASH值，20个Byte，同样是使用SHA1作为HASH函数。由于info字段是发布的文件信息构成的，所以INFO\_HASH在BT协议中是用来识别不同的种子文件的。基本上每个种子文件的INFO\_HASH都是不同的(至少现在还没有人发现有SHA的冲突)，所以BT服务器以及客户端都是以这个值来识别不同的种子文件的。  
  
计算的具体范围是从info字段开始(不包含"info"这四个字节)，一直到nodes字段为止(不包含"nodes"这5个字节和nodes前边表示nodes字段长度的"5:"这两个字节)。另外，INFO\_HASH值是即时计算的，并不包含在Torrent文件中。  
  
nodes:最后的一个字段是nodes字段，这个字段包含一系列ip和相应端口的列表，是用于连接DHT初始node。  
  
综上，多文件Torrent的结构的树形图为：

Multi-file Torrent  
├─announce  
├─announce-list  
├─comment  
├─comment.utf-8  
├─creation date  
├─encoding  
├─info  
│ ├─files  
│ │ ├─length  
│ │ ├─path  
│ │ └─path.utf-8  
│ ├─name  
│ ├─name.utf-8  
│ ├─piece length  
│ ├─pieces  
│ ├─publisher  
│ ├─publisher-url  
│ ├─publisher-url.utf-8  
│ └─publisher.utf-8  
└─nodes  
  
单文件Torrent的结构的树形图为：  
  
Single-File Torrent  
├─announce  
├─announce-list  
├─comment  
├─comment.utf-8  
├─creation date  
├─encoding  
├─info  
│ ├─length  
│ ├─name  
│ ├─name.utf-8  
│ ├─piece length  
│ ├─pieces  
│ ├─publisher  
│ ├─publisher-url  
│ ├─publisher-url.utf-8  
│ └─publisher.utf-8  
└─nodes  
  
下面是一个实际的例子来说明Torrent文件的实际结构：  
  
首先要了解在Torrent文件里数据的逻辑表示方法  
  
**字符串**： 字符串由 字符个数:字符串 的形式构成，比如 5:files 就表示"files"  
**整数**： Torrent用10进制来表示整数，由 i数字e 的形式构成，如 i50e 就表示50  
**列表**： 列表由 l 开头，e 结尾，中间是多个字符串，如 l5:files4:infoe 就表示 ["files","info"]  
**字典**： 由 d 开头，e 结尾，中间可以是任何结构， 如 d5:filesl4:info3:eggee 就表示 {files, ["info, "egg""]}  
  
  
示例的文件是一个由包含三个pdf文件的Torrent，在目录 MySampleTorrent 内，目录结构为：  
  
MySampleTorrent  
├─protocol.pdf  
├─Draft\_DHT\_protocol.pdf  
└─fast\_extensions.pdf  
  
通过BitComet制作出的Torrent文件名为 MySampleTorrent.torrent，大小为 976 Byte，分块大小为8M，以下是用WinHex打开该Torrent的显示：  
  
由上图可以得到的信息对应结构树：  
  
├─announce ->http://www.sample.org/announce  
├─announce-list  
├─comment ->This is a sample comment of thsi torrent file.  
├─comment.utf-8 ->This is a sample comment of thsi torrent file.  
├─creation date ->1182679174  
├─encoding ->Shift-JIS  
├─info  
│ ├─files  
│ │ ├─length ->59126  
│ │ ├─path ->Draft\_DHT\_protocol.pdf  
│ │ └─path.utf-8 ->Draft\_DHT\_protocol.pdf  
│ ├─name ->MySampleTorrent  
│ ├─name.utf-8 ->MySampleTorrent  
│ ├─piece length ->8388608  
│ ├─pieces ->751CE8FD85D69D5EDEB31F6B15F3C729587BDDD6  
│ ├─publisher ->Ci Bech  
│ ├─publisher-url ->http://www.cibech.cn  
│ ├─publisher-url.utf-8  
│ └─publisher.utf-8  
└─nodes  
  
由于只有一个Tracker，所以没有 announce-list ，其余不在图上的信息就不列举了。  
这和uTorrent读到的结果是一致的  
  
再来看看校验码的部分，由于设置为8M的文件分块，而总文件大小不到1M，所以三个文件混合成一个块来计算SHA1值。所以先将三个文件用2进制方式合并成一个，再计算SHA1值，如下：  
  
对比这里划白线的SHA1值，是否和上面文件中的一致。  
  
pieces ->751CE8FD85D69D5EDEB31F6B15F3C729587BDDD6  
  
另外看看INFO\_HASH的计算，先将INFO部分用WinHex拷贝出来，单独形成一个文件，命名为 info.part ，需要拷贝的部分如图中所选择部分  
  
红线标识出了开始和结尾，同样，计算该文件的SHA1值，得到结果如下：  
  
将这个校验值和BitComet计算的特征码比较  
  
结果是一致的。BT的服务器同样是根据这部分计算的特征值，确保了无相同的torrent被上传。而BT客户端也是根据这个INFO\_HASH来请求Peer的。

## BT通过.torrent下载的原理

### BT协议之连接TrackerServer

TrackerServer有HTTP和UDP协议两种，这里简单介绍一下HTTP协议的TrackerServer，BTClient通过

HTTP Get请求完成Peers获取和自身的注册。请求的URL格式如下：

Trackerserver-url?info\_hash=xxxxxxxxxxxx,peer\_id=xxxxxxxxxxx,ip=x.x.x.x,port=xxxx,uploaded=xx,downloaded=xx,left=xx,event=x

URL中各参数需要经过urlencode处理，各个参数的意义如下：

|  |  |
| --- | --- |
| 参数名称 | 参数意义 |
| info\_hash | torrent文件中info属性的value部分(bencode格式)的SHA1 哈希值，这个哈希值是BT下载任务的唯一标识。 |
| peer\_id | 任务启动时BT Client为自己随机分配的20字节的ID |
| ip | BTClient的外网IP |
| port | BTClient监听的端口 |
| uploaded | BTClient对该任务已上传的字节数 |
| downloaded | BTClient对该任务已下载的字节数 |
| left | BTClient对该任务尚未下载的字节数 |
| event | 可选字段，表示BTClient对当前下载任务的状态，可选值：started,completed,stoped |

Hash 94b9354069a1347e0401de80411d6a3269cd12a5

分片长度 1048576

主要地http://tracker.trackerfix.com:80/announce

其他地址http://tracker.trackerfix.com:80/announce

其他地址 udp://9.rarbg.me:2710/announce

其他地址 udp://9.rarbg.to:2710/announce

分片数量 62480

Christmas.Inheritance.2017.720p.WEBRip.XviD.AC3-FGT\RARBG.txt.part (3274217704+31 byte(s)).

<http://tracker.trackerfix.com:80/announce>?

info\_hash=%94%B95%40i%A14%7E%04%01%DE%80A%1Dj2i%CD%12%A5&

peer\_id=-TO0042-7d946ca87b91&

port=49155&

uploaded=0&

downloaded=0&

left=3192035246&

compact=1&

no\_peer\_id=0&

event=started&

ip=172.16.49.194

d

8:interval

i60e

5:peers

384:{摜�=瞷摜�=r…

e

peers:[peer://123.147.165.166:15154/?, peer://123.147.165.166:15218/?,]

用户只能够从其他的BT客户端（称为peers)

Interval=60 ????? Interva什么意思s



## DHT

### BEP-5协议

<http://www.bittorrent.org/beps/bep_0005.html>

* **"peer"** 是在一个TCP端口上监听的客户端/服务器，它实现了BitTorrent协议。
* **"节点"**是在一个UDP端口上监听的客户端/服务器，它实现了DHT(分布式哈希表)协议。

DHT由节点组成，它存储了peer的位置。BitTorrent客户端包含一个DHT节点，这个节点用来联系DHT中其他节点，从而得到peer的位置，进而通过BitTorrent协议下载。

当节点要为torrent寻找peer时，它将自己路由表中的节点ID和torrent的infohash进行"距离对比"。然后向路由表中离infohash最近的节点发送请求，问它们正在下载这个torrent的peer的联系信息。如果一个被联系的节点知道下载这个torrent的peer信息，那个peer的联系信息将被回复给当前节点。否则，那个被联系的节点则必须回复在它的路由表中离该torrent的infohash最近的节点的联系信息。最初的节点重复地请求比目标infohash更近的节点，直到不能再找到更近的节点为止。查询完了之后，客户端把自己作为一个peer插入到所有回复节点中离种子最近的那个节点中。

请求 peer 的返回值包含一个不透明的值，称之为"令牌(token)"。如果一个节点宣布它所控制的 peer 正在下载一个种子，它必须在回复请求节点的同时，附加上对方向我们发送的最近的"令牌(token)"。这样当一个节点试图"宣布"正在下载一个种子时，被请求的节点核对令牌和发出请求的节点的 IP 地址。这是为了防止恶意的主机登记其它主机的种子。由于令牌仅仅由请求节点返回给收到令牌的同一个节点，所以没有规定他的具体实现。但是令牌必须在一个规定的时间内被接受，超时后令牌则失效。在 BitTorrent 的实现中，token 是在 IP 地址后面连接一个 secret(通常是一个随机数)，这个 secret 每五分钟改变一次，其中 token 在十分钟以内是可接受的。

* 路由表Routing Table

Every node maintains a routing table of known good nodes. The nodes in the routing table are used as starting points for queries in the DHT. Nodes from the routing table are returned in response to queries from other nodes.

每个node维护一个路由表，路由表保存已知好node，这些路由表里的nodes用作请求DHT的起始点，来自路由表的节点来自其他节点的请求的结果。

好节点是指在过去的15分钟以内，曾经对我们的某一个请求给出过回复的节点；或者曾经对我们的请求给出过一个回复(不用在15分钟以内)，并且在过去的15分钟给我们发送过请求。上述两种情况都可将节点视为好节点。在15 分钟之后，对方没有上述2种情况发生，这个节点将变为可疑的。当节点不能给我们的一系列请求给出回复时，这个节点将变为坏的。相比那些未知状态的节点，已知的好节点会被给于更高的优先级。

路由表覆盖从0到2^160全部的节点ID空间。路由表又被划分为桶(bucket)，每个桶包含一部分的ID空间。空的路由表只有一个桶，它的ID范围从min=0到max=2^160。当ID为N的节点插入到表中时，它将被放到ID范围在min<=N<max的桶中。空的路由表只有一个桶，所以所有的节点都将被放到这个桶中。每个桶最多只能保存K个节点，当前K=8。当一个桶放满了好节点之后，将不再允许新的节点加入，除非我们自身的节点ID在这个桶的范围内。在这样的情况下，这个桶将被分裂为2个新的桶，每个新桶的范围都是原来旧桶的一半。原来旧桶中的节点将被重新分配到这两个新的桶中。如果一个新表只有一个桶，这个包含整个范围的桶将总被分裂为2个新的桶，每个桶的覆盖范围从0..2^159和2^159..2^160。

路由表Routing Table—>桶(bucket)

当桶装满了好节点，新的节点会被丢弃。一旦桶中的某个节点变为了坏的节点，那么我们就用新的节点来替换这个坏的节点。如果桶中有在15分钟内都没有活跃过的节点，我们将这样的节点视为可疑的节点，这时我们向最久没有联系的节点发送ping。如果被ping的节点给出了回复，那么我们向下一个可疑的节点发送ping，不断这样循环下去，直到有某一个节点没有给出ping的回复，或者当前桶中的所有节点都是好的(也就是所有节点都不是可疑节点，他们在过去15分钟内都有活动)。如果桶中的某个节点没有对我们的ping给出回复，我们最好再试一次(再发送一次ping，因为这个节点也许仍然是活跃的，但由于网络拥塞，所以发生了丢包现象，注意DHT的包都是UDP的)，而不是立即丢弃这个节点或者直接用新节点来替代它。这样，我们得路由表将充满稳定的长时间在线的节点。

每个桶都应该维持一个lastchange字段来表明桶中节点的"新鲜"度。当桶中的节点被ping并给出了回复，或者一个节点被加入到了桶，或者一个节点被新的节点所替代，桶的lastchange字段都应当被更新。如果一个桶的lastchange在过去的15分钟内都没有变化，那么我们将更新它。这个更新桶操作是这样完成的：从这个桶所覆盖的范围中随机选择一个ID，并对这个ID执行find\_nodes查找操作。常常收到请求的节点通常不需要常常更新自己的桶，反之，不常常收到请求的节点常常需要周期性的执行更新所有桶的操作，这样才能保证当我们用到DHT的时候，里面有足够多的好的节点。

在插入第一个节点到路由表并启动服务后，这个节点应试着查找DHT中离自己更近的节点，这个查找工作是通过不断的发出find\_node消息给越来越近的节点来完成的，当不能找到更近的节点时，这个扩散工作就结束了。路由表应当被启动工作和客户端软件保存（也就是启动的时候从客户端中读取路由表信息，结束的时候客户端软件记录到文件中）。

* KRPC 协议 KRPC Protocol

KRPC 协议是由bencode编码组成的一个简单的RPC结构，他使用UDP报文发送。一个独立的请求包被发出去然后一个独立的包被回复。这个协议没有重发。它包含3种消息：请求，回复和错误。对DHT协议而言，这里有4种请求：ping，find\_node，get\_peers和announce\_peer。

一条KRPC消息由一个独立的字典组成，其中有2个关键字是所有的消息都包含的，其余的附加关键字取决于消息类型。每条消息都包含**t**关键字，它是一个代表了transaction ID的字符串类型。Transaction ID由请求节点产生，并且回复中要包含回显该字段，所以回复可能对应一个节点的多个请求。transactionID应当被编码为一个短的二进制字符串，比如2个字节，这样就可以对应2^16个请求。另外每个KRPC消息还应该包含的关键字是y，它由一个字节组成，表明这个消息的类型。y对应的值有三种情况：q表示请求，r表示回复，e表示错误。

* 联系信息编码 Contact Encoding

Peers的联系信息被编码为6字节的字符串。又被称为"CompactIP-address/portinfo"，其中前4个字节是网络字节序的IP地址，后2个字节是网络字节序的端口。

节点的联系信息被编码为26字节的字符串。又被称为"Compactnodeinfo"，其中前20字节是网络字节序的节点ID，后面6个字节是peers的"CompactIP-address/portinfo"。

* DHT请求DHT Queries

所有的请求都包含一个关键字id，它包含了请求节点的节点ID。所有的回复也包含关键字id，它包含了回复节点的节点ID。

* ping

最基础的请求就是ping。这时KPRC协议中的"q" = "ping"。Ping请求包含一个参数id，它是一个20字节的字符串包含了发送者网络字节序的节点ID。对应的ping回复也包含一个参数id，包含了回复者的节点ID。

参数:{"id" : "<querying nodes id>"}

回复:{"id" : "<queried nodes id>"}

报文包例子 Example Packets

ping Query = {"t":"aa", "y":"q", "q":"ping", "a":{"id":"abcdefghij0123456789"}}

bencoded=d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:aa1:y1:qe

Response = {"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz123456"}}

bencoded = d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

* find\_node

find\_node 被用来查找给定ID的节点的联系信息。这时KPRC协议中的"q"=="find\_node"。find\_node请求包含2个参数，第一个参数是id，包含了请求节点的ID。第二个参数是target，包含了请求者正在查找的节点的ID。当一个节点接收到了find\_node的请求，他应该给出对应的回复，回复中包含2个关键字id和nodes，nodes是字符串类型，包含了被请求节点的路由表中最接近目标节点的K(8)个最接近的节点的联系信息。

参数: {"id" : "<querying nodes id>", "target" : "<id of target node>"}

回复: {"id" : "<queried nodes id>", "nodes" : "<compact node info>"}

报文包例子 Example Packets

find\_node Query = {"t":"aa", "y":"q", "q":"find\_node", "a": {"id":"abcdefghij0123456789", "target":"mnopqrstuvwxyz123456"}}

bencoded = d1:ad2:id20:abcdefghij01234567896:target20:mnopqrstuvwxyz123456e1:q9:find\_node1:t2:aa1:y1:qe

Response = {"t":"aa", "y":"r", "r": {"id":"0123456789abcdefghij", "nodes": "def456..."}}

bencoded = d1:rd2:id20:0123456789abcdefghij5:nodes9:def456...e1:t2:aa1:y1:re

* get\_peers

get\_peers与torrent文件的infohash有关。这时KPRC协议中的"q"="get\_peers"。get\_peers请求包含2个参数。第一个参数是id，包含了请求节点的ID。第二个参数是info\_hash，它代表torrent文件的infohash。如果被请求的节点有对应info\_hash的peers，他将返回一个关键字values，这是一个列表类型的字符串。每一个字符串包含了"CompactIP-address/portinfo"格式的peers信息。如果被请求的节点没有这个infohash的peers，那么他将返回关键字nodes，这个关键字包含了被请求节点的路由表中离info\_hash最近的K个节点，使用"Compactnodeinfo"格式回复。在这两种情况下，关键字token都将被返回。token关键字在今后的annouce\_peer请求中必须要携带。token是一个短的二进制字符串。

参数:{"id" : "<querying nodes id>", "info\_hash" : "<20-byte infohash of target torrent>"}

回复: {"id" : "<queried nodes id>", "token" :"<opaque write token>", "values" : ["<peer 1 info string>", "<peer 2 info string>"]}

或: {"id" : "<queried nodes id>", "token" :"<opaque write token>", "nodes" : "<compact node info>"}

报文包例子 Example Packets:

get\_peers Query = {"t":"aa", "y":"q", "q":"get\_peers", "a": {"id":"abcdefghij0123456789", "info\_hash":"mnopqrstuvwxyz123456"}}

bencoded = d1:ad2:id20:abcdefghij01234567899:info\_hash20:mnopqrstuvwxyz123456e1:q9:get\_peers1:t2:aa1:y1:qe

Response with peers = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "values": ["axje.u", "idhtnm"]}}

bencoded = d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth6:valuesl6:axje.u6:idhtnmee1:t2:aa1:y1:re

Response with closest nodes = {"t":"aa", "y":"r", "r": {"id":"abcdefghij0123456789", "token":"aoeusnth", "nodes": "def456..."}}

bencoded = d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token8:aoeusnthe1:t2:aa1:y1:re

* announce\_peer

这个请求用来表明发出announce\_peer请求的节点，正在某个端口下载torrent文件。announce\_peer包含4个参数。第一个参数是id，包含了请求节点的ID；第二个参数是info\_hash，包含了torrent文件的infohash；第三个参数是port包含了整型的端口号，表明peer在哪个端口下载；第四个参数数是token，这是在之前的get\_peers请求中收到的回复中包含的。收到announce\_peer请求的节点必须检查这个token与之前我们回复给这个节点get\_peers的token是否相同。如果相同，那么被请求的节点将记录发送announce\_peer节点的IP和请求中包含的port端口号在peer联系信息中对应的infohash下。

参数: {"id" : "<querying nodes id>", "implied\_port": <0 or 1>, "info\_hash" : "<20-byte infohash of target torrent>", "port" : <port number>, "token" : "<opaque token>"}

回复: {"id" : "<queried nodes id>"}

报文包例子 Example Packets:

announce\_peers Query={"t":"aa","y":"q","q":"announce\_peer","a":{"id":"abcdefghij0123456789","implied\_port":1,"info\_hash":"mnopqrstuvwxyz123456", "port": 6881, "token": "aoeusnth"}}

bencoded = d1:ad2:id20:abcdefghij01234567899:info\_hash20:<br /> mnopqrstuvwxyz1234564:porti6881e5:token8:aoeusnthe1:q13:announce\_peer1:t2:aa1:y1:qe

Response = {"t":"aa", "y":"r", "r": {"id":"mnopqrstuvwxyz123456"}}

bencoded = d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

BT种子嗅探器就是根据这个来得到消息的，不过得到消息后我们还需要进一步下载。

## 嗅探器原理

BT种子嗅探器就是利用了DHT协议得到peer信息后会向他之前查询过的节点发送通知这一点，这就是嗅探器的核心。

剩下的工作就是我们要让更多的节点发给我们通知。那么如何让更多的节点发给我们通知呢？

我们要不断的查询自己的好友节点表，并对返回回来的节点进行查询，这样才会有更多的人认识我们

别人向我们查询Target的时候，我们要伪装成Target的好友，返回结果里边包括自己，这样会有更多被查询、收到通知的机会

这就是BT种子嗅探器的原理

## 种子下载器

在BT网络中，通过上述原理收到信息并不是种子，而是发送消息者的ip和port、种子infohash（可以理解为种子的id）。我们如果想要得到种子的话，还需要做一番工作。这里涉及到另外一个非常重要的协议BEP-09，BEP-09规定了如何通过种子infohash得到种子。

大致过程。首先同我们收到的消息里边的ip:port建立TCP连接，然后发送握手消息，并告知对方自己支持BEP-09协议，然后向对方请求种子的信息，收到对方返回的种子信息后，依次或同时请求每一个块。最后所有块收集完后，对其进行拼接并通过sha1算法计算其infohash，如果和我们请求的infohash值相同则保存起来，否则丢掉。