BitTorrent Architecture and Protocol

Ryan Toole
CIS 475: Spring 2006
Vinod Vokkarane
University of Massachusetts Dartmouth
April 17, 2006

Abstract

BitTorrent is a new popular application layer network protocol used to distribute files. BitTorrent is efficient at content delivery by maximizing the upload utilization and by preventing unfairness. This paper discusses the BitTorrent architecture and protocol in great detail by discussing the tracker and all the messages sent between the peers and the tracker and also between peers and peers. Also this paper will describe how BitTorrent is efficient by looking at algorithms that decide which peers to upload to and which pieces of the file to download first.

Introduction

BitTorrent is an application layer network protocol used to distribute files. It uses a peer-to-peer (P2P) network architecture where many peers act as a client and a server by downloading from peers at the same time they are uploading to others. The serving capacity increases as the number of downloaders increases making the system self-scaling [2]. It also uses a client-server architecture where peers contact the server to find other peers that they may connect to. This paper will give an overview of the BitTorrent architecture and the protocols it uses.

Background

BitTorrent was created by Bram Cohen in 2002. He created it as a way to distribute the free Linux operating system [5]. It is very different from other P2P file sharing protocols because it does not have any search functionality in the protocol. This means that users cannot search for files to download using the BitTorrent protocol. The advantage of this lack of content localization is that BitTorrent focuses on its main goal of delivering content as efficiently as possible [1].

BitTorrent does not allow users to share files directly from their computer like other P2P networks. Instead a torrent file needs to be created which represents a peer-to-peer transfer session, for a particular file or files. There is no way of searching for these torrent files by using the BitTorrent protocol. Instead most peers download these torrent files from a website, which usually hosts many torrent files and allows users to upload their own torrent files. Once a peer has the torrent file it may join the torrent session by opening this file in their BitTorrent application. A peer must be in one of two states. It is in the leecher state when it is still downloading the file while uploading pieces it has to other leechers. A peer is in the seed state if it has the complete file and is uploading to leechers. There needs to be at least one seeder in order for the torrent to be alive otherwise the leechers will not be able to finish. Although if the original seed has uploaded every piece once to only one participating peer, it is possible for the seed to leave and the leechers can finish the download by downloading all the other pieces off of other leechers. A file is split into equal size *pieces*, which are further divided into smaller *blocks*. Blocks are the transmission unit of the network, but the protocol keeps track of what pieces have been downloaded. Each peer must maintain a list of peers it is connected to, which is called the peer set. Also a peer can only upload to a subset of this peer set called the active peer set. Peers also need to know what pieces of the content each peer in its peer set has [1]. By knowing what peers a peer can upload to and by knowing what pieces all of its connected peers have, BitTorrent can use this information in order to deliver content efficiently. This results in less than a tenth

percent of bandwidth overhead and it reliably utilizes all available upload capacity [3]. A P2P network is efficient if it maximizes its upload capacity, a good piece selection algorithm, and it is fair. By being fair, a peer can not download too much more than it has uploaded. We will see that BitTorrent accomplishes these goals by using algorithms called choke algorithm and rarest first piece selection algorithm. Before the details of these algorithms are discussed, the architecture and the protocol of BitTorrent will be discussed next.

BitTorrent Architecture

BitTorrent is a hybrid network using both the client-server architecture and the peer-to-peer architecture. The centralized server is called the tracker. The tracker's responsibility is to help peers find other peers. A tracker consists of many torrent sessions with each session it keeps track of all of the peers participating in the particular torrent. The peer contacts the tracker and the tracker responds with a list of peers it may connect to. The tracker is not responsible for the actual distribution of the content at all. The bandwidth of the tracker is very low because it is a simple protocol, which peers only connect to when they start up and at defined time intervals of usually 30 minutes [3]. The peer knows the URL of the tracker because it is defined in the torrent file.

The torrent file is a static 'metainfo' file that represents a session of content being distributed. The torrent file is created with the URL of the tracker and the actual file or files to be part of this torrent. The format of the torrent file is bencoding. Bencoding consists of nested dictionaries and lists. These dictionaries and lists can contain strings and integers [4]. The torrent file is a bencoded dictionary containing two keys, *announce* and *info*. The *announce* key is the URL of the tracker [4]. The info key maps to a dictionary described below.

The info key is a dictionary with the following keys: *name, piece length, pieces*, and either *length* or *files* key. The *name* key is a string that is the suggested name to download the file as. It is suggested meaning that the downloader peer can choose the name that they desire. *Piece length* maps to the number of bytes each piece the file is split into. The file is split into equal size pieces except possibly the last piece. The *pieces* key maps to a string of the SHA1 hash of each piece. Each SHA1 hash is a string of 20 characters long so for example the hash value of piece 4 would be the substring of *pieces* at character 60 to character 79 assuming that the string begins at index 0. This is used so the downloader can verify the data by checking what they download with these hash values [3]. The next key can either be *length* or *files* depending if this torrent is a single file or a directory of files [4].

For a single file the key is *length* and it is simply the length of the file in bytes. If the torrent represents a directory the key *files* is used. This key maps to a *files list* and includes a list of dictionaries containing the following keys, *length* and *path*. The *length* is number of bytes of the file and *path* is a list of strings that correspond to subdirectory names. This is it for the contents of the torrent file [4]. The parts that are the most important thing to remember about this file is that it contains the tracker URL and splits the file or files into equal size pieces and includes the SHA1 hash values of all the pieces of the file.

Now that a peer has a torrent file and connects to the tracker for a list of peers it may join the peer network. The peer must first allocate space for the file or files, which it gets from the torrent file. This is necessary because the peer will not download pieces of a file in order so the BitTorrent application will need to assemble these pieces in order while it receives them. The peer network is a peer-to-peer network where the leechers act as clients and servers and the seeders act just as servers. The peers distribute the file to each other by using the swarming technique [1]. Peers download pieces from multiple peers at the same time. It also uploads to either the same or different peers, pieces of the file it has already downloaded.

The Tracker Protocol

The tracker plays an essential part of BitTorrent because without it, there would be no way for peers to find other peers to download from. Trackers use a simple protocol layered on top of HTTP [3]. The tracker receives HTTP GET requests and it sends bencoded messages to the peer's request. The tracker GET requests contain the following keys; *info_hash*, *peer_id*, *ip*, *port*, *uploaded*, *downloaded*, *left*, and *event*. The *info_hash* key is how the tracker determines which torrent session the client is a part of or is joining. This key is the 20 byte SHA1 hash of the *info* key from the torrent file. The *peer_id* is the id the client randomly generated at the start of the download. This is a string of 20 characters long. The next key, *ip* is optional, which is generally used for the origin if it is on the same computer as the tracker. The *port* key is the port number the peer is listening. The key *uploaded*, *downloaded*, and *left* correspond to the total amount uploaded so far, total amount downloaded so far, and number of bytes left to download respectively. These are used so the tracker can keep track of statistics. For instance it may now how many seeders and leechers is in torrent session or how many times the particular content has been downloaded. The next key *event* is optional which can have the possible values of started, completed, stopped, or empty. Started is used when the download just begins. Completed is used when the downloader finished downloading. Stopped is used when the peer stops downloading. The string empty is used during the announcements the peer makes at regular intervals [4].

The responses the tracker sends to the peers are bencoded dictionaries. This dictionary must contain two keys, *interval* and *peers*. The *interval* key is the number of seconds the peer should wait between regular requests. The *peers* key maps to a list of dictionaries that represents a list of peers which contains the keys, *peer id*, *ip*, and *port*. The *peer id* is the peer id the peer sent to the tracker in tracker request. The *ip* and *port* is the IP address and port number of the peer [4]. Typically the tracker returns 50 random peers in this response [1]. Notice how it can keep track of statistics by recording all of the information it receives from the peer requests. The bandwidth of the tracker is very low since peers only connect to the tracker for a very short time in long time intervals (usually 30 minutes). The total amount of bandwidth used by the tracker is currently around a thousandth the total amount of bandwidth used [3].

The BitTorrent Peer Protocol

Peers communicate with each other by sending messages directly to each other using the BitTorrent Peer Protocol. This protocol operates over TCP. In order for two peers to send messages to each other, they must first connect to each other by sending a handshake message. This handshake message starts with the string, "19 BitTorrent Protocol". The 19 is the length prefix. After this string, there are eight reserved bytes, which currently are not used, but these are added to allow the protocol to be extendable. The next 20 bytes is the SHA1 hash value of the *info* value of the torrent file. This is same value that is used in tracker requests. This value is needed because a peer may be participating in many torrent swarms and when a peer sends a handshake message, it needs to know what specific swarm it is joining. The next 20 bytes is the 20-byte peer id, which is the same value that is sent to the tracker in the requests [4]. This completes the connection and now the peers may start sending other types of messages to this and all of their connected peers in their peer set.

For all of the connections a peer has it must also maintain specific information about them. All connections are either in the choked or the unchoked state. If a peer is choked then it is not allowed to download data in this connection. All connections must also be in the interested or not state. A peer is interested in another peer, if that peer has pieces of the content it does not have. Only when a peer is interested in another peer and is unchoked it may download from the connection. Since a peer must send messages to state that they are interested, all peers must also need to know what pieces of the content the peer has already downloaded. All connections start

off choked and not interested [4]. Besides the handshake message, there are 9 other messages that can be sent to other peers.

These messages are distinguished from each other by the first byte which specifies what type of message it is. These messages are [4]:

0. choke
1. unchoke
2. interested
3. not interested
4. have
5. bitfield
6. request
7. piece
8. cancel

The protocol overhead for both downloading and uploading is very low. Legout has determined that the overhead is less than 2% in his experiments [1]. These messages will be explained in an example of Peer A joining a torrent session and describing what messages he sends and receives as part of the swarm.

First Peer A sends a request to the tracker to receive a list of peers to connect to. Peer A than proceeds to initiate BitTorrent connections with a subset of this list by sending *handshake* messages. Usually no more than 40 connections are initiated [1]. After Peer A connects to another peer, it expects a *bitfield* message. Usually this message is sent only if the peer has already downloaded at least one piece. This message is a bitfield, which corresponds to the pieces of the file the peer has already downloaded. This is needed because all peers in the swarm must now what pieces all other peers have. Peer A must maintain its active peer set. This set consists of the peers that are unchoked and usually contain only four peers [1]. This set must change constantly in order to maximize upload speed. This is done by using the choke algorithm, which will be discussed later but it cannot be called until Peer A has at least one complete piece.

After Peer A has connections with other peers and it knows what pieces they have, it can now send interested messages to these peers that have pieces it wants. This message has no extra data because it does not need to specify anything specific. When a peer receives this message from Peer A, they update the state of this peer to interested. Eventually, a peer will unchoke Peer A by sending an unchoke message to Peer A. When Peer A receives these messages, it can now proceed by sending *request* messages to the peer that has unchoked them. *Request* messages contain three parameters; index, begin and length. Index refers to which piece of the file and begin and length are used to specify which part of this piece they want called blocks. Peer A will receive piece messages from the peers it send request messages, which contain the actual data. When Peer A downloads a complete piece (not a block) it checks the integrity of this piece by computing the SHA1 hash of this piece and checking this value against the one in the torrent file. If the values are the same, it sends out *have* messages to all peers Peer A is connected to specifying what piece it has just downloaded. Also Peer A will receive *have* messages from other peers when they have completed downloading a piece. Peer A must maintain which pieces all peers have and send out *interested* and *not interested* messages to all peers whenever the state changes by either they receive a *have* message or they have downloaded a complete piece. Peer A cannot upload anything to any peers until it has completed its first piece [4]. Peer A will decide which peers to upload to by using the choke algorithm.

The Choke Algorithm

The choke algorithm is used to change the active peer set, which are the peers that a peer uploads to. This makes BitTorrent fair because peers upload to other peers who give them the fastest download speeds. To be fair, peers should not be allowed to download much more than they have uploaded. Since peers upload to peers that let them download, this algorithm favors

peers that upload than those that don't [1]. This algorithm tries to maximize upload utilization for example if two peers are getting poor download rates, they can start uploading to each other and both peers will get a better download rate than before [3]. The way a peer finds out if it can get better transfer rates is by trying out unutilized connections on a trial basis [3].

The choke algorithm makes a distinction between the seeder and leecher state. When in the leecher state, a peer must maintain the current download rate for its peer connections. This algorithm is called every 10 seconds, or whenever an unchoked peer becomes interested or uninterested [1]. The time this algorithm is called is important because resources are wasted if they rapidly choke and unchoke peers. Ten seconds allow the TCP connection to get to their full capacity [3]. This algorithm orders peers that are interested and have sent a block via a piece message in the last 30 seconds. These peers are sorted by their download rate. If a peer has not sent a block in the last 30 seconds, the peer is considered snubbed. Snubbed peers are left out of this algorithm in ordered to guarantee that active peers are unchoked [1]. The three fastest peers are unchoked. Every three rounds, which is 30 seconds, one peer is chosen randomly that is choked and interested is unchoked. This peer is called the planned optimistic unchoked peer. If this peer is part of the three fastest peers another peer is chosen at random and than unchoked [1]. If a peer is snubbed and interested, it can be selected as the optimistic unchoke. The optimistic unchoking is necessary because without it peers would have no method of discovering better connections than the ones they currently have [3]. Also without this optimistic unchoking, new peers who have no pieces would never be able to receive their first piece [1]. The choke algorithm is different in the seed state.

In previous versions of BitTorrent the choke algorithm was the same in the leecher state as the seed state, except the seed state uses the upload speed to determine which peers to unchoke. The reason why this older version was changed because it favors peers with a high download rate. This may allow a peer to dominate all the resources of the seed if it has the highest download rate. This can allow a peer that does not upload anything (free rider) to get a high download rate. This will make the protocol unfair because it is possible for a peer to download without ever contributing anything back [1]. The current choke algorithm is called every 10 seconds and only peers that are unchoked and interested are used to determine what happens. The algorithm orders the peers by the time they were last unchoked for all the peers were unchoked recently. The upload rate is then used to decide between peers with the same last unchoked time. Because of this step, peers are not ordered according to their upload rate, but rather than their time of their last unchoke. This is more beneficial because the active peer set changes more frequently [1]. Also every two out of ever three rounds, the first three peers are unchoked and another interested peer at random. The third round, the first four peers are unchoked [1].

Bharambe's research in analyzing and improving the performance of BitTorrent has found that this current algorithm does not prevent unfairness. Especially in heterogeneous settings where high bandwidth peers connect to low bandwidth peers. He came up with a solution to this problem by incorporating a bandwidth matching tracker [2]. Although this algorithm is not perfect, it does an efficient job at maximizing the utilization and does a better job than other P2P file sharing applications at trying to prevent unfairness.

Piece Selection Algorithm

BitTorrent uses the rarest first algorithm to determine which piece it should currently download. The goal of this algorithm is to maximize the diversity of pieces available, which makes the number of replicas of each piece as equal as possible. This makes it more unlikely that peers will have trouble downloading pieces because of "rare" blocks that are difficult to find [2]. Also by using this algorithm it is more likely that a peer will always have something to offer to other peers [3].

Since each peer maintains what pieces all of their peers in their peer set has, it can then determine how many copies of the pieces are available to download. It uses this information to determine which piece to download first by keeping track of a rarest piece set. This set is updated every time it receives a *have* message or whenever a peer leaves the peer set [1].

When a peer has just joined a torrent and has currently no pieces it uses the random first policy. This policy is used until the peer has downloaded 3 complete pieces and then it switches to the rarest first [1]. The purpose of the random first policy is that a random piece is more likely to be more duplicated than rare pieces so its download time will be faster. It is important for the peer to download its first piece, because it cannot perform the choke algorithm unless it has a piece that another peer is interested in [1].

Also whenever a peer selects a piece to download it uses a strict priority policy, which is at the block level. Whenever a peer has downloaded a block of a piece, it will then choose to download other blocks of the same piece [1]. This will allow the peer to download complete pieces instead of having a bunch of blocks of rare pieces without having complete pieces. It is important to have complete pieces because a peer cannot upload blocks of a piece until it has downloaded the complete piece.

Also when a leecher is near completion of the download, it enters the end game mode [3]. This mode starts once the peer has requested all of the blocks. In this mode a peer requests all the blocks it has not received to all connected peers. Each time it receives a block, it sends out a cancel message to all other connected peers [1]. These cancel messages are needed to make sure that not too much bandwidth is wasted on redundant piece messages. It has been proven that this period does not waste too much bandwidth because this period is very short and the end of a file is always downloaded quickly [1] [3].

Conclusion

BitTorrent is a new successful protocol used to distribute files. It does not have any search capability like other P2P networks, which allows BitTorrent to focus on only the content delivery. It uses a hybrid architecture with the file distribution only occurring in the peer-to-peer architecture. There needs to be a centralized server called the tracker, which is used for peers to find other peers. The protocol overhead is very low and BitTorrent does a good job at maximizing the upload utilization and does a decent job at trying to prevent unfairness. We have seen that BitTorrent is successful from the use of the choke algorithm and the rarest first piece selection. A lot of studies have been done on improving BitTorrent and since its still a fairly new protocol the algorithms are not perfect and are still being improved [2].

References

[1] Legout, Arnaud, "Understanding BitTorrent: An Experimental Perspective," in INRIA-00000156, Version 3 – 9 November 2005. http://hal.inria.fr/docs/00/04/31/40/PDF/bt_experiments_techRepINRIA-00000156_VERSION3_9NOVEMBER2005.pdf

[2] Bharambe, Ashwin R. "Analyzing and Improving BitTorrent Performance," in MSR-TR-2005-03.http://hal.inria.fr/docs/00/04/31/40/PDF/bt_experiments_techRepINRIA-00000156_VERSION3_9NOVEMBER2005.pdf

[3] Cohen, Bram. "Incentives Build Robustness in BitTorrent. 2003. http://www.bittorrent.com/bittorrentecon.pdf

[4] BitTorrent Protocol. http://www.bittorrent.org/protocol.html

[5] Green, Heather. "BitTorrent's Grab at Respectability." *BuisnessWeek Online*. 27 September 2005.http://www.businessweek.com/technology/content/sep2005/tc20050927_3006_tc024.htm?campaign_id=rss_topStories