[BitTorrent的DHT协议(译自官方版本)](http://blog.csdn.net/xxxxxx91116/article/details/7970815)

译者前序

DHT协议早在2005年就已经成为了官方BitTorrent协议的一部份，但是我竟然一直没有找到国内的官方翻译稿，所以将其进行翻译，若文中错误，欢迎各位指正。

其次，若想彻底理解DHT协议的原理，建议各位阅读Kademlia协议，在本博客中，有其翻译稿，参见DHT协议基础1，2.

本文英文版官方地址：<http://www.bittorrent.org/beps/bep_0005.html>

DHT协议

BitTorrent使用一种叫做分布式哈希表(distributedsloppy hashtable)的技术，来实现在无tracker的torrent文件中peer的联系信息存储。这个时候，每个peer都是一个tracker。这个协议是基于Kademila协议的，并且在UDP协议基础上实现。

请注意本文档所使用的术语以免引起混淆。“peer”是一个实现了BT协议，且正在监听TCP端口的client/server。“node”是实现了DHT协议的，且正在监听UDP端口的client/server。DHT由nodes组成并保存peer的位置信息。BitTorrent客户端也包括DHTnode，这个DHTnode主要是用来联系DHT中的其他nodes，以得peer的位置信息，从而通过BitTorrent协议下载。

**概述**

每一个node都有一个全局的唯一标识“nodeID”。NodeIDS的产生是随机的，且使用与BitTorrent的infohashes相同的160-bit空间。“distancemetric”用来比较2个nodeIDs或者nodeID与infohash的接近程度。Nodes必须维护一个路由表，其中保存了一部分其他nodes的联系信息。越接近自身节点时，路由表的信息会更加详细。nodes保存了很多接近自己的节点，但是离自己很远的节点的联系信息确知道得很少。

在Kademlia中，“distancemetric”采用XOR异或计算，并转换为一个无符号整数。distance(A,B)= |A xor B| ，并且距离越小表示2个节点越接近。

当一个node想得到某个torrent文件的peers，它首先使用distancemetric来比较torrent文件的info\_hash和路由表中节点的nodeID。接下来向路由表中nodeID与info\_hash最接近的那些节点发送请求，得到当前正在下载这个torrent文件数据的peers的联系信息。如果被请求的节点知道这个torrent文件的peers，那么peer的联系信息将包含在回复中。否则，被请求的节点必须返回他的路由表中更接近info\_hash得那些节点。原始的请求node不断向新获得的那些node中，更接近目标info\_hash的那些node发送请求，直到不能获得更近的nodes。当查找结束时，client将自己的信息作为一个peer插入到在刚才请求中给出回复的那些节点中，nodeid与info\_hash最接近的哪个节点上，这样，哪个节点又多保存了一个peer信息。

在请求peers的时候，对方给我们的回复必须还包含一个不透明的令牌，我们称他为“token”。这样当我们宣布我们正在下载某个torrent，想让对方保存我们的信息时，我们必须使用对方向我们发送的最近的一个token。这样当我们宣布我们在下载一个torrent时，被请求的node检查这个token和IP是否与之前他们向我们回复的一样。这样是为了防止恶意的攻击。由于token仅仅由请求的节点返回，所以我们不规定他的具体实现。但是token必须有一个可接受的时间范围，超过这个时间，token将失效。在BitTorrent的实现中，token是在IP地址后面连接一个secret(可以视为一个随机数)，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**

每一个node维护一个路由表保存已知的好节点。这些路由表中的nodes被作为DHT请求的起始节点。路由表中的nodes是在不断的向其他node请求过程中，对方节点回复的。

并不是我们在请求过程中收到得节点都是平等的，有的node是好的，而有的node是死掉的。很多使用DHT协议的nodes都可以发送请求并接收回复，但是不能主动回复其他节点的请求(我认为这是由于防火墙或者NAT的原因)。对每一个node的路由表，只包含好的nodes是很重要的。好的node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node视为好的node。在15分钟之后，对方没有上述2种情况发生，这个node将变为可疑的。当nodes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。相比未知状态的nodes，我们将给好的节点更高的优先权。

路由表覆盖从0到2160完整的nodeID空间。路由表又被划分为buckets(桶)，每一个bucket包含一个子部分的nodeID空间。一个空的路由表只有一个bucket，它的ID范围从min=0到max=2160。当一个nodeID为“N”的node插入到表中时，它将被放到ID范围在min&lt;= N &lt;max的bucket中。一个空的路由表只有一个bucket所以所有的node都将被放到这个bucket中。每一个bucket最多只能保存K个nodes，当前K=8。当一个bucket放满了好的nodes之后，将不再允许新的节点加入，除非我们自身的nodeID在这个bucket的范围内。在这样的情况下，这个bucket将被分裂为2个新的buckets，每一个新桶的范围都是原来旧桶的一半。原来旧桶中的nodes将被重新分配到这两个新的buckets中。如果是一个只有一个bucket的新表，这个包含整个范围的bucket将总被分裂为2个新的buckets，第一个的覆盖范围从0..2159，第二个的范围从2159..2160。

当bucket装满了好的nodes，那么新的node将被丢弃。一旦bucket中的某一个node变为了坏的node，那么我们就用新的node来替换这个坏的node。如果bucket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pinged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node没有给出ping的回复，或者当前bucket中的所有nodes都是好的(也就是所有nodes都不是可疑nodes，他们在过去15分钟内都有活动)。如果bucket中的某个node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node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node或者直接用新node来替代它。这样，我们得路由表将充满稳定的长时间在线的nodes。

每一个bucket都应该维持一个“lastchange”字段来表明bucket中的nodes有多新鲜。当一个bucket中的node被ping并给出了回复，或者一个node被加入到了bucket，或者一个node被一个新的node所替代，bucket的“lastchanged”字段都应当被更新。如果一个bucket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bucket操作是这样完成的：从这个bucket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nodes通常不需要常常更新自己的buckets，反之，不常常收到请求的nodes常常需要周期性的执行更新所有buckets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nodes。

在第一个node插入路由表并开始服务后，这个node应该试着查找离自身更近的node，这个查找工作是通过不断的发布find\_node消息给越来越近的nodes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）。

**BitTorret协议扩展**

BitTorrent协议已经被扩展为可以在通过tracker得到的peer之间互相交换nodeUDP端口号(也就是告诉对方我们的DHT服务端口号)，在这样的方式下，客户端可以通过下载普通的种子文件来自动扩展DHT路由表。新安装的客户端第一次试着下载一个无tracker的种子时，它的路由表中将没有任何nodes，这是它需要在torrent文件中找到联系信息。

peers如果支持DHT协议就将BitTorrent协议握手消息的保留位的第八字节的最后一位置为1。这时如果peer收到一个handshake表明对方支持DHT协议，就应该发送PORT消息。它由字节0x09开始，payload的长度是2个字节，包含了这个peer的DHT服务使用的网络字节序的UDP端口号。当peer收到这样的消息是应当向对方的IP和消息中指定的端口号的node发送ping。如果收到了ping的回复，那么应当使用上述的方法将新node的联系信息加入到路由表中。

**Torrent文件扩展**

一个无tracker的torrent文件字典不包含announce关键字，而使用一个nodes关键字来替代。这个关键字对应的内容应该设置为torrent创建者的路由表中K个最接近的nodes。可供选择的，这个关键字也可以设置为一个已知的可用节点，比如这个torrent文件的创建者。请不要自动加入router.bittorrent.com到torrent文件中或者自动加入这个node到客户端路由表中。

nodes= [["<host>", <port>], ["<host>",<port>], ...]

nodes= [["127.0.0.1", 6881], ["your.router.node",4804]]

**KRPC协议**

KRPC协议是由B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。

一个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。每一个消息都包含t关键字，它是一个代表了transactionID的字符串类型。transactionID由请求node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。另一个每个KRPC消息都包含的关键字是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

Peers的联系信息被编码为6字节的字符串。又被称为"CompactIP-address/port info"，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。

Nodes的联系信息被编码为26字节的字符串。又被称为"Compactnode info"，其中前20字节是网络字节序的nodeID，后面6个字节是peers的"CompactIP-address/port info"。

请求

请求，对应于KPRC消息字典中的“y”关键字的值是“q”，它包含2个附加的关键字“q”和“a”。关键字“q”是一个字符串类型，包含了请求的方法名字。关键字“a”一个字典类型包含了请求所附加的参数。

回复

回复，对应于KPRC消息字典中的“y”关键字的值是“r”，包含了一个附加的关键字r。关键字“r”是一个字典类型，包含了返回的值。发送回复消息是在正确解析了请求消息的基础上完成的。

错误

错误，对应于KPRC消息字典中的y关键字的值是“e”，包含一个附加的关键字e。关键字“e”是一个列表类型。第一个元素是一个数字类型，表明了错误码。第二个元素是一个字符串类型，表明了错误信息。当一个请求不能解析或出错时，错误包将被发送。下表描述了可能出现的错误码：

|  |  |
| --- | --- |
| 错误码 | 错误描述 |
| 201 | 一般错误 |
| 202 | 服务错误 |
| 203 | 协议错误,比如不规范的包，无效的参数，或者错误的token |
| 204 | 未知方法 |

错误包例子:

一般错误={"t":"aa", "y":"e", "e":[201,"A Generic Error Ocurred"]}

B编码=d1:eli201e23:AGenericErrorOcurrede1:t2:aa1:y1:ee

**DHT请求**

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

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nodeID。对应的ping回复也包含一个参数id，包含了回复者的nodeID。

参数: {"id"&nbsp;: "<querying nodes id>"}

回复:{"id"&nbsp;: "<queried nodes id>"}

报文包例子

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

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

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

B编码=d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

find\_node

Findnode被用来查找给定ID的node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node的nodeID。第二个参数是target，包含了请求者正在查找的node的nodeID。当一个node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是一个字符串类型，包含了被请求节点的路由表中最接近目标node的K(8)个最接近的nodes的联系信息。

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

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

报文包例子

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

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

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

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

get\_peers

Getpeers与torrent文件的info\_hash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node的nodeID。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nodes，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"&nbsp;: "<querying nodes id>","info\_hash"&nbsp;: "<20-byte infohash of targettorrent>"}

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

or:{"id"&nbsp;: "<queried nodes id>","token"&nbsp;:"<opaque write token>","nodes"&nbsp;: "<compact node info>"}

报文包例子

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

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

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

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

回复最接近的nodes= {"t":"aa", "y":"r", "r":{"id":"abcdefghij0123456789", "token":"aoeusnth","nodes": "def456..."}}

B编码=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node，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node的nodeID；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node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>", "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

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

B编码=d1:ad2:id20:abcdefghij01234567899:info\_hash20:<br />

mnopqrstuvwxyz1234564:porti6881e5:token8:aoeusnthe1:q13:announce\_peer1:t2:aa1:y1:qe

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

B编码=d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

译者注：

在实践观察中发现两种新的消息

1.vote:表示客户端发送速度过快，希望可以减慢

2.v:表示客户端的版本号