Scott Felch
Spring 2015
CSCI 467: Networks 2
Term paper

BitTorrent Protocol -- BTP/1.0 and BEPS Extensions

Abstract: In this paper I will describe the BitTorrent Protocol version 1.0, commonly referred to as "BTP/1.0". BTP/1.0 is a peer-to-peer (P2P) file distribution protocol written by Bram Cohen in 2001 with the intent of replacing FTP for large file transfers. Today, BT makes up 25-30% of all internet traffic. In addition to the technical details of the protocol, the paper will discuss the policies surounding additions and modifications to the Bit Torrent codebase or community.

BitTorrent (BT) is the world's most widely used peer-to-peer file distribution protocol (Sandvine, 2014). Invented by Bram Cohen in February 2001, it was designed with the intention of making it easier to distribute large amounts of data to many people at once than with the traditional FTP or HTTP client-server paradigm. Rather than a hub-spoke model, with a single server uploading a file to many users, BT is in the form of a mesh model. A torrent may contain one or many files, but for consistency within this paper we'll refer to a download as being a single file. This lightens the workload on the original uploader, known as a "seed". As the seed sends pieces of the file to different clients (known as "leeches"), they are also sending those pieces to other leeches. While the file spreads, leeches who complete the download become seeds. The file propagates as more users join and complete the torrent. A network of users downloading a file from a torrent is called a "swarm". As more users join the swarm, more upstream bandwidth becomes available and average download speeds for all users will increase, until connection saturation (Cohen, 2013).

In the original specification for BTP/1.0 the swarm was still dependent on the tracker in order to continue to function, as otherwise there would be no way to coordinate and distribute metainfo between peers. However later additions to the BT protocol have been made since then to move to a nearly completely decentralized system. This way, even if the torrent tracker and the original seeder(s) go offline the swarm can still survive (Cohen, 2013). Another advantage to the decentralized nature of the swarm is that it provides tremendous amounts of fault tolerance. It is for these reasons that BT has grown to such a massively popular protocol. As of 2014, BT accounts for approximately 25% of all upstream traffic on the internet. Down from 33% in 2008, the decline in share can be attributed to rising popularity of video streaming services, with BT being recently overtaken by Netflix as largest draw of total aggregate internet bandwidth (Sandvine, 2014). Though often associated with the social stigma of being exclusively a tool for piracy, BT is an efficient, robust and reliable protocol to maximize efficiency of any kind of large file distribution.

So where do torrents come from? In order for any torrent to function, there are six entities which must be present: a *.torrent metainfo file, a webserver to host the .torrent file, a BT tracker (usually the same server as the webserver), an original seed user, and an end user with a web browser and torrent client. The webserver functions as the visual front-end which allows a user to navigate to wherever the .torrent file is located, whereas the torrent tracker server is accessed by client indrectly by opening a metainfo file using the BT client. The metainfo file is generated by the host user using the BT client based on the files being created into a torrent, and then uploaded that to the webserver. When a user downloads the .torrent file and opens it in their client, it will connect them to other users in the swarm. The original host’s client will upload the data to incoming users to get the swarm started (Fonseca, 2005).

You may be wondering at this point what is contained in a metainfo file. The metainfo file utilizes *bencoding* (pronounced “bee encoding”, also invented by Bram so he named it after himself) to store data types in plaintext strings. This provides a means of easily storing and transmitting data structures. The augmented BNF syntax for the bencoding format is as follows (RFC 2234):

```
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"
```

(Crocker, Overell, 1997).

It looks confusing, but is easier to understand with some examples. A string is represented by a leading integer indicating the length of the string, a colon, and then the string. For instance, “8:announce” is a frequently used message. An integer such as 420 would be represented with a leading ‘i’ to indicate an upcoming integer, the digits 420, and a trailing ‘e’ to indicate the end of the integer. Dictionaries are extremely important, as these are used for storing verification hashes for torrent data. A dictionary is represented in string form by a leading ‘d’, a series of key:value pairs separated by colons, and then a trailing ‘e’. An example would be “d5:monthi4e4:name5:aprile”, which translates to a dictionary containing “month” => ‘4’ and “name” => “april”. For purposes of processing efficiency and ease of manipulating the data structure, dictionary contents must be sorted. (Fonesca, 2005).

The metainfo file uses UTF-8 encoded strings to represent two bencoded dictionaries, called “announce” and “info”. The contents of these dictionaries are as follows:

- “announce”: contains the URL of the tracker
- “info”: maps to a dictionary containing information about the files in the torrent.
    - “name”: UTF-8 encoded string containing the suggested name to use to save file/directory
    - “piece_length”: indicates the size of the pieces used in the torrent. All pieces are uniformly sized, with the exception of the final piece of the torrent which may be truncated
    - “pieces”: a string whose length is a multiple of 20, to allow for an arbitrary number of 20 character substrings. These substrings contain the SHA1 hashes of pieces with the corresponding index
    - Single-file torrents will contain:
        - “length”: a dictionary containing representation of filesize in bytes
    - Multi-file torrents will contain:

- "files": a dictionary which maps out the subdirectory structure of the torrent. For implementation of piece/block indexing, the individual files are treated as being all concatenated into a single binary. (Cohen, 2013).

In order to distribute the file in a non-linear fashion, the torrent contents are divided up into segments called *pieces*, which are also divided into smaller segments called *blocks*. The size of a piece is defined by the creator of a torrent by choosing the number of pieces, and follows the format:

fixed_piece_size = size_of_torrent / number_of_pieces

Like many aspects of the BTP/1.0 protocol, there is room for interpretation on what is a "correct" size for pieces to be. Smaller pieces will result in more pieces being tracked in the metainfo file, and thus increasing the size of the metainfo file. A generally accepted best practice is to ensure the metainfo file stays under 70 kB to minimize the amount of overhead costs incurred from managing an excessive number of TCP connections for short periods of time. The size of a block is implementation-dependent. The formula for the number of blocks produced per piece is as follows:

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

Similarly, to find the index of a desired block within a piece, there is the formula:

block_index = block_offset % fixed_block_size

This explains how a metainfo file is created and structured. The next step is for a client to connect to the swarm and begin downloading. This is accomplished through simultaneously utilizing two different protocols, *Tracker HTTP Protocol (THP)*, and *Peer Wire Protocol (PWP)*. THP is used by a client to get necessary information from the tracker to join the swarm, as well as maintain information about what other users are in the swarm. PWP is used to connect to remote peers to exchange status information and send data payloads. The nature of THP is the Achille's heel of BTP/1.0, as it is a centralized server that must be operational for the swarm to survive. This was addressed in later enhancements to the protocol to create a (almost) completely decentralized nature, which will be addressed later in this paper.

To connect to the swarm, a peer will send an HTTP GET request to the URL located in the "announce" entry of the .torrent file. The contents of this GET request will vary based on whether the client is joining for the first time, or has a partially completed download already. Improperly structured requests will be discarded by the server and the user will be unable to join the swarm. The body of the request must contain the following information:

- "info_hash": Required value, contains a 20-bye SHA1 hash of the "info" key in the metainfo file. This ensures validation of data integrity.
- "peer_id": Required value, contains a 20-byte string of a self-defined unique identifier for the client
- "port": Required value, since BT does not define default ports, the port desired must always be explicitly requested
- "uploaded": Required value, it is a base 10 integer denoting the total number of bytes the peer has uploaded to the swarm since it connected to the tracker
- "downloaded": Required value, same as above but for downloading
- "left": Required value, same as above but for bytes remaining to download

- “ip”: Optional value, indicates Internet-wide IP address in IPv4 or IPv6 format, or a DNS name. This speeds up initial connection to peers
- “numwant”: Optional value, used to define the desired number of peers to connect to at a time. If undefined, the default value of 5 will be used.
- “event”: Optional value. This indicates special kinds of messages in the form of flags. If no flags are present, it is assumed this message is just so the client can check in with the server and exchange tracking statistics and records of other users in the swarm. If one of the flags is set, then it indicates a specific event occurring.
    - “started”: A flag within “event”, it is used for the first HTTP GET request sent to the tracker.
    - “stopped”: Flag used to gracefully disconnect from server and indicated end of session
    - “completed” Flag sent to the server when a peer completes a download and transitions to a seeding role. It is not sent if the client connects with an already fully completed torrent. (Fonseca, 2005).

The tracker will respond to the client with a similarly structured HTTP GET request. It uses the same plaintext format to represent a bencoded dictionary, containing the following fields:

- “failure_reason”: Optional key, used to pass a human readable string containing a reason for a connection error. If this field is populated then every other field will be blank.
- “interval”: Required key, indicates how frequently the client must check in with tracker to update list of peers and exchange statistics. If a client does not check in with the server within the specified amount of time, they are assumed to have dropped off and the server terminates the connection.
- “complete”: Optional key, indicates number of seeds currently in the swarm
- “incomplete”: Option key, indicates number of leeches currently in the swarm
- “peers”: Required key, it is another dictionary which contains an ordered list of peers to connect to in order to get an initial file. This dictionary contains the following fields for each peer:
    - “peer_id”: Required value, string representation of the name of a remote peer
    - “ip”: Required value, contains IPv4, IPv6, or DNS name for the remote peer
    - “port”: Required value, integer representation of the port the remote peer is accepting connections on.

Now that the client has connected to the tracker over THP, the client will use metainfo retrieved from the tracker to begin to connect to peers in the swarm over PWP. PWP is a TCP-based asynchronous message passing protocol that is used for all communications from one peer to another. PWP is able to exchange some metainfo about the swarm -- such as what pieces each peer has -- which is used to determine what peers to connect to. However the swarm is still reliant on the tracker for notification of new users. The other primary usage of PWP is to distribute the data payloads of the torrent file. BTP/1.0 does not explicitly define what algorithms should be used for determining what peers to connect to, as long as the algorithms implemented adhere to the following guidelines:

- The algorithm shall not be constructed to minimize uploading to peers, or “greedy client” behavior. Ideally all clients should upload to a 1.0 ratio to perpetuate the life of the swarm.
- The algorithm shall not use a strict tit-for-tat (TFT) data exchange policy for newly connected peers. In other words, the client shall not deprive the remote peer of data for not reciprocating upload since this peer does not yet have anything to upload.
- The algorithm shall minimize TCP overhead costs and maximize efficiency of downstream and upstream bandwidth utilization by limiting the number of connections permitted at any one time.
- The algorithm shall form TCP requests into a pipeline that can be sent in rapid succession to essentially create a steady stream of incoming data, rather than bursts.
- The algorithm shall be able to safely interface with algorithms used by any other client. (Fonesca, 2005).

The client will now open a TCP port and listen for incoming connections from remote peers that have been notified of the newcomer to the swarm. Before any data payloads can be exchanged, a handshake must be performed with the remote peer to ensure both clients are operating with identical information on how this particular torrent is structured, and the packets are going to the intended peer. A handshake is defined as a 68-byte string which contains the following values:

- “name_length”: an unsigned int indicating the length of the string indicating the protocol name. This ensures that if the two clients are using incompatible protocols, it is in most cases a cheaper operation to catch it by mismatched lengths than character comparisons.
- “protocol_name”: an ASCII representation of the protocol name, the same length as the previous field. Used to inform other peers what revision of the BTP is being used.
- “reserved”: 8 bytes in the string which will be used in the future to indicate extensions running on top of the BTP
- “info_hash”: 20 bytes used as the SHA1 value of the “info” key in the .torrent file, or in other words verifying the integrity of the list of all files to be acquired.
- “peer_id”: 20 bytes used to indicate the self-defined unique name of the peer. Having a peer ID allows for a few different capabilities. First, it enables the client to easily discard packets that are received erroneously. It also ensures that if multiple users are connected to swarms using the same IP and port, data payloads can still be directed to the proper recipient.

If any of these fields are empty or contain invalid data, the remote peer will drop the connection immediately. Strict adherence to policy at this stage protects the swarm from errors that would be induced by incompatible protocols, such as wasted bandwidth usage or corrupted downloads. (Harrison, 2008).

After the PWP handshake has been completed, the peer on each end of the TCP channel is now able to utilize PWP to asynchronously communicate back and forth. The messages transferred will be in two categories: state-oriented messages, and data-oriented messages. State-oriented messages communicate metainfo about the client’s state, such as what files it has and is looking for. Data-oriented messages are requests for files or data payloads. All messages are structured with a leading integer indicating message length in bytes, a big Endian 4-byte integer value indicating the message ID, and a variable-length data payload, if applicable. The message length field includes the data payload, and excludes the size of the message length indicator. Reserved values are used to indicate special scenarios. A message length of 1 indicates there is no data payload attached to the message, and it is merely a flag update. A message length of 0 is purely a “stay alive” message sent to the server to avoid an interval timeout, if no other activity is occurring. A client receiving improperly formatted PWP messages will drop the connection. (Fonseca, 2005).

Peer states are a set of flags that must always be properly defined in order to inform remote peers of changes in any neighboring peers. Peer states are a subset of the messages allowed by PWP. The possible PWP peer state messages are as follows:

- “choke”: ID 0, no payload. Sent to remote peer to notify they’re being choked
- “unchoke”: ID 1, no payload. Sent to remote peer to notify they’re being unchoked
- “interested”: ID 2, no payload. Sent to remote peer to indicate that the client will request pieces as soon as it is unchoked.
- “uninterested”: ID 3, no payload. Sent to remote peer to inform which files the client already has or has chosen not to download from the torrent.
- “have”: ID 4, payload length 4. Payload is a number indicating index of a piece the peer successfully downloaded and validated. Remote peer will evaluate and drop if invalid (ie, index out of bounds), or express interest if they don’t have the piece.
- “bitfield: ID 5, variable length payload. Sends bytes indicating pieces of the torrent which are and are not acquired, represented by 1s (acquired) and 0s (not acquired). Cite here

The data-oriented messages are used to transmit metainfo on which pieces a client is looking for, and then subsequently transfer the data payload of those pieces. The possible PWP data-oriented messages are as follows:

- “request”: ID 6, payload length 12. This is the first data-oriented message. The payload is 3 integers, containing a piece index, block offset, and block length to indicate a piece of the torrent that the sender is interested in receiving. Piece messages can not be sent proactively, only in response to request messages.
- “piece”: ID 7, variable length payload. Contains two ints: a piece index, and block offset. The payload is a block of requested data.
- “cancel”: ID 8, payload length 12. Contents: piece index, block offset, block length. Tells receiving peer that sender is no longer interested in a piece of file, either due to receiving it from a different remote peer, or choosing not to download that file. The recipient of the cancel message must purge all stored request information for the sender.

Choking sounds like a strange concept, and is perhaps unfortunately named, but is an essential part of keeping the swarm functioning. The total number of remote peers a client is connected to at any given time is user definable, but defaults to 20. However the number of remote peers being actively transferred to is known as the *neighborhood*, and is user definable but defaults to 5. By limiting the number of active transfers to a small number, TCP overhead is minimized and bandwidth is used more effectively. The purpose of having the other connections is so that the client can periodically poll the other non-neighborhood remote peers and see which have more pieces available, are uploading at a faster rate, and have not been actively choking the client. If a remote peer is detected with better performance than the worst performer in the current neighborhood, the worst performer’s connection will be terminated and the client will establish a connection with the new peer. As a result of this process, bandwidth utilization is maximized as high-bandwidth peers tend to be matched with each other. In addition, peers which are exhibiting greedy behavior -- meaning utilizing a peer connection algorithm to minimize their data uploaded – will be punished and deprived of data from peers. This is known as the tit-for-tat (TFT) policy, to ensure fairness of distribution of uploading, however there is an exception to this rule. Every time a new set of peers is acquired into the neighborhood, there is one selected at random from the tracker's pool, instead of by evaluating stats. A small number of random selections of peers provides newcomers the opportunity to receive pieces even though they have nothing to upload, and also serves to prevent subgraph fragmentation within the swarm. In other words, subgragph fragmentation is when there are complete copies of the file available in the swarm, but only being distributed amongst high-bandwidth peers and low-bandwidth peers are unable to progress any further. (Cohen, 2013).

Piece selection is another intentionally ambiguously defined aspect of BTP/1.0, so that different clients may have different algorithms to approach the problem. However the algorithm used by the original BT client, and most other variant clients currently available, is known as Rarest First (RF). Rarity is calculated by summing the number of times a piece index appears in neighbors' bitfields and chooses the piece with the lowest sum. Without RF there is a much higher rate of the frustrating situation where a download ceases to progress if all the seeds drop off. Even if every seed drops off the swarm, as long as there is at least one copy of every piece in the swarm somewhere, the leeches will be able to finish their downloads and become seeds. The only problem is that RF doesn't behave in quite as desirable a way at the very end of a torrent, which is what necessitates a modified version of this protocol known as End Game (EG).

End Game is initialized when there is a single piece remaining in the download. The client will send out requests for the blocks it needs to every single peer it is connected to, even those not in its neighborhood. After a remote peer responds and begins transmitting the final blocks, the client will send cancel messages to all the other remote peers it just spammed. At this point the client has completely downloaded the file, become a seed, and continues to upload to support the swarm.

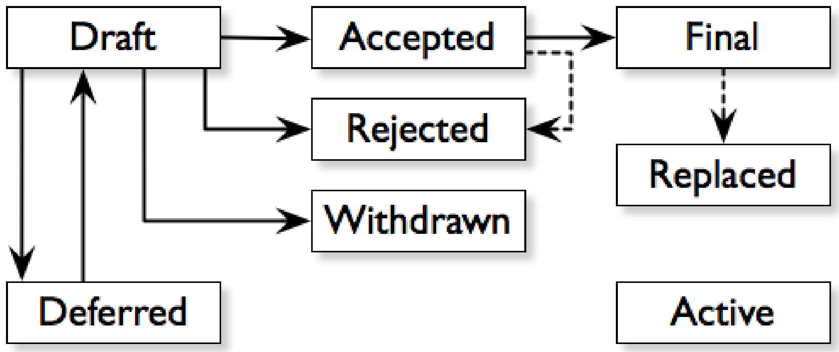
This concludes how Bit Torrent Protocol 1.0 functions in a typical download scenario. However there have been a number of protocol patches and extensions which build upon the protocol introduced since then. Suggested changes are known as BitTorrent Enhancement Proposals (BEPs), and must go through the BEPS Process to be approved. (Harrison, 2008). It was through this process that massive improvements have been introduced, such as magnet links and Distributed Hash Tables (DHT) to enable functionality of decentralized, tracker-less torrents. (Loewenstern, 2013).

Conceptually, a BEP is essentially identical to an Internet Request For Comments (RFC), with the exception of the representative governing figures. (Robinson, 2014). The styling of the document and process of approval is based heavily on Python's PEP documentation and workflow pattern. (Harrison, 2008). There are three different categories of BEPs:

- Standards Track: describes an extension or behavioral change of a client, tracker, or web server
- Informational Track: describes a BT design issue, discovery of new information, or suggestions of best practices. Unlike Standards BEPs, there is no call to action, and the community may choose to engage or ignore the BEP at their leisure.
- Process Track: similar to a Standards BEP, but pertaining to a process surrounding BT rather than directly pertaining to the technical implementation of BT. An example would be a change to

policy or guidelines, decision-making and workflow management, or changes to toolsets used with BT.

Any idea that will eventually become an accepted and finalized BEP has to go through a rigorous vetting process, to be reviewed by many stakeholders and BT community members to ensure a high degree of quality. It helps to have a visual to conceptualize the process:



(Harrison, 2008).

The BEP proposal starts with a user submitting an idea to the BT forums (located at forums.bittorrent.org) for casual discussion. If the idea gains traction there, then the BEP must be assigned a product champion, which is generally the original author of the BEP. The champion will then submit a proposed title and draft of the formal BEP to the BEP editors. All the tracks require a design document, however standards BEPs additionally require a reference implementation. The implementation of the proposed change doesn't have to be complete or bulletproof, but it must be fully functional in at least one BT client and the source code must be publically available.

There are currently two BEPs editors, David Harrison and Arvid Norbeg, who are responsible for vetting all proposed BEPs. There are quite a few reasons a BEP may be rejected, but the most common are as follows:

- Too broad/vague (for instance, proposing more than one change in a single BEP)
- Duplication of existing functionality
- Serious technical flaws or infeasibility
- Not backwards compatible with existing standards
- The BDFL doesn't like it.

The BDFL is Bram Cohen himself, the self-appointed Benevolent Dictator For Life. He reserves the right to veto decisions made by the editors. BEPs which are rejected are permitted to be resubmitted after a mandatory forums feedback session and revision process. The criteria for BEP acceptance are few, but broad. The BEP must:

- Demonstrate a clear, complete description of proposed enhancement
- Represent a net improvement to BT
- Standards BEPs must be functional and tested in live swarms
    - It's recommended to include statistical results from analyses, test bed benchmarks and event simulations.

    - The design document should include rationale behind the proposal, and briefly summarize the results to illustrate the enhancement.
    - Results should go into just enough depth to explain the proposal. Beyond that it is recommended that the champion write an Info BEP or publish a research paper.

Once a BEP is accepted by the editors, it is assigned a BEP number and a label track. It is flagged with the status of "draft" and a data repository is created on GitHub to track the revisions and life cycle of the document. It is up to the BEP's champion to get community support or contribution to the project. After a sufficient number of revisions, the editors will assign it to a final stage. The different possibilities are:

- Active
    - Deployment of new Finalized standard has occurred.
- Deferred
    - Good idea, but nobody actively working on it.
    - Fully functioning implementation is ready to deploy.
- Finalized
    - Implementation is finalized and ready to deploy.
    - Editors and BDFL sign off.
- Rejected
    - Idea is not feasible to implement, but was a good enough idea to be worth keeping around as a record of posterity. This also ensure that other people won't inadvertently propose an already failed idea.
- Replaced
    - Able to replace existing BEPS (ex: to upgrade an API to v.2.0)
    - Officially document the replacement of an old BEP with a new BEP

The last stage of Finalization is defining the legal licensing of the idea. BT is open source and released under public domain, and consequently all proposed changes to BT must be as well. In order for a BEP to be Finalized, the author, editors, and Bram must sign a contract to give up all intellectual property rights to the public domain. It may sound extreme, but it is a powerful sign of dedication to open source and free software. This explains where BEPs come from and how they work, so now let's look at the most exciting things to come out of BEPs yet. (Harrison, 2008)

BEP0009 describes "Extension for Peers to Send Metadata Files", or the introduction of magnet links for the initiation of a client joining a swarm as an alternative to a .torrent file. This is closely related to BEP0005, which defines the "DHT Protocol for Decentralized P2P tracking". A magnet link is just a simple URL, not a file. It is like a stripped down, flattened .torrent file. The URL contains information for the client to establish initial connections with the swarm, but provides no metainfo of the files in the torrent or ability to continuously track peers. Upon connection to the swarm, the client is able to retrieve the file metainfo from peers. This is done by an extension to the handshake messages by utilizing some of the reserved bits in the original BTP/1.0 specification. The inability for continued tracking is the biggest glaring weakness, which is where the concept of the Distributed Hash Table comes in. (Hazel, Norberg 2012).

The DHT is a very complex and intricate data structure and unfortunately explaining it in full detail would be well outside the scope of this paper, but the general idea is not incomprehensible. Essentially the job of a tracker is distributed so that all of the nodes of the network handle the job cooperatively. The DHT creates a separate UDP-based P2P network that exists alongside the existing TCP-based PWP network. An important clarification to make to avoid confusion is that clients on the PWP network are called “peers”, and clients on the DHT network are called “nodes”. PWP messaging is still used for transmitting data payloads between peers, but now DHT takes over the responsibilities that were previously handled by the tracker. Much like a single tracker will be responsible for metainfo of many torrents, a DHT also contains metainfo about many torrents as well, not just the one the client is currently connected to. When a peer connects to a torrent, in order to gain connections to other peers the DHT node will send out a RPC request, which is basically just like addressing a dictionary locally except it is stored in a distributed fashion amongst the other nodes. The key being sent out as a request is the hash of the “info” file describing all the files in the torrent. The value going to be returned is a list of IP addresses and ports of peers who are in the same swarm. Like most operations on the internet, it is not a direct jump from point A to point B, and it takes a few hops to find the information needed. Every node will have a cache of peers they are aware of, and if they have one matching the key that the node is requesting, they send back the peer information. If the receiving node does not have contact information for that swarm, they will reply with a node address that is a shorter “distance” away. Distance, in this context, is not referring to geographical location but rather indexing values getting closer and closer to the desired data. (Loewenstern, Norberg, 2013).

DHT is somewhat misleadingly advertised as allowing a swarm complete independence of any centralized server, but unfortunately that is not entirely correct. While a swarm running on DHT is capable of sustaining itself without a tracker, for the initial creation of the torrent the first user must bootstrap their way into the DHT network. This is done by connecting to one of a few servers hosted by BitTorrent.org and a few other network-related companies. The data transmitted is only a few bytes, and once the single peer is on the DHT network then every subsequent peer will have no need to connect to the DHT bootstrap server. Though not a perfect replacement for a tracker, it is a substantial improvement.

BitTorrent was a truly revolutionary technology when it came out, and over a decade later is still having a tremendous impact on the world. What many people take for granted as just a progress bar moving across the screen is the result of a beautiful work of engineering and design. The forethought put into ensuring technical and procedural modularity enables it to continue to be a relevant and successful protocol, and we will surely see it continue to grow and evolve in the coming years. The most unfortunate part of BitTorrent is the social stigma associating it with being a tool for piracy, when in actuality it could be used to improve bandwidth utilization in countless computing applications. Some companies such as Blizzard Entertainment have adopted torrent-like protocols in their distribution platforms, and Linux communities often use torrents to distribute OS images. It will be exciting to see who else will take advantage of this unique technology in the years to come.

Glossary

BDFL: Benevolent Dictator For Life, the title of Bram Cohen who will forever be in charge of BitTorrent.

bencoding: a means of storing data structures such as lists, dictionaries, arrays, etc in a flat text file for storage and transmission of tracking metainfo

BEP: BitTorrent Enhancement Proposal. Public, peer-reviewed suggestion for an improvement or addition to the Bit Torrent protocol, surrounding environment, procedures, or toolset.

bitfield: a byte or collection of bytes used as a series of binary flags to map out which portions of a file are available or not

block: a subset of a piece

choke: refusing to upload to a peer

client: software program used to connect to a torrent as a peer. The original BitTorrent client was written initially in Python, and later ported to C++. However there are now dozens of paid, freeware, and open source competing products such as µTorrent, Azeurus, BitComet, Transmission, and Vuze. As a result, the vanilla BitTorrent client is now often referred to simply as Mainline. Clients vary in terms of license, operating system supported, language used for implementation, user interface, and some variation in network protocols.

DHT: distributed hash table. A data structure of (key, value) pairs, contains a SHA-1 hash of the filename (the key) and the value is the identity of the node(s) in possession of that file. Data is propagated from node to node so every node is always in possession of a routing table of known good nodes.

End Game: when downloading the final piece of a torrent, the client sends out requests for blocks to every peer it is connected to, rather than just the neighborhood. After receiving the first response, the client subsequently sends out cancellation messages.

FTP: File Transfer Protocol, a means of transferring large files more reliably than supported over HTTP. This is the protocol BitTorrent aimed to replace.

leech: a user who is in process of downloading the file

metainformation: information about what files are in the torrent, how big they are, what the filenames are, who else is connected to the torrent, bandwidth statistics, etc

neighbors: peers directly connected to a user and actively transferring

node: a user connected to the DHT network (tracking info)

peer: a user connected to the PWP network (data payload)

piece: segments of the file broken up into equal user-determined size, except the final piece which may be truncated

PWP: Peer Wire Protocol, a TCP-based protocol that allows peers on a torrent to exchange file availability metainfo and data payloads

RF: Rarest First. BitTorrent's algorithm for finding the most uncommon piece of a file and acquiring that, to spread it and make it less rare

RFC: Request For Comments, a publication of the Internet Engineering Task Force and Internet Society used for proposing changes to technology, infrastructure, or policy regarding internet. As the name suggests, the posting is an invitation for feedback and criticism to better the idea being presented.

seed: a user who's completed the file download, and is purely uploading

swarm: the collection of users connected to a torrent

TFT: Tit-For-Tat, algorithm for discouraging greedy downloaders, refuses to upload data to someone who isn't also uploading back

THP: Tracker HTTP Protocol, uses HTTP GET requests to communicate information between client/server to connect to other peers in the swarm

torrent: (noun) a file or set of files to be downloaded by a swarm of people

(verb) to download files using BitTorrent

tracker: website that holds .torrent and magnet files, which contain metainformation to get a user connected to others in the swarm and start downloading the torrent. Trackers come in public form, where anyone can browse to the website and locate torrent files and magnet links, as well as private trackers that require an invitation code to register an account. Private trackers often have greater diversity in content and more reliable seeds, as bandwidth usage and ratios are tracked per user to increase accountability.

Works Cited

- Jonas Fonseca. 2005. BitTorrent Protocol version 1.0. (April 2005). Retrieved June 9, 2015 from http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html
- Ned Freed, Nathaniel Borenstein. 1996. Internet RFC 2046: Multipurpose Internet Mail Extensions Part 2, Media Types. (November 1996). Retrieved June 9, 2015 from http://tools.ietf.org/html/rfc2046
- Crocker, D. and P. Overell. 1997. RFC 2234: Augmented BNF for Syntax Specifications: ABNF. (Nov 1997) Retrieved June 9, 2015 from ftp://ftp.isi.edu/in-notes/rfc2234.txt
- Paul Robinson. 2014. Request for Comments. (May 2015). Retrieved June 9, 2015 from http://en.wikipedia.org/wiki/Request_for_Comments
- David Harrison. 2015. Index of BitTorrent Enhancement Proposals. (Jan 2015). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0000.html
- David Harrison. 2008. The BitTorrent Enhancement Proposal Process. (May 2008). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0001.html
- David Harrison. 2008. Sample reStructured Text BEP Template. (May 2008). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0002.html
- Bram Cohen. 2013. The BitTorrent Protocol Specification. (Oct 2013). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0003.html
- David Harrison. 2008. Assigned Numbers. (Sept 2008). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0004.html
- Andrew Loewenstern, Arvid Norberg. DHT Protocol. (April 2013). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0005.html
- Greg Hazel, Arvid Norberg. Extension for Peers to Send Metadata Files. (Oct 2012). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0009.html
- Arvid Norberg, Ludvig Strigeus, Greg Hazel. Extension Protocol. (Feb 2008). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0010.html
- David Harrison. Tracker Returns Compact Peer Lists. (Sept 2008). Retrieved June 9, 2015 from http://www.bittorrent.org/beps/bep_0023.html
- Sandvine Intelligent Broadband Networks. 2014. Sandvine Global Internet Phenomena Report. (Nov 2014). Retrieved June 9, 2015 from https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf