第17讲 | P2P协议：我下小电影，99%急死你

笔记本:	P.趣谈网络协议		
创建时间:	2018/6/26 9:11	**更新时间:**	2018/6/26 9:11
作者:	hongfenghuoju		
URL:			

第17讲 | P2P协议：我下小电影，99%急死你

2018-06-25 刘超

如果你想下载一个电影，一般会通过什么方式呢?

当然，最简单的方式就是通过HTTP进行下载。但是相信你有过这样的体验，通过浏览器下载的时候，只要文件稍微大点，下载的速度就奇慢无比。

还有种下载文件的方式，就是通过FTP，也即文件传输协议。FTP 采用两个 TCP 连接来传输一个文件。

- 控制连接：服务器以被动的方式，打开众所周知用于 FTP 的端口 21，客户端则主动发起连接。该连接将命令从客户端传给服务器，并传回服务器的应答。常用的命令有：list——获取文件目录；reter——取一个文件；store——存一个文件。
- 数据连接：每当一个文件在客户端与服务器之间传输时，就创建一个数据连接。

FTP 的两种工作模式

每传输一个文件，都要建立一个全新的数据连接。FTP 有两种工作模式，分别是主动模式（PORT）和被动模式（PASV），这些都是站在 FTP 服务器的角度来说的。

主动模式下，客户端随机打开一个大于 1024 的端口 N，向服务器的命令端口 21 发起连接，同时开放 N+1 端口监听，并向服务器发出 “port N+1” 命令，由服务器从自己的数据端口 20，主动连接到客

户端指定的数据端口 N+1。

被动模式下，当开启一个 FTP 连接时，客户端打开两个任意的本地端口 N（大于 1024）和 N+1。第一个端口连接服务器的 21 端口，提交 PASV 命令。然后，服务器会开启一个任意的端口 P（大于 1024），返回“227 entering passive mode”消息，里面有 FTP 服务器开放的用来进行数据传输的端口。客户端收到消息取得端口号之后，会通过 N+1 号端口连接服务器的端口 P，然后在两个端口之间进行数据传输。

P2P 是什么?

但是无论是 HTTP 的方式，还是 FTP 的方式，都有一个比较大的缺点，就是难以解决单一服务器的带宽压力， 因为它们使用的都是传统的客户端服务器的方式。

后来，一种创新的、称为 P2P 的方式流行起来。P2P就是peer-to-peer。资源开始并不集中地存储在某些设备上，而是分散地存储在多台设备上。这些设备我们姑且称为 peer。

想要下载一个文件的时候，你只要得到那些已经存在了文件的 peer，并和这些 peer 之间，建立点对点的连接，而不需要到中心服务器上，就可以就近下载文件。一旦下载了文件，你也就成为 peer 中的一员，你旁边的那些机器，也可能会选择从你这里下载文件，所以当你使用 P2P 软件的时候，例如 BitTorrent，往往能够看到，既有下载流量，也有上传的流量，也即你自己也加入了这个 P2P 的网络，自己从别人那里下载，同时也提供给其他人下载。可以想象，这种方式，参与的人越多，下载速度越快，一切完美。

种子（.torrent）文件

但是有一个问题，当你想下载一个文件的时候，怎么知道哪些 peer 有这个文件呢?

这就用到种子啦，也即咱们比较熟悉的.torrent 文件。.torrent 文件由两部分组成，分别是：announce（tracker URL）和文件信息。

文件信息里面有这些内容。

- info 区：这里指定的是该种子有几个文件、文件有多长、目录结构，以及目录和文件的名字。
- Name 字段：指定顶层目录名字。
- 每个段的大小：BitTorrent（简称 BT）协议把一个文件分成很多个小段，然后分段下载。
- 段哈希值：将整个种子中，每个段的 SHA-1 哈希值拼在一起。

下载时，BT 客户端首先解析.torrent 文件，得到 tracker 地址，然后连接 tracker 服务器。tracker 服务器回应下载者的请求，将其他下载者（包括发布者）的 IP 提供给下载者。下载者再连接其他下载者，根据.torrent 文件，两者分别对方告知自己已经有的块，然后交换对方没有的数据。此时不需要其他服务器参与，并分散了单个线路上的数据流量，因此减轻了服务器的负担。

下载者每得到一个块，需要算出下载块的 Hash 验证码，并与.torrent 文件中的对比。如果一样，则说明块正确，不一样则需要重新下载这个块。这种规定是为了解决下载内容的准确性问题。

从这个过程也可以看出，这种方式特别依赖 tracker。tracker 需要收集下载者信息的服务器，并将此信息提供给其他下载者，使下载者们相互连接起来，传输数据。虽然下载的过程是非中心化的，但是加入这个 P2P 网络的时候，都需要借助 tracker 中心服务器，这个服务器是用来登记有哪些用户在请求哪些资源。

所以，这种工作方式有一个弊端，一旦 tracker 服务器出现故障或者线路遭到屏蔽，BT 工具就无法正常工作了。

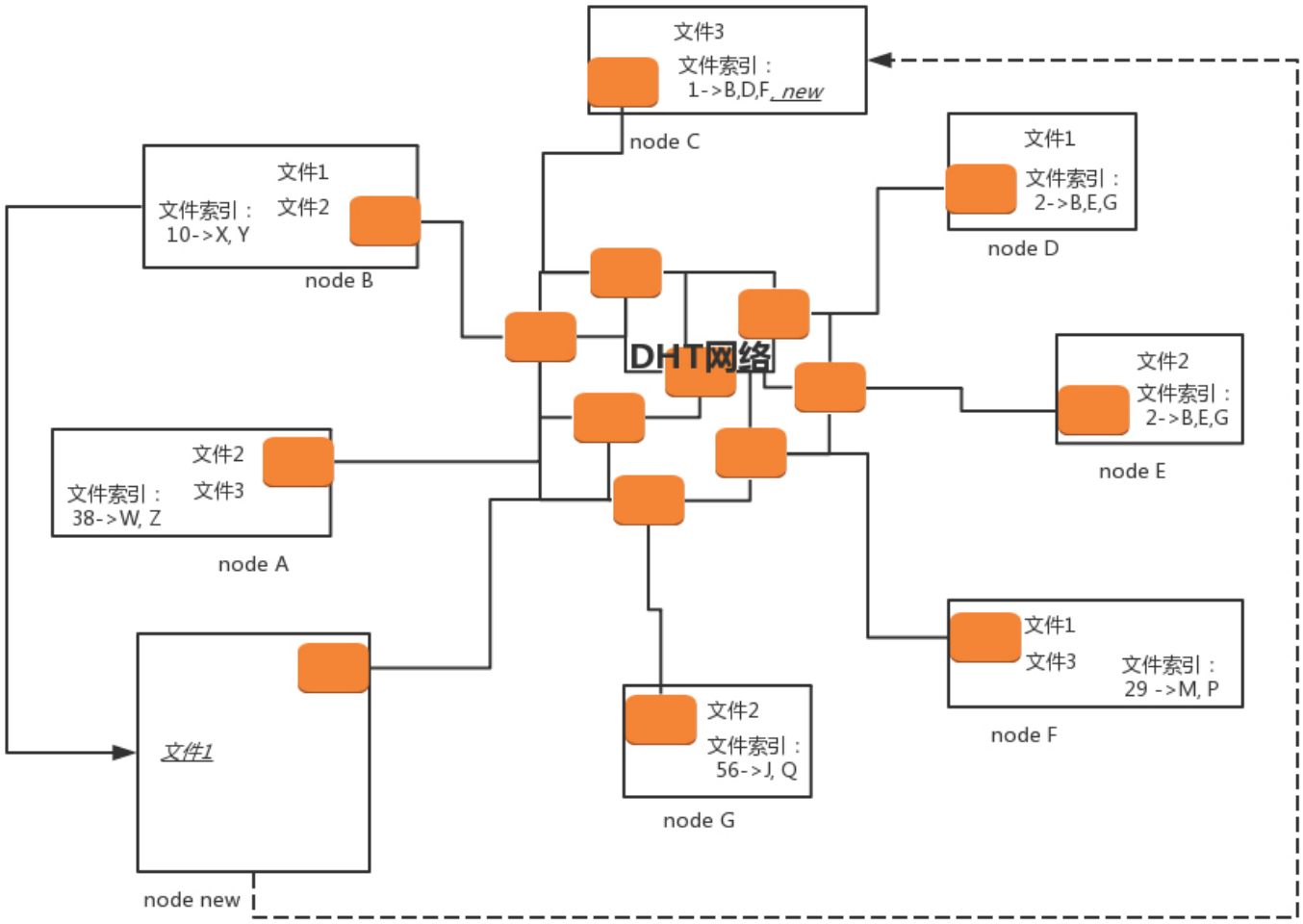
去中心化网络（DHT）

那能不能彻底非中心化呢？

于是，后来就有了一种叫作DHT（Distributed Hash Table）的去中心化网络。每个加入这个 DHT 网络的人，都要负责存储这个网络里的资源信息和其他成员的联系信息，相当于所有人一起构成了一个庞大的分布式存储数据库。

有一种著名的 DHT 协议，叫Kademlia 协议。这个和区块链的概念一样，很抽象，我来详细讲一下这个协议。

任何一个 BitTorrent 启动之后，它都有两个角色。一个是peer，监听一个 TCP 端口，用来上传和下载文件，这个角色表明，我这里有某个文件。另一个角色DHT node，监听一个 UDP 的端口，通过这个角色，这个节点加入了一个 DHT 的网络。



在 DHT 网络里面，每一个 DHT node 都有一个 ID。这个 ID 是一个很长的串。每个 DHT node 都有责任掌握一些知识，也就是文件索引，也即它应该知道某些文件是保存在哪些节点上。它只需要有这些知识就可以了，而它自己本身不一定就是保存这个文件的节点。

哈希值

当然，每个 DHT node 不会有全局的知识，也即不知道所有的文件保存在哪里，它只需要知道一部分。那应该知道哪一部分呢？这就需要用哈希算法计算出来。

每个文件可以计算出一个哈希值，而DHT node 的 ID 是和哈希值相同长度的串。

DHT 算法是这样规定的：如果一个文件计算出一个哈希值，则和这个哈希值一样的那个 DHT node，就有责任知道从哪里下载这个文件，即便它自己没保存这个文件。

当然不一定这么巧，总能找到和哈希值一模一样的，有可能一模一样的 DHT node 也下线了，所以 DHT 算法还规定：除了一模一样的那个 DHT node 应该知道，ID 和这个哈希值非常接近的 N 个 DHT node 也应该知道。

什么叫和哈希值接近呢？例如只修改了最后一位，就很接近；修改了倒数 2 位，也不远；修改了倒数 3 位，也可以接受。总之，凑齐了规定的 N 这个数就行。

刚才那个图里，文件 1 通过哈希运算，得到匹配 ID 的 DHT node 为 node C，当然还会有其他的，我这里没有画出来。所以，node C 有责任知道文件 1 的存放地址，虽然 node C 本身没有存放文件 1。

同理，文件 2 通过哈希运算，得到匹配 ID 的 DHT node 为 node E，但是 node D 和 E 的 ID 值很近，所以 node D 也知道。当然，文件 2 本身没有必要一定在 node D 和 E 里，但是碰巧这里就在 E 那有一份。

接下来一个新的节点 node new 上线了。如果想下载文件 1，它首先要加入 DHT 网络，如何加入呢?

在这种模式下，种子.torrent 文件里面就不再是 tracker 的地址了，而是一个 list 的 node 的地址，而所有这些 node 都是已经在 DHT 网络里面的。当然随着时间的推移，很可能有退出的，有下线的，但是我们假设，不会所有的都联系不上，总有一个能联系上。

node new 只要在种子里面找到一个 DHT node，就加入了网络。

node new 会计算文件 1 的哈希值，并根据这个哈希值了解到，和这个哈希值匹配，或者很接近的 node 上知道如何下载这个文件，例如计算出来的哈希值就是 node C。

但是 node new 不知道怎么联系上 node C，因为种子里面的 node 列表里面很可能没有 node C，但是它可以问，DHT 网络特别像一个社交网络，node new 只有去它能联系上的 node 问，你们知道不知道 node C 的联系方式呀?

在 DHT 网络中，每个 node 都保存了一定的联系方式，但是肯定没有 node 的所有联系方式。DHT 网络中，节点之间通过互相通信，也会交流联系方式，也会删除联系方式。和人们的方式一样，你有你的朋友圈，你的朋友有它的朋友圈，你们互相加微信，就互相认识了，过一段时间不联系，就删除朋友关系。

有个理论是，社交网络中，任何两个人直接的距离不超过六度，也即你想联系比尔盖茨，也就六个人就能够联系到了。

所以，node new 想联系 node C，就去万能的朋友圈去问，并且求转发，朋友再问朋友，很快就能找到。如果找不到 C，也能找到和 C 的 ID 很像的节点，它们也知道如何下载文件 1。

在 node C 上，告诉 node new，下载文件 1，要去 B、D、 F，于是 node new 选择和 node B 进行 peer 连接，开始下载，它一旦开始下载，自己本地也有文件 1 了，于是 node new 告诉 node C 以及和 node C 的 ID 很像的那些节点，我也有文件 1 了，可以加入那个文件拥有者列表了。

但是你会发现 node new 上没有文件索引，但是根据哈希算法，一定会有某些文件的哈希值是和 node new 的 ID 匹配上的。在 DHT 网络中，会有节点告诉它，你既然加入了咱们这个网络，你也有责任知道某些文件的下载地址。

好了，一切都分布式了。

这里面遗留几个细节的问题。

- DHT node ID 以及文件哈希是个什么东西?

节点 ID 是一个随机选择的 160bits（20 字节）空间，文件的哈希也使用这样的 160bits 空间。

- 所谓 ID 相似，具体到什么程度算相似?

在 Kademlia 网络中，距离是通过异或（XOR）计算的。我们就不以 160bits 举例了。我们以 5 位来举例。

01010 与 01000 的距离，就是两个 ID 之间的异或值，为 00010，也即为 2。 01010 与 00010 的距离为 01000，也即为 8,。01010 与 00011 的距离为 01001，也即 8+1=9 。以此类推，高位不同的，表示距离更远一些；低位不同的，表示距离更近一些，总的距离为所有的不同的位的距离之和。

这个距离不能比喻为地理位置，因为在 Kademlia 网络中，位置近不算近，ID 近才算近，所以我把这个距离比喻为社交距离，也即在朋友圈中的距离，或者社交网络中的距离。这个和你住的位置没有关系，和人的经历关系比较大。

还是以 5 位 ID 来举例，就像在领英中，排第一位的表示最近一份工作在哪里，第二位的表示上一份工作在哪里，然后第三位的是上上份工作，第四位的是研究生在哪里读，第五位的表示大学在哪里读。

如果你是一个猎头，在上面找候选人，当然最近的那份工作是最重要的。而对于工作经历越丰富的候选人，大学在哪里读的反而越不重要。

DHT 网络中的朋友圈是怎么维护的?

就像人一样，虽然我们常联系人的只有少数，但是朋友圈里肯定是远近都有。DHT 网络的朋友圈也是一样，远近都有，并且按距离分层。

假设某个节点的 ID 为 01010，如果一个节点的 ID，前面所有位数都与它相同，只有最后 1 位不同。这样的节点只有 1 个，为 01011。与基础节点的异或值为 00001，即距离为 1；对于 01010 而言，这样的节点归为“k-bucket 1”。

如果一个节点的 ID，前面所有位数都相同，从倒数第 2 位开始不同，这样的节点只有 2 个，即 01000 和 01001，与基础节点的异或值为 00010 和 00011，即距离范围为 2 和 3；对于 01010 而言，这样的节点归为“k-bucket 2”。

如果一个节点的 ID，前面所有位数相同，从倒数第 i 位开始不同，这样的节点只有 2^(i-1) 个，与基础节点的距离范围为 [2^(i-1), 2^i)；对于 01010 而言，这样的节点归为“k-bucket i”。

最终到从倒数 160 位就开始都不同。

你会发现，差距越大，陌生人越多，但是朋友圈不能都放下，所以每一层都只放 K 个，这是参数可以配置。

DHT 网络是如何查找朋友的?

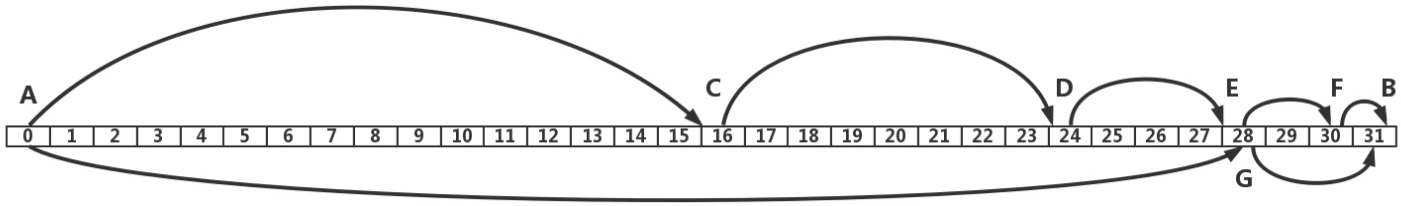
假设，node A 的 ID 为 00110，要找 node B ID 为 10000，异或距离为 10110，距离范围在 [2^4, 2^5)，所以这个目标节点可能在“k-bucket 5”中，这就说明 B 的 ID 与 A 的 ID 从第 5 位开始不同，所以 B 可能在“k-bucket 5”中。

然后，A 看看自己的 k-bucket 5 有没有 B。如果有，太好了，找到你了；如果没有，在 k-bucket 5 里随便找一个 C。因为是二进制，C、B 都和 A 的第 5 位不同，那么 C 的 ID 第 5 位肯定与 B 相同，即它与 B 的距离会小于 2^4，相当于比 A、B 之间的距离缩短了一半以上。

再请求 C，在它自己的通讯录里，按同样的查找方式找一下 B。如果 C 知道 B，就告诉 A；如果 C 也不知道 B，那 C 按同样的搜索方法，可以在自己的通讯录里找到一个离 B 更近的 D 朋友（D、B 之间距离小于 2^3），把 D 推荐给 A，A 请求 D 进行下一步查找。

Kademlia 的这种查询机制，是通过折半查找的方式来收缩范围，对于总的节点数目为 N，最多只需要查询 log2(N) 次，就能够找到。

例如，图中这个最差的情况。



A 和 B 每一位都不一样，所以相差 31，A 找到的朋友 C，不巧正好在中间。和 A 的距离是 16，和 B 距离为 15，于是 C 去自己朋友圈找的时候，不巧找到 D，正好又在中间，距离 C 为 8，距离 B 为 7。于是 D 去自己朋友圈找的时候，不巧找到 E，正好又在中间，距离 D 为 4，距离 B 为 3，E 在朋友圈找到 F，距离 E 为 2，距离 B 为 1，最终在 F 的朋友圈距离 1 的地方找到 B。当然这是最最不巧的情况，每次找到的朋友都不远不近，正好在中间。

如果碰巧了，在 A 的朋友圈里面有 G，距离 B 只有 3，然后在 G 的朋友圈里面一下子就找到了 B，两次就找到了。

在 DHT 网络中，朋友之间怎么沟通呢?

Kademlia 算法中，每个节点只有 4 个指令。

- PING：测试一个节点是否在线，还活着没，相当于打个电话，看还能打通不。
- STORE：要求一个节点存储一份数据，既然加入了组织，有义务保存一份数据。
- FIND_NODE：根据节点 ID 查找一个节点，就是给一个 160 位的 ID，通过上面朋友圈的方式找到那个节点。
- FIND_VALUE：根据 KEY 查找一个数据，实则上跟 FIND_NODE 非常类似。KEY 就是文件对应的 160 位的 ID，就是要找到保存了文件的节点。

DHT 网络中，朋友圈如何更新呢?

- 每个 bucket 里的节点，都按最后一次接触的时间倒序排列，这就相当于，朋友圈里面最近联系过的人往往是最熟的。

- 每次执行四个指令中的任意一个都会触发更新。
- 当一个节点与自己接触时，检查它是否已经在 k-bucket 中，也就是说是否已经在朋友圈。如果在，那么将它挪到 k-bucket 列表的最底，也就是最新的位置，刚联系过，就置顶一下，方便以后多联系；如果不在，新的联系人要不要加到通讯录里面呢？假设通讯录已满的情况，PING 一下列表最上面，也即最旧的一个节点。如果 PING 通了，将旧节点挪到列表最底，并丢弃新节点，老朋友还是留一下；如果 PING 不通，删除旧节点，并将新节点加入列表，这人联系不上了，删了吧。

这个机制保证了任意节点加入和离开都不影响整体网络。

小结

好了，今天的讲解就到这里了，我们总结一下：

- 下载一个文件可以使用 HTTP 或 FTP，这两种都是集中下载的方式，而 P2P 则换了一种思路，采取非中心化下载的方式；
- P2P 也是有两种，一种是依赖于 tracker 的，也即元数据集中，文件数据分散；另一种是基于分布式的哈希算法，元数据和文件数据全部分散。

接下来，给你留两个思考题：

1. 除了这种去中心化分布式哈希的算法，你还能想到其他的应用场景吗？
2. 在前面所有的章节中，要下载一个文件，都需要使用域名。但是网络通信是使用 IP 的，那你知道怎么实现两者的映射机制吗？

我们的专栏马上更新过半了，不知你掌握得如何？每节课后我留的思考题，你都有没有认真思考，并在留言区写下答案呢？我会从已发布的文章中选出一批认真留言的同学，赠送学习奖励礼券和我整理的独家网络协议知识图谱。