BitTorrent Protocol – BTP/1.0

摘要

本文当描述了 BitTorrent 协议（BTP/1.0）。BTP 是一种通过互联网的分享文件的协议，始于 2002 年。虽然它包含高度中央化的内容但还是可以看作一种点对点（P2P）协议。虽然一份正式的，详尽的，完整的协议描述一直欠缺，BTP 本身却已经相当成熟，在不同的平台都有相应的实现。

BTP 由 Bram Cohen 设计和实现，FTP 实现在某些情况下导致服务器和宽带资源不堪重负，在这种情况下 BTP 作为一种 P2P 协议用以取代 FTP。通常情况下，客户端在下载一个文件的时候不会占用上行带宽。BTP 利用该情况让客户端互相传输一些数据。对比 FTP，这是 BTP 有了巨大的可伸缩性和成本管理的优势。

目录

1. 介绍

“MUST”，“MUST NOT”，“REQUIRED”，“SHALL”，“SHALL NOT”，“SHOULD”，“SHOULD NOT”，“RECOMMANDED”，“MAY”和“OPTIONAL”这些词在本文中的具体意思在 RFC 2119 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.)[3]中描述。

文件传输协议（File Transfer Protocol (FTP) (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.)[1]）加上最新的安全扩展，依然是通过互联网传输文件的安全可靠的标准方式。然而，FTP 是高度中心化的客户端-服务器模式，这意味着它不适合发布同时会有大量客户端请求的文件。为了应对这种情况，很多组织采用了限制同一个文件的并发请求数量的方式，或者就将请求分配到其他镜像服务器。毫无疑问，两种方法都有缺点。因此急需一种可以解决这些问题的方法。

方法就是 BTP，让客户端互相上传下载数据而不是镜像站点来分担数据流量。因为一般客户端在下载文件时不会占用上行带宽，这种方法不会对客户端造成什么麻烦。这种方法还有一个额外的优势，一些没有足够资源的小组织可以不需要太多设备投入就可以在互联网上发布大型文件。

1.1 扩展

一些个人和社区提出了很多 BTP 的修改版和扩展。BitTorrent 社区认为让它们成为 BTP 的一部分的最好做法就是让本文档包括它们。可是很多扩展都没有可靠的实现来支撑，或者被认为是不够成熟的。

1.2 面向的读者

本文档主要针对希望在特定平台上实现 BTP 的开发者。当然，系统管理员或者构架师，可以利用本文档来更加深入地理解如何安装一个 BTP。比如，在一个有很多敏感数据的机器上安装 BTP，就可以学习更多关于安全的细节。安全细节在第七节（安全考虑）具体描述。

1.3 术语

节点（Peer）：

　　节点是在网络中参与文件传输的一个点，既可以是客户端也可以是服务器，还可以同时是。

相邻节点（Neighboring peers）：

　　相互之间有活动 TCP 连接的节点。

客户端（Client）：

　　客户端是用户的节点代理。

种子（Torrent）：

　　种子就是客户端在下载的单个文件（单文件种子）或者多个文件（多文件种子）。

种子群（Swarm）：

　　在同一个种子上活动的节点组成的网络。

做种人（Seeder）：

　　具有一份完整种子拷贝的节点。

种子服务器（Tracker）：

种子服务器是一个中心服务器，保存了种子的信息和相关的种子群。它作为节点进入种子群的网关。

元数据文件（Metainfo file）：

一个文本文件，保存了中心信息。比如，种子服务器的URL。一般用.torrent扩展名。

节点ID（Peer ID）：

一个20字节的字符串用来标识节点。节点ID是如何获取的超出了本文的范畴。但节点必须保证其节点ID在种子群中是唯一的。

种子特征码（Info hash）：

一个SHA1的哈希值唯一地标识了种子。从元数据文件计算而来。

1.4 概述

BTP逻辑上由两种协议组成，分别是THP（Tracker HTTP Protocol）和PWP（Peer Wire Protocol）。THP用来与种子服务器通信，加入种子群，报告进度等。PWP是节点之间通信的机制，也是真正执行上传下载任务的。

客户端为了下载种子，必须遵循一下步骤：

1. 读取元数据文件（.torrent）。
2. 允许客户端与其他节点通信的指令不许周期性地通过THP向种子服务器获取。
3. 使用PWP与种子群中的其他节点连接并交换数据，完成种子下载。

发布一个种子不许遵循以下步骤：

1. 种子服务器必须运行。
2. 生成包含种子服服务器地址和种子信息的元数据文件，并且对外发布。
3. 至少有一个做种人。

2.BT编码

BT使用一种平台独立的方式编码数据。在BTP/1.0中，元数据文件和所有种子服务器的响应都采用了BT编码方式。BT编码定义了两种基本类型（整数和字符串）和两种复合类型（列表和字典）。

使用增强的BNF语法（Augmented BNF syntax (Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” November 1997.)[5]）来表示BT编码的形式如下：

```
dictionary = "d" 1*(string anytype) "e" ; non-empty dictionary
list       = "l" 1*anytype "e"          ; non-empty list
integer    = "i" signumber "e"
string     = number ":" <number long sequence of any CHAR>
anytype    = dictionary / list / integer / string
signumber  = "-" number / number
number     = 1*DIGIT
CHAR       = %x00-FF                    ; any 8-bit character
DIGIT      = "0" / "1" / "2" / "3" / "4" /
             "5" / "6" / "7" / "8" / "9"
```

2.1 基本类型

整数被编码成一个以字母i开头和字母e结尾的字符串，中间是整数的字符串形式。例如，整数123编

码成 i123e。

字符串被编码成在前面加上字符串长度以及冒号。例如，字符串”announce”编码成“8:announce”。

2.2 复合类型

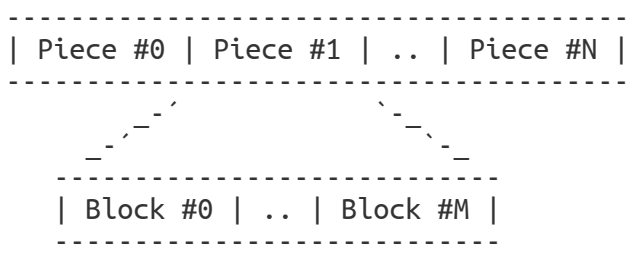
复合类型是其他 BT 编码类型的结构化表示。

列表是任意数量的 BT 编码类型以字母 l 开头和 e 结尾。列表可以嵌入列表和字典。比如，列表“li2e3:fooe”是一个包含数字 2 和字符串“fooe”的列表。

字典是任意数量的键/值对，以字母 d 开头和字母 e 结尾。所有的键是 BT 编码的字符串，而相关的值可以是任何 BT 类型。比如，字典“d5:monthi4e4:name5:aprile”是一个包含“month”/4 和“name”/”aprile”的字典。所有的字典键必须分类（MUST be sorted）。

3. 分片和块

本节描述一个种子是如何被分片和分块的。种子被分为一个或多个分片。每片表示了一个范围内的数据，可以用分片特征码（SHA1）来验证完整性。当通过 PWP 传输数据时，分片又被分为一个或多个块，如下图所示：



3.1 分片

元数据文件中指明了种子的分片数量。每片大小是固定的，可以用下面的公式计算：

```
fixed_piece_size = size_of_torrent / number_of_pieces
```

/ 表示除法运算符。只有最后一个分片可以比固定的大小小。

分片大小由种子的发布者决定。一般使元数据文件不要超过 70KB 的分片大小比较好。

为了计算一个分片在一个文件或多个文件中的位置，种子被看作是一个单一的，连续的字节流。如果种子包含多个文件，那种子就被看作这些文件以出现在元数据文件中的顺序的串联。从概念上讲，种子只在最后所有分片都下载完成，而且检验没问题后，才被转化为一个或者多个文件。但在实际应用中，BT 实现可能会为了操作系统和文件系统的需要，采用一个更好的方式。

3.2 分块

块大小是由具体 BT 实现决定的，与分片大小无关。一旦大小固定了，每分片中的块数量可以用以下公式计算：

```
number_of_blocks = (fixed_piece_size / fixed_block_size)
                 + !!(fixed_piece_size % fixed_block_size)
```

% 是模运算符，！是否定运算符。否定运算在这里是为了保证最后一个因子要么是 0 要么是 1。给出块

在片中的起始位置，块的编号可以用以下公式计算：

```
block_index = block_offset % fixed_block_size
```

4. 元数据文件

元数据文件为客户端提供了种子服务器和种子的信息。不仅列出了需要下载的文件，还告诉客户端如何分片如何检验分片。

为了让客户端可以辨别元数据文件，元数据文件应该（SHOULD）以扩展名.torrent 结尾，并且关联媒体类型“application/x-bittorrent”。客户端如何读取元数据文件超出了本文档的范畴，一种最用户友好的方式是让客户端从网页上下载，然后打开，开始下载。在这种方式下，明显的，BTP 的复杂性对用户来说是透明的，因为只涉及到 FTP 或者 HTTP。

4.1 元数据文件的结构

元数据文件包含了一个 BT 编码的字典。以下的键是必须的（REQUIRED），除非有特别说明。

‘announce‘：

一个字符串值，包含了种子服务器的 URL。

‘announce-list’：

这是一个任意的（OPTIONAL）字符串列表。每个字符串值都是一个备用种子服务器的 URL。在 BTP/1.0 中没有使用。

‘comment’：

这是一个任意的（OPTIONAL）字符串值，表示种子作者的评注。

‘created by’：

这是一个任意的字符串值，表示创建此元数据文件的客户端名字与版本。

‘creation date’：

这是一个任意的字符串值。是以 Unix 记时格式表示的种子创建时间。

‘info’：

这个键指向了一个目录，保存了下载文件的信息。下面章节具体说明。

4.1.1 单文件种子

如果种子只有一个文件，那么字典‘info’必须包含以下键：

‘length’：

这是一个整数值，表示文件的字节大小。

‘md5sum’：

这是一个任意的值。如果有，必须是一个 32 字符长度的字符串，表示文件的 MD5 值。在 BTP/1.0 中没有使用。

‘name’：

这是一个字符串值，表示文件名。

‘piece length’：

这是一个整数值，表示分片的字节长度。

‘pieces’：

这是一个字符串值，是所有分片的 20 字节特征码（SHA1）的串联。比如，该字符串的前 20 个字符是编号为 0 的分片的特征码。

完整的文件是所有分片的合并。

4.1.2 多文件种子

如果种子有多个文件，那么字典‘info’必须包含以下键，

‘file’：这是一个字典的列表。每个种子中的文件都有一个字典结构相关联：

‘length’：

一个整数值，一个文件的字节长度。

‘md5sum’：

这是一个任意的字符串值。没有文件的 md5 检验和。此值在 BTP/1.0 中不使用。

‘path’：

这是一个字符串列表，是相对于顶级目录的文件路径。列表中最后一个字符串是文件名，而前面的字符串则是文件的路径。

‘name’：

这是一个字符串值，表示顶级目录名。

‘piece length’：

这是一个整数值。表示分片的字节大小。

‘pieces’：

这是一个字符串值，是所有分片的 20 字节特征码（SHA1）的串联。比如，该字符串的前 20 个字符是编号为 0 的分片的特征码。

5. THP（The Tracker HTTP Protocol）

THP 是让节点互相了解的一种机制。一个种子服务器是一个 HTTP 服务器，与节点连接并让节点加入种子群。种子服务器作为 BTP/1.0 中唯一的中心化元素。种子服务器本身并不提供任何下载数据。种子服务器依靠节点发送请求。如果节点没有请求，种子服务器会认为节点已‘死’。

5.1 请求

客户端要与种子服务器取得链接，必须（MUST）向元数据文件中‘announce’指名的 URL 发送一个标准的 HTTP GET 请求。GET 请求必须按照 HTTP 协议添加参数。以下参数是必须的：

‘info_hash’:

这是一个必须的（REQUIRED）20 字节长度的特征码（SHA1）值。为了取得这个值，节点必须计算元数据中‘info’字段的特征码（SHA1）。

‘peer_id’：

这是一个必须的（REQUIRED）字符串值。20 字节长度节点自定义。

‘port’：

节点用来接受来自其他节点的数据的端口号。BTP/1.0 没有指定特定的端口或者特定范围的端口号。这个值是必须的。

‘uploaded’：

这是一个 10 进制值。表示自从节点向种子服务器发送‘started’后在种子群中上传的数据字节大小。这个字段是必须的（REQUIRED）。

‘downloaded’：

这是一个 10 进制值。表示自从节点向种子服务器发送‘started’后在种子群中下载的数据字节大小。这个字段是必须的（REQUIRED）。

‘left’：

这是一个 10 进制值。表示节点要完成该种子的下载还需要的数据字节大小。这个字段是必须的。

‘ip’：

这是一个可选的（OPTIONAL）值，如果存在，则表示节点的 IPv4 或者 IPv6 地址，也可以是 DNS 域名。

‘event’：

这个参数是可选的（OPTIONAL）。如果没有指定，那么这个请求被认为是一个周期性的请求。否则，必须是以下三个值之一：

‘started’：

第一个发送给种子服务器的 GET 请求，必须（MUST）包含此值。

‘stopped’：

当节点正常退出时，这个值应该（SHOULD）被发送给种子服务器。

‘completed’：

当节点下载了完整的种子后，应该（SHOULD）向种子服务器发送此值。不应该（SHOULD NOT）发送，在有完整种子的情况启动。

5.2 响应

在收到 HTTP 的 GET 请求后，种子服务器必须回应一个以“text/plain”MIME 类型标注的文档。该文档必须包含一个 BT 编码的字典，且有以下字段：

‘failure reason’：

这个值是可选的（OPTIOINAL）。如果存在，这个字典必须不（MUST NOT）包含其他任何字段。节点将认为加入种子群失败，而该字段值则是一个可读的失败原因。

‘interval’：

节点必须向种子服务器发送通常的 HTTP GET 请求来获取更新的节点列表以及种子服务器的状态。该字段值表示了节点向种子服务器发送该请求的时间周期。该字段是必须的（REQUIRED）。

‘complete’：

这是一个表示做种者数量的整数值。可选（OPTIONAL）。

‘incomplete’：

这是一个表示下载者数量的整数值。可选（OPTIONAL）。

‘peers’：

这是一个 BT 编码的字典列表，包含了下载文件时应该连接的节点。该字段是必须的（REQUIRED）。每个字典有如下字段：

‘peer id’：

这是一个必须的（REQUIRED）字符串表示节点自定义的节点 ID。

'ip'：

这是一个必须的（REQUIRED）字符串值，表示了节点的网络地址。可以是点分 IPv4 地址、十六进制 IPV6 地址或者是 DNS 域名。

‘port’：

必须的（REQUIRED）整数值，节点开发数据传输的端口号。

6. PWP（The Peer Wire Protocol）

PWP 的目的是实现相邻节点之间进行通信，并传输文件内容。在节点读取元数据文件并与种子服务器通信后，获取了其他节点的信息，然后使用 PWP 与其他节点通信。PWP 是建立在 TCP 上的，并用异步消息处理所有通信。

6.1 节点选择算法原则

PWP 没有提供一种标准的算法来选择与哪个相邻节点传输分片。不过以下建议可以看作是类似算法的指南。

算法应该有助于减少上传流量与下载流量的比值。最坏情况下，上传和下载流量一样。

算法不应该用一种“以牙还牙”的方式来处理那种刚加入种子群的节点，因为它们没有任何分片可以上传。

算法应该充分利用下载和上传带宽，通过限制上传或者接受数据的连接的数量。通过减少活动链接的数量，TCP 拥塞可以被避免。

算法应该有序地处理数据请求，使活动的连接达到饱和。

算法应该可以与使用不同算法的其他节点很好地工作。

6.2 握手

本地节点，打开一个断开来接听其他节点的连接。这个端口通过 THP 告诉种子服务器。因为 BTP/1.0 没有指定任何端口，BTP 实现负责选择一个端口。

想要与本地节点通信的其他节点必须打开一个连接到此端口的 TCP 连接，并且完成握手操作。握手必须在（MUST）发送其他任何数据之间完成。在通过以下规则完成握手过程之前，本地节点不能（MUST NOT）向其他节点发送任何数据。如果握手违反了这些规则，本地节点必须（MUST）关闭与其他节点的链接。

一次握手是一个有以下结构的字节组。

```
-----------------------------------------------------------------
| Name Length | Protocol Name | Reserved | Info Hash | Peer ID |
-----------------------------------------------------------------
```

Name Length：

一个无符号的值，第一个字节表示了包含协议名称的字符串的长度。在 BTP/1.0 中，该值是 19。本地节点知道它自己的协议，当然也包括协议的字符串长度。如果该长度与这第一个字节的值不一致，那么该链接必须（MUST）被丢弃。

Protocol Name：

这是一个字符串，必须（MUST）用 ASCII 码形式表示协议的名字，而且长度要和 Name Length 指定的一致。协议名称用来判断节点所使用的 BTP 版本。在 BTP/1.0 中，该名字是 “BitTorrent protocol”。如果该字符串与本地节点所使用的协议名称不一致，那么该链接必须（MUST）被丢弃。

Reserved：

接下来的 8 个字节是为了以后的扩展而保留的，不需要做任何解释。

Info Hash：

下面的 20 字节是一个特征码（SHA1），在元数据文件中的。如果两个节点读取的是相同的元数据文件，那么该值就相同。如果不相同，那么链接必须（MUST）被丢弃。这种情况可能是因为客户端因为某些原因不再上传该文件。因为这个特征码的存在，客户端可以在一个端口上同时监听不同种子的请求。

Peer ID：

最后的 20 字节是节点自定义的节点 ID。本地节点通过该值来确定连接。因此，如果该值与本地节点本身的 ID 相同，则链接必须（MUST）被丢弃。同样的，如果已经存在该节点 ID 的连接，该链接也必须（MUST）被丢弃。

在 BTP/1.0 中，握手结构总共占 68 字节。

6.3 消息通信

完成 PWP 握手的 TCP 链接两端都有可能向对方发送消息。PWP 消息有两种作用，一就是更新相邻节点的状态，二是传送数据块。

PWP 消息可以分为两类：

面向状态的消息：

此类消息用了告知相邻节点节点状态的改变。在节点状态改变时必须（MUST）发送这种消息，不管对方是什么状态。面向状态的消息可以分为这几类：Interested,Unintereted,Choked,UnChoked,Have 和 bitfield。

面向数据的消息：

此类消息处理消息的请求和发送。可分为三类：Request,Cancel,Piece。

6.3.1 节点状态

对于链接的两端，节点必须维护以下两种状态标志：

Choked：

当设置时，意味着堵塞的节点不允许请求数据。

Interested：

当设置时，意味着节点希望从其他节点那获取数据。这表明节点将开始请求数据，如果不是堵塞的。

一个堵塞的节点不能（MUST not）发送任何面向数据的消息。但是可以发送任何其他消息给其他节点。

一个未堵塞的节点可以发送面向数据的消息给其他节点。要如何堵塞，堵塞多少节点和开发多少节点全由具体实现决定。这是故意允许节点采用不同的启发式方法来进行节点选择。

一个 interested 的节点其实就是告知其他节点它希望尽早地得到面向数据的消息一旦该节点被开发。必须注意的是，节点不能一厢情愿假设其他节点需要它的数据，并对其 interested。也许有一些原因让节点对除了有数据的节点感兴趣。

6.3.2 节点消息

所有在 PWP 消息中的整数成员都是是 4 字节大端序的。还有就是所有编号和偏移量都是 0 开始的。

一个 PWP 消息有如下的结构：

```
----------------------------------------
| Message Length | Message ID | Payload |
----------------------------------------
```

Message Length：

一个表示消息长度的整数，不包括该字段本身。如果一条消息没有实际负载内容，那么该大小为 1.大小为 0 的消息也可能（MAY）作为保活消息周期性地发送。除了这个限制，消息上都要加上 4 字节，BTP 没有指定一个最大长度。因此 BTP 实现可以（MAY）选择一个不同的限制，比如可以选择与使用太大消息长度的节点断开链接。

Message ID：

这是一个单字节值，表示了消息的类型。BTP/1.0 定义了 9 种消息类型。后面会提到。

Payload：

这是一个可变长度的字节流。

如果收到一条消息不遵循该结构，那么该链接应该（SHOULD）被直接丢弃。比如，接受方必须保证消息 ID 是一个合法的值，而负载是期望中的。

为了兼容以后可能的协议扩展，客户端应该（SHOULD）忽略未知的消息。当然也有一些情况下客户端在收到未知消息后会选择关闭链接，为了安全考虑或者认为保存大型的未知消息是一种资源的浪费。

BTP/1.0 定义了以下几种消息：

6.3.3 Choke

该消息 ID 是 0 并且没有负载。节点发送该消息告诉对方，对方已被堵塞。

6.3.4 Unchoke

该消息 ID 是 1 并且没有负载。节点发送该消息告诉对方，对方已被解除堵塞。

6.3.5 Interested

该消息 ID 是 2 并且没有负载。节点发送该消息告诉对方，希望获得对方的数据。

6.3.6 Uninterested

该消息 ID 是 3 并且没有负载。节点发送该消息告诉对方，已经不对对方的任何数据分片感兴趣。

6.3.7 Have

该消息 ID 是 4 并且有一个长度是 4 的负载。负载表示的是节点已经拥有的数据分片的编号。收到该消息的节点必须验证编号，如果编号不在指定范围内则关闭链接。同样的，收到该消息的节点必须（MUST）发送一个 interested 消息给发送者如果它确实也需要该分片。或者发送一个对该分片的请求。

6.3.8 Bitfield

该消息 ID 是 5 并且有一个变长的负载。负载表示了发送者成功下载的数据分片，第一个字节的高位表示了编号为 0 的分片。如果一个位是 0 说明发送者没有该分片。节点必须（MUST）在完成握手后立刻发送该消息，不过如果一个分片都没有可以（MAY）选择不发送。在节点之间通信过程的其他时间该消息不能（MUST not）发送。

6.3.9 Request

该消息 ID 是 6 有一个长度是 12 的负载。负载是 3 个整数值，表示了发送者希望获取的分片中的块。接受者只能（MUST）发送 piece 消息给发送者作为回应。不过要遵循上述的 choke 和 interested 机制。负载有如下的结构：

```
--------------------------------------------
| Piece Index | Block Offset | Block Length |
--------------------------------------------
```

6.3.10 Piece

该消息 ID 是 7 并且有一个变长的负载。负载前两个两个整数指名哪个分片中哪个块，第三个字段表示该块的数据。注意，数据长度可以通过消息总长度减去 9 来获得。负载有如下结构：

```
-----------------------------------------
| Piece Index | Block Offset | Block Data |
-----------------------------------------
```

6.3.11 Cancel

该消息 ID 是 8 并且有一个长度为 12 的负载。负载是三个整数值，表示了发送者曾请求过但现在不需要了的块编号和所在分片编号。接受者收到该消息后必须（MUST）去除请求标志。负载有如下结构：

```
--------------------------------------------
| Piece Index | Block Offset | Block Length |
--------------------------------------------
```

6.4 The End Game

为了结束下载任务，节点可能会加速向相邻节点发送请求消息。客户端一旦获取了所需的数据，必须向所有未回应的相邻节点发送取消消息。这被看作是结束任务。

客户端通常阶段性地发送块请求消息；收到之前请求的响应后再发送新的请求。当所有块都获得后，客户端进入结束任务阶段。

6.5 分片选择策略

BTP/1.0 没有规定要用什么特定的顺序下载分片。然后，经验表明，按照“最少优先”下载等待时间是最少的。为了找到最少的分片，客户端必须从所有相邻节点中计算分片编号二进制位为 1 时的分片。和最小的分片就是最少分片，应该选择它请求。

6.6 节点选择策略

本节描述了选择相邻节点交换数据的堵塞算法。BTP 实现可以选择其他任何策略只要遵循第 6.1 节描述的指导原则。

完成初始化握手后，链接双方都设置了 Choked 为 true 和 Interested 为 false。

所有链接都会根据它们的下载率周期性地进行评估。评估分数受各种因素影响，比如节点在特定时间内维护开放链接的意愿，节点上传率和其他实现定义的标准。

节点根据安装上述方法得出评估分数排序。假设只有五个节点可以同时下载。节点选择算法会开放尽可能多高评分的节点。如果一个高评分的节点后来变为 interested。那么节点选择算法将堵塞最差劲的开放节点。

上述算法唯一缺乏的是让新节点有一个公平的下载机会的能力。尽管新节点的评估很差。有一个简单的方法是，让一个节点是周期性随机选择的无论它评估如何。因为这是一种不断轮循的方式，它使新节点也有机会开放下载。

7. 安全考虑

本节讨论 BTP/1.0 的安全问题。但不包括提到问题的具体解决方法，只是给降低风险提了一些建议。

7.1 THP 问题

使用 HTTP 来进行客户端和种子服务器之间的通信使 BTP/1.0 变得易受攻击。在 RFC 2616 (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)[6]的安全考虑一节有提及。

7.2 种子服务器的拒绝服务攻击

种子服务器本来就是要服务多个客户端。然后种子服务器一旦遭到拒绝服务攻击，那么相关的种子群就无法正常工作。这种形式的攻击难以防范。不过元数据文件中可以定义多个备用的种子服务器，因此可以躲过此类攻击。

7.3 节点认证问题

当客户端与种子服务器连接时没有可靠的认证机制。主要的办法就是让种子服务器检查节点 ID 和客户端的 IP 地址。缺乏认证机制可能会导致这种问题，当一个客户端退出时，在同一主机上的另一个客户端运行却共用同一个 IP 地址。另外，恶意的节点可能会使用多个节点 IP 来伪装自己。

7.4 DNS 欺骗

使用 BTP/1.0 的客户端依赖于 DNS 系统，来连接种子服务器或者其他节点。因此客户端有可能遭到 DNS 欺骗这种攻击手段。客户端需要对 IP 地址与域名之间的映射留心。

具体来说，BTP/1.0 客户端应该（SHOULD）依赖域名解析服务器来完成 IP 地址和域名之间的映射，而不是依靠之前的缓存。如果客户端为了提供效率而采用缓存的映射，那么一定要留意 DNS 服务器的 TTL 报告。

如果客户端没有遵循这些规则，可能会被之前的 IP 地址欺骗。因为根据 RFC 1900 (Carpenter, B. and Y. Rekhter, "Renumbering Needs Work," February 1996.)[2]网络地址的重新分配变得越来越平常了。这种攻击的风险正在加大。考虑这种情况可以减少潜在的危险。

7.5 文件和目录名问题

元数据文件提供了为下载文件以及路径命名的建议。这种功能非常像在 RFC 2183 (Troost, R., Dorner, S., and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field," August 1997.)[4]中说明了的附加头，而且在该 RFC 文档中描述的安全问题在 BT 中也同样适用。简单地说，BTP 客户端应该（SHOULD）检查元数据文件中建议的文件名，以免对本地系统造成不必要的麻烦。此外，多文件种子尤其需要注意，因为文件路径是相对于顶级目录的，要避免类似..这样向上级目录引用的方式，可能会造成文件保存在了顶级目录外面。

以 Unix 作为例子，可能的危害有：

- 创建启动文件（如".login"）
- 创建或者覆盖系统文件（如"/etc/password"）
- 覆盖任何已存在的文件
- 将可执行文件放置在任何命令搜索路径（如"~/bin/more"）
- 发送文件到管道（如"|sh"）

要清楚地认识到，上述没有包含所有的问题。这只是一小部分例子而已。实现者必须警惕任何在目标系统上潜在的危险。通常来说，BTP 客户端不应该（SHOUD NOT）命名或者保存文件，使文件可能在用户不知情的情况下被解释或者执行。

7.6 验证节点之间数据交换的完整性

默认情况下，所有从其他节点收到的数据都被认为是不可靠的，客户端应该（SHOULD）验证数据的完整性再确定接受数据。元数据文件包含检验每一分片的特征码（SHA1）和每个文件的 MD5。SHA1 比 MD5 更加可靠，推荐使用。

信任最后下载的文件就意味着信任元数据文件提供的信息。确定元数据文件的可靠性超出了本文档的范畴。

7.7 敏感信息的传输

一些客户端在生成节点 ID 时包含了一些它们自身的信息。客户端实现者应该明白，这些信息可能会被用来确定客户端是否存在可被利用的安全漏洞。

8. 引用

[1] Postel, J. and J. Reynolds, “File Transfer Protocol,” STD 9, RFC 959, October 1985.
[2] Carpenter, B. and Y. Rekhter, “Renumbering Needs Work,” RFC 1900, February 1996.
[3] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[4] Troost, R., Dorner, S., and K. Moore, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field,” RFC 2183, August 1997 (TXT, HTML, XML).
[5] Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” RFC 2234, November 1997 (TXT, HTML, XML).
[6] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).