J. Fonseca
B. Reza
L. Fjeldsted
DIKU

BitTorrent Protocol -- BTP/1.0

Abstract

This document describes the BitTorrent Protocol version 1.0 referred to as "BTP/1.0". The BitTorrent Protocol (BTP) is a protocol for collaborative file distribution across the Internet and has been in place on the Internet since 2002. It is best classified as a peer-to-peer (P2P) protocol, although it also contains highly centralized elements. BTP has already been implemented many times for different platforms, and could well be said to be a mature protocol, although a formal, detailed and complete description has so far been lacking.

BTP was devised and implemented by Bram Cohen as a P2P replacement to standard File Transfer Protocol (FTP) to be used in places where the usage of an FTP implementation poses too much strain on the server in terms of request-processing and sheer bandwidth. Normally a client does not use her upload capacity while downloading a file. The BTP approach capitalizes on this fact by having clients upload bits of the data to each other. In comparison to FTP this adds huge scalability and cost-management advantages.

Table of Contents

TOC

1. Introduction

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119**[3].

The **File Transfer Protocol (FTP)**[1] with the recent additions of security extensions, still remains the standard for secure and reliable transmission of large files over the Internet. However, its highly centralized client-server approach also means, it is inadequate for mass publication of files, where a single point may expect to be requested by a critically large number of clients simultaneously. To remedy this situation many organizations either implement a cap on the number of simultaneous requests, or spread the load on multiple mirror servers. Needless to say both approaches have their drawbacks, and a solution that addresses these problems is highly needed.

The approach in BitTorrent Protocol (BTP) is to spread the load not on mirror servers, but to the clients themselves by having them upload bits of the file to each other while downloading it. Since the clients usually do not utilize their upload capacity while fetching a file, this approach does not put the clients in any disadvantage. This has the added advantage that even small organizations with limited resources can publish large files on the Internet without having to invest in costly infrastructure.

1.1 Extensions

Since the introduction of BTP many modifications and extensions have been proposed by individuals and community forums. To the extent that these extensions have become part of what the BitTorrent community considers best practice they have been included in this document. However, many extensions have been omitted either because they have been deemed to lack interoperability with existing implementations, or because they are not regarded as being sufficiently mature.

1.2 Audience

This document is aimed at developers who wish to implement BTP for a particular platform. Also, system administrators and architects may use this document to fully

understand the implications of installing an implementation of BTP. In particular, it is advised to study the security implications in more detail, before installing an implementation on a machine that also contains sensitive data. Security implications are discussed in **Section 7**.

1.3 Terminology

Peer:
: A peer is a node in a network participating in file sharing. It can simultaneously act both as a server and a client to other nodes on the network.

Neighboring peers:
: Peers to which a client has an active point to point TCP connection.

Client:
: A client is a user agent (UA) that acts as a peer on behalf of a user.

Torrent:
: A torrent is the term for the file (single-file torrent) or group of files (multi-file torrent) the client is downloading.

Swarm:
: A network of peers that actively operate on a given torrent.

Seeder:
: A peer that has a complete copy of a torrent.

Tracker:
: A tracker is a centralized server that holds information about one or more torrents and associated swarms. It functions as a gateway for peers into a swarm.

Metainfo file:
: A text file that holds information about the torrent, e.g. the URL of the tracker. It usually has the extension .torrent.

Peer ID:
: A 20-byte string that identifies the peer. How the peer ID is obtained is outside the scope of this document, but a peer must make sure that the peer ID it uses has a very high probability of being unique in the swarm.

Info hash:
: A SHA1 hash that uniquely identifies the torrent. It is calculated from data in the metainfo file.

1.4 Overall Operation

BTP consists of two logically distinct protocols, namely the Tracker HTTP Protocol (THP), and the Peer Wire Protocol (PWP). THP defines a method for contacting a tracker for the purposes of joining a swarm, reporting progress etc. PWP defines a mechanism for communication between peers, and is thus responsible for carrying out the actual download and upload of the torrent.

In order for a client to download a torrent the following steps must be carried through:

1. A metainfo file must be retrieved.
2. Instructions that will allow the client to contact other peers must be periodically requested from the tracker using THP.
3. The torrent must be downloaded by connecting to peers in the swarm and trading pieces using PWP.

To publish a torrent the following steps must be taken:

1. A tracker must be set up.
2. A metainfo file pointing to the tracker and containing information on the structure of the torrent must be produced and published.
3. At least one seeder with access to the complete torrent must be set up.

TOC

2. Bencoding

Bencoding encodes data in a platform independent way. In BTP/1.0 the metainfo file and all responses from the tracker are encoded in the bencoding format. The format specifies two scalar types (integers and strings) and two compound types (lists and dictionaries).

The **Augmented BNF syntax**[5] for bencoding format is:

```
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 Scalar Types

Integers are encoded by prefixing a string containing the base ten representation of the integer with the letter "i" and postfixing it with the letter "e". E.g. the integer 123 is encoded as i123e.

Strings are encoded by prefixing the string content with the length of the string followed by a colon. E.g. the string "announce" is encoded as "8:announce".

2.2 Compound Types

The compound types provides a mean to structure elements of any bencoding type.

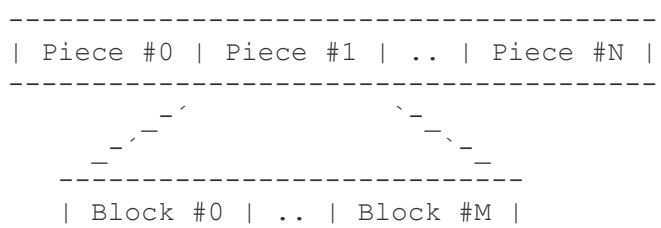
Lists are an arbitrary number of bencoded elements prefixed with the letter "l" and postfixed with the letter "e". It follows that lists can contain nested lists and dictionaries. For instance "li2e3:fooe" defines a list containing the integer "2" and the string "foo".

Dictionaries are an arbitrary number of key/value pairs delimited by the letter "d" at the beginning and the letter "e" at the end. All keys are bencoded strings while the associated value can be any bencoded element. E.g. "d5:monthi4e4:name5:aprile" defines a dictionary holding the associations: "month" => "4" and "name" => "april". All dictionary keys MUST be sorted.

TOC

3. Pieces and Blocks

This section describes how a torrent is organized in pieces and blocks. The torrent is divided into one or more pieces. Each piece represents a range of data which it is possible to verify using a piece SHA1 hash. When distributing data over PWP pieces are divided into one or more blocks, as shown in the following diagram:



3.1 Pieces

The number of pieces in the torrent is indicated in the metainfo file. The size of each piece in the torrent remains fixed and can be calculated using the following formula:

```
fixed_piece_size = size_of_torrent / number_of_pieces
```

where "/" is the integer division operator. Only the last piece of the torrent is allowed to have fewer bytes than the fixed piece size.

The size of a piece is determined by the publisher of the torrent. A good recommendation is to use a piece size so that the metainfo file does not exceed 70 kilobytes.

For the sake of calculating the correct position of a piece within a file, or files, the torrent is regarded as a single continuous byte stream. In case the torrent consists of multiple files, it is to be viewed as the concatenation of these files in the order of their appearance in the metainfo file. Conceptually, the torrent is only translated into files when all its pieces have been downloaded and verified using their respective SHA1 values; although in practice an implementation may choose a better approach in accordance with local operating system and filesystem specific demands.

3.2 Blocks

The size of a block is an implementation defined value that is not dependant on the fixed piece size. Once a fixed size is defined, the number of blocks per piece can be calculated using the formula:

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

where "%" denotes the modulus operator, and "!" the negation operator. The negation operator is used to ensure that the last factor only adds a value of 0 or 1 to the sum. Given the start offset of the block its index within a piece can be calculated using the formula:

```
block_index = block_offset % fixed_block_size
```

TOC

4. The Metainfo File

The metainfo file provides the client with information on the tracker location as well as the torrent to be downloaded. Besides listing which files will result from downloading the torrent, it also lists how the client should split up and verify individual pieces making up the complete torrent.

In order for a client to recognize the metainfo file it SHOULD have the extension .torrent and the associated the media type "application/x-bittorrent". How the client retrieves the metainfo file is beyond the scope of this document, however, the most user-friendly approach is for a client to find the file on a web page, click on it, and start the download immediately. This way, the apparent complexity of BTP as opposed to FTP or HTTP transfer is transparent to the user.

4.1 The Structure of the Metainfo File

The metainfo file contains a bencoded dictionary with the following structure. A key below is REQUIRED unless otherwise noted.

'announce':

This is a string value. It contains the announce URL of the tracker.

'announce-list':
This is an OPTIONAL list of string values. Each value is a URL pointing to a backup tracker. This value is not used in BTP/1.0.

'comment':
This is an OPTIONAL string value that may contain any comment by the author of the torrent.

'created by':
This is an optional string value and may contain the name and version of the program used to create the metainfo file.

'creation date':
This is an OPTIONAL string value. It contains the creation time of the torrent in standard Unix epoch format.

'info':
This key points to a dictionary that contains information about the files to download. The entries are explained in the following sections.

4.1.1 Single File Torrents

If the torrent only specifies one file, the info dictionary must have the following keys:

'length':
This is an integer value indicating the length of the file in bytes.

'md5sum':
This is an OPTIONAL value. If included it must be a string of 32 characters corresponding to the MD5 sum of the file. This value is not used in BTP/1.0.

'name':
A string containing the name of the file.

'piece length':
An integer indicating the number of bytes in each piece.

'pieces':
This is a string value containing the concatenation of the 20-byte SHA1 hash value for all pieces in the torrent. For example, the first 20 bytes of the string represent the SHA1 value used to verify piece index 0.

The complete file is derived by combining all the pieces into one string of bytes.

4.1.2 Multi File Torrents

If the torrent specifies multiple files, the info dictionary must have the following structure:

'files':
This is a list of dictionaries. Each file in the torrent has a dictionary associated to it having the following structure:

'length':
This is an integer indicating the total length of the file in bytes.

'md5sum':
This is an OPTIONAL value. if included it must be a string of 32 characters corresponding to the MD5 sum of the file. This value is not used in BTP/1.0.

'path':
This is a list of string elements that specify the path of the file, relative to the topmost directory. The last element in the list is the name of the file, and the elements preceding it indicate the directory hierarchy in which this file is situated.

'name':

This is a string value. It contains the name of the top-most directory in the file structure.

'piece length':
This is an integer value. It contains the number of bytes in each piece.

'pieces':
This is a string value. It must contain the concatenation of all 20-byte SHA1 hash values that are used by BTP/1.0 to verify each downloaded piece. The first 20 bytes of the string represent the SHA1 value used to verify piece index 0.

TOC

5. The Tracker HTTP Protocol

The Tracker HTTP Protocol (THP) is a simple mechanism for introducing peers to each other. A tracker is a HTTP service that must be contacted by a peer in order to join a swarm. As such the tracker constitutes the only centralized element in BTP/1.0. A tracker does not by itself provide access to any downloadable data. A tracker relies on peers sending regular requests. It may assume that a peer is dead if it misses a request.

5.1 Request

To contact the tracker a peer MUST send a standard HTTP GET request using the URL in the "announce" entry of the metainfo file. The GET request must be parametrized as specified in the HTTP protocol. The following parameters must be present in the request:

'info_hash':
This is a REQUIRED 20-byte SHA1 hash value. In order to obtain this value the peer must calculate the SHA1 of the value of the "info" key in the metainfo file.

'peer_id':
This is a REQUIRED string and must contain the 20-byte self-designated ID of the peer.

'port':
The port number that the peer is listening to for incoming connections from other peers. BTP/1.0 does not specify a standard port number, nor a port range to be used. This key is REQUIRED.

'uploaded':
This is a base ten integer value. It denotes the total amount of bytes that the peer has uploaded in the swarm since it sent the "started" event to the tracker. This key is REQUIRED.

'downloaded':
This is a base ten integer value. It denotes the total amount of bytes that the peer has downloaded in the swarm since it sent the "started" event to the tracker. This key is REQUIRED.

'left':
This is a base ten integer value. It denotes the total amount of bytes that the peer needs in this torrent in order to complete its download. This key is REQUIRED.

'ip':
This is an OPTIONAL value, and if present should indicate the true, Internet-wide address of the peer, either in dotted quad IPv4 format, hexadecimal IPv6 format, or a DNS name.

'numwant':
This is an OPTIONAL value. If present, it should indicate the number of peers that the local peer wants to receive from the tracker. If not present, the tracker uses an implementation defined value.

'event':

This parameter is OPTIONAL. If not specified, the request is taken to be a regular periodic request. Otherwise, it MUST have one of the three following values:

'started':
: The first HTTP GET request sent to the tracker MUST have this value in the "event" parameter.

'stopped':
: This value SHOULD be sent to the tracker when the peer is shutting down gracefully.

'completed':
: This value SHOULD be sent to the tracker when the peer completes a download. The peer SHOULD NOT send this value if it started up with the complete torrent.

5.2 Response

Upon receiving the HTTP GET request, the tracker MUST respond with a document having the "text/plain" MIME type. This document MUST contain a bencoded dictionary with the following keys:

'failure reason':
: This key is OPTIONAL. If present, the dictionary MUST NOT contain any other keys. The peer should interpret this as if the attempt to join the torrent failed. The value is a human readable string containing an error message with the failure reason.

'interval':
: A peer must send regular HTTP GET requests to the tracker to obtain an updated list of peers and update the tracker of its status. The value of this key indicated the amount of time that a peer should wait between to consecutive regular requests. This key is REQUIRED.

'complete':
: This is an integer that indicates the number of seeders. This key is OPTIONAL.

'incomplete':
: This is an integer that indicates the number of peers downloading the torrent. This key is OPTIONAL.

'peers':
: This is a bencoded list of dictionaries containing a list of peers that must be contacted in order to download a file. This key is REQUIRED. It has the following structure:

'peer id':
: This is a REQUIRED string value containing the self-designated ID of the peer.

'ip':
: This is a REQUIRED string value indicating the IP address of the peer. This may be given as a dotted quad IPv4 format, hexadecimal IPv6 format or DNS name.

'port':
: This is an integer value. It must contain the self-designated port number of the peer. This key is REQUIRED.

TOC

6. The Peer Wire Protocol

The aim of the PWP, is to facilitate communication between neighboring peers for the purpose of sharing file content. PWP describes the steps taken by a peer after it has read in a metainfo file and contacted a tracker to gather information about other

peers it may communicate with. PWP is layered on top of TCP and handles all its communication using asynchronous messages.

6.1 Peer Wire Guidelines

PWP does not specify a standard algorithm for selecting elements from a clients neighboring peers with whom to share pieces, although the following guidelines are expected to be observed by any such algorithm:

> The algorithm should not be constructed with the goal in mind to reduce the amount of data uploaded compared to downloaded. At the very least a peer should upload the same amount that it has downloaded.
>
> The algorithm should not use a strict tit-for-tat schema when dealing with remote peers that have just joined the swarm and thus have no pieces to offer.
>
> The algorithm should make good use of both download and upload bandwidth by putting a cap on the number of simultaneous connection that actively send or receive data. By reducing the number of active connections, TCP congestion can be avoided.
>
> The algorithm should pipeline data requests in order so saturate active connections.
>
> The algorithm should be able to cooperate with peers that implement a different algorithm.

6.2 Handshaking

The local peer opens a port on which to listen for incoming connections from remote peers. This port is then reported to the tracker. As BTP/1.0 does not specify any standard port for listening it is the sole responsibility of the implementation to select a port.

Any remote peer wishing to communicate with the local peer must open a TCP connection to this port and perform a handshake operation. The handshake operation MUST be carried out before any other data is sent from the remote peer. The local peer MUST NOT send any data back to the remote peer before a well constructed handshake has been recognized according to the rules below. If the handshake in any way violates these rules the local peer MUST close the connection with the remote peer.

A handshake is a string of bytes with the following structure:

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

Name Length:
: The unsigned value of the first byte indicates the length of a character string containing the protocol name. In BTP/1.0 this number is 19. The local peer knows its own protocol name and hence also the length of it. If this length is different than the value of this first byte, then the connection MUST be dropped.

Protocol Name:
: This is a character string which MUST contain the exact name of the protocol in ASCII and have the same length as given in the Name Length field. The protocol name is used to identify to the local peer which version

of BTP the remote peer uses. In BTP/1.0 the name is 'BitTorrent protocol'. If this string is different from the local peers own protocol name, then the connection is to be dropped.

Reserved:
: The next 8 bytes in the string are reserved for future extensions and should be read without interpretation.

Info Hash:
: The next 20 bytes in the string are to be interpreted as a 20-byte SHA1 of the info key in the metainfo file. Presumably, since both the local and the remote peer contacted the tracker as a result of reading in the same .torrent file, the local peer will recognize the info hash value and will be able to serve the remote peer. If this is not the case, then the connection MUST be dropped. This situation can arise if the local peer decides to no longer serve the file in question for some reason. The info hash may be used to enable the client to serve multiple torrents on the same port.
At this stage, if the connection has not been dropped, then the local peer MUST send its own handshake back, which includes the last step:

Peer ID:
: The last 20 bytes of the handshake are to be interpreted as the self-designated name of the peer. The local peer must use this name to identify the connection hereafter. Thus, if this name matches the local peers own ID name, the connection MUST be dropped. Also, if any other peer has already identified itself to the local peer using that same peer ID, the connection MUST be dropped.

In BTP/1.0 the handshake has a total of 68 bytes.

6.3 Message Communication

Following the PWP handshake both ends of the TCP channel may send messages to each other in a completely asynchronous fashion. PWP messages have the dual purpose of updating the state of neighboring peers with regard to changes in the local peer, as well as transferring data blocks between neighboring peers.

PWP Messages fall into two different categories:

State-oriented messages:
: These messages serve the sole purpose of informing peers of changes in the state of neighboring peers. A message of this type MUST be sent whenever a change occurs in a peer's state, regardless of the state of other peers. The following messages fall into this category: Interested, Uninterested, Choked, Unchoked, Have and Bitfield.

Data-oriented messages:
: These messages handle the requesting and sending of data portions. The following messages fall into this category: Request, Cancel and Piece.

6.3.1 Peer States

For each end of a connection, a peer must maintain the following two state flags:

Choked:
: When true, this flag means that the choked peer is not allowed to request data.

Interested:
: When true, this flag means a peer is interested in requesting data from another peer. This indicates that the peer will start requesting blocks if it is unchoked.

A choked peer MUST not send any data-oriented messages, but is free to send any other message to the peer that has choked it. If a peer chokes a remote peer, it MUST also discard any unanswered requests for blocks previously received from the remote

peer.

An unchoked peer is allowed to send data-oriented messages to the remote peer. It is left to the implementation how many peers any given peer may choose to choke or unchoke, and in what fashion. This is done deliberately to allow peers to use different heuristics for peer selection.

An interested peer indicates to the remote peer that it must expect to receive data-oriented messages as soon as it unchokes the interested peer. It must be noted, that a peer must not assume a remote peer is interested solely because it has pieces that the remote peer is lacking. There may be valid reasons why a peer is not interested in another peer other than data-based ones.

6.3.2 Peer Wire Messages

All integer members in PWP messages are encoded as a 4-byte big-endian number. Furthermore, all index and offset members in PWP messages are zero-based.

A PWP message has the following structure:

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

Message Length:
: This is an integer which denotes the length of the message, excluding the length part itself. If a message has no payload, its size is 1. Messages of size 0 MAY be sent periodically as keep-alive messages. Apart from the limit that the four bytes impose on the message length, BTP does not specify a maximum limit on this value. Thus an implementation MAY choose to specify a different limit, and for instance disconnect a remote peer that wishes to communicate using a message length that would put too much strain on the local peer's resources.

Message ID:
: This is a one byte value, indicating the type of the message. BTP/1.0 specifies 9 different messages, as can be seen further below.

Payload:
: The payload is a variable length stream of bytes.

If an incoming message in any way violates this structure then the connection SHOULD be dropped. In particular the receiver SHOULD make sure the message ID constitutes a valid message, and the payload matches the the expected payload, as given below.

For the purpose of compatibility with future protocol extensions the client SHOULD ignore unknown messages. There may arise situations in which a client may choose to drop a connection after receiving an unknown message, either for security reasons, or because discarding large unknown messages may be viewed as excessive waste.

BTP/1.0 specifies the following messages:

6.3.3 Choke

This message has ID 0 and no payload. A peer sends this message to a remote peer to inform the remote peer that it is being choked.

6.3.4 Unchoke

This message has ID 1 and no payload. A peer sends this message to a remote peer to inform the remote peer that it is no longer being choked.

6.3.5 Interested

This message has ID 2 and no payload. A peer sends this message to a remote peer to inform the remote peer of its desire to request data.

6.3.6 Uninterested

This message has ID 3 and no payload. A peer sends this message to a remote peer to inform it that it is not interested in any pieces from the remote peer.

6.3.7 Have

This message has ID 4 and a payload of length 4. The payload is a number denoting the index of a piece that the peer has successfully downloaded and validated. A peer receiving this message must validate the index and drop the connection if this index is not within the expected bounds. Also, a peer receiving this message MUST send an interested message to the sender if indeed it lacks the piece announced. Further, it MAY also send a request for that piece.

6.3.8 Bitfield

This message has ID 5 and a variable payload length. The payload is a bitfield representing the pieces that the sender has successfully downloaded, with the high bit in the first byte corresponding to piece index 0. If a bit is cleared it is to be interpreted as a missing piece. A peer MUST send this message immediately after the handshake operation, and MAY choose not to send it if it has no pieces at all. This message MUST not be sent at any other time during the communication.

6.3.9 Request

This message has ID 6 and a payload of length 12. The payload is 3 integer values indicating a block within a piece that the sender is interested in downloading from the recipient. The recipient MUST only send piece messages to a sender that has already requested it, and only in accordance to the rules given above about the choke and interested states. The payload has the following structure:

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

6.3.10 Piece

This message has ID 7 and a variable length payload. The payload holds 2 integers indicating from which piece and with what offset the block data in the 3rd member is derived. Note, the data length is implicit and can be calculated by subtracting 9 from the total message length. The payload has the following structure:

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

6.3.11 Cancel

This message has ID 8 and a payload of length 12. The payload is 3 integer values indicating a block within a piece that the sender has requested for, but is no longer interested in. The recipient MUST erase the request information upon receiving this messages. The payload has the following structure:

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

6.4 The End Game

Towards the end of a download session, it may speed up the download to send request messages for the remaining blocks to all the neighboring peers. A client must issue cancel messages to all pending requests sent to neighboring peers as soon as a piece is downloaded successfully. This is referred to as the end game.

A client usually sends requests for blocks in stages; sending requests for newer blocks as replies for earlier requests are received. The client enters the end game, when all remaining blocks have been requested.

6.5 Piece Selection Strategy

BTP/1.0 does not force a particular order for selecting which pieces to download. However, experience shows that downloading in rarest-first order seems to lessen the wait time for pieces. To find the rarest piece a client must calculate for each piece index the number of times this index is true in the bitfield vectors of all the neighboring peers. The piece with the lowest sum is then selected for requesting.

6.6 Peer Selection Strategy

This section describes the choking algorithm recommended for selecting neighboring peers with whom to exchange pieces. Implementations are free to implement any strategy as long as the guidelines in Section 6.1 are observed.

After the initial handshake both ends of a connection set the Choked flag to true and the Interested flag to false.

All connections are periodically rated in terms of their ability to provide the client with a better download rate. The rating may take into account factors such as the remote peers willingness to maintain an unchoked connection with the client over a certain period of time, the remote peers upload rate to the client and other implementation defined criteria.

The peers are sorted according to their rating with regard to the above mentioned scheme. Assume only 5 peers are allowed to download at the same time. The peer selection algorithm will now unchoke as many of the best rated peers as necessary so that exactly 5 of these are interested. If one of the top rated peers at a later stage becomes interested, then the peer selection algorithm will choke the the worst unchoked peer. Notice that the worst unchoked peer is always interested.

The only lacking element from the above algorithm is the capability to ensure that new peers can have a fair chance of downloading a piece, even though they would evaluate poorly in the above schema. A simple method is to make sure that a random peer is selected periodically regardless of how it evaluates. Since this process is repeated in a round robin manner, it ensures that ultimately even new peers will have a chance of being unchoked.

TOC

7. Security Consideration

This section examines security considerations for BTP/1.0.The discussion does not include definitive solutions to the problems revealed, though it does make some suggestions for reducing security risks.

7.1 Tracker HTTP Protocol Issues

The use of the HTTP protocol for communication between the tracker and the client makes BTP/1.0 vulnerable to the attacks mentioned in the security consideration section of **RFC 2616**[6].

7.2 Denial of Service Attacks on Trackers

The nature of the tracker is to serve many clients. By mounting a denial of service attack against the tracker the swarm attached to the tracker can be starved. This type of attack is hard to defend against, however, the metainfo file allows for multiple trackers to be specified, making it possible to spread the load on a number of trackers, and thus containing such an attack.

7.3 Peer Identity Issues

There is no strong authentication of clients when they contact the tracker. The main option for trackers is to check peer ID and the IP address of the client. The lack of authentication can be used to mount an attack where a client can shut down another client if the two clients are running on the same host and thus are sharing the same IP address. In addition, a rogue peer may masquerade its identity by using multiple peer IDs. Clients should there refrain from taking the peer ID at face value.

7.4 DNS Spoofing

Clients using BTP/1.0 rely heavily on the Domain Name Service, which can be used for both specifying the URI of the tracker and how to contact a peer. Clients are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. Clients need to be cautious in assuming the continuing validity of an IP address/DNS name association.

In particular, BTP/1.0 clients SHOULD rely on their name resolver for confirmation of an IP number/DNS name association, rather than caching the result of previous host name lookups. If clients cache the results of host name lookups in order to achieve a performance improvement, they MUST observe the TTL information reported by DNS.

If clients do not observe this rule, they could be spoofed when a previously-accessed peers or trackers IP address changes. As network renumbering is expected to become increasingly common according to **RFC 1900**[2], the possibility of this form of attack will grow. Observing this requirement reduces this potential security vulnerability.

7.5 Issues with File and Directory Names

The metainfo file provides a way to suggest a name of the downloaded file for single-file torrents and the top-most directory for multi-file torrents. This functionality is very like the Content-Disposition header field documented in **RFC 2183**[4] and the security considerations mentioned in this RFC also apply to BitTorrent clients. In short, BTP clients SHOULD verify that the suggested file names in the metainfo file do not compromise services on the local system. Furthermore, care must be taken for multi-file torrents to validate that individual files are relative to the top-most directory and that the paths do not contain path elements to the parent (that is directory elevators such as ``..''), which can be used to place files outside the top-most directory.

Using UNIX as an example, some hazards would be:

- Creating startup files (e.g., ".login").
- Creating or overwriting system files (e.g., "/etc/passwd").
- Overwriting any existing file.
- Placing executable files into any command search path (e.g., "~/bin/more").
- Sending the file to a pipe (e.g., "| sh").

It is very important to note that this is not an exhaustive list; it is intended as a small set of examples only. Implementers must be alert to the potential hazards on their target systems. In general, the BTP client SHOULD NOT name or place files such that they will get interpreted or executed without the user explicitly initiating the action.

7.6 Validating the Integrity of Data Exchanged Between Peers

By default, all content served to the client from other peers should be considered tainted and the client SHOULD validate the integrity of the data before accepting it. The metainfo file contains information for checking both individual pieces using SHA1, and optionally individual files using MD5. SHA1, being the strongest of the two, is preferred. Furthermore, sole reliance on whole-file checking can potentially render otherwise valid pieces invalid, and should only be considered for small files, to limit the amount of data being discarded.

Trusting the validity of the resulting file or files ends up being a matter of trusting the content of the metainfo file. Ensuring the validity of the metainfo file is beyond the scope of this document.

7.7 Transfer of Sensitive Information

Some clients include information about themselves when generating the peer ID string. Clients should be aware that this information can potentially be used to determine whether a specific client has a exploitable security hole.

---

8. IANA Considerations

TOC

This document makes no request of IANA.

---

9. References

TOC

[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**).

Authors' Addresses

TOC

Jonas Fonseca
DIKU
Email: **fonseca@diku.dk**

Basim Reza
DIKU
Email: **basim@diku.dk**

Lilja Fjeldsted
DIKU
Email: **lilja@diku.dk**