Incentives Build Robustness in BitTorrent

Bram Cohen
bram@bitconjurer.org

May 22, 2003

Abstract

The BitTorrent file distribution system uses tit-for-tat as a method of seeking pareto efficiency. It achieves a higher level of robustness and resource utilization than any currently known cooperative technique. We explain what BitTorrent does, and how economic methods are used to achieve that goal.

1 What BitTorrent Does

When a file is made available using HTTP, all upload cost is placed on the hosting machine. With BitTorrent, when multiple people are downloading the same file at the same time, they upload pieces of the file to each other. This redistributes the cost of upload to downloaders, (where it is often not even metered), thus making hosting a file with a potentially unlimited number of downloaders affordable.

Researchers have attempted to find practical techniqes to do this before[3]. It has not been previously deployed on a large scale because the logistical and robustness problems are quite difficult. Simply figuring out which peers have what parts of the file and where they should be sent is difficult to do without incurring a huge overhead. In addition, real deployments experience very high churn rates. Peers rarely connect for more than a few hours, and frequently for only a few minutes [4]. Finally, there is a general problem of fairness [1]. The total download rate across all downloaders must, of mathematical necessity, be equal to the total upload rate. The strategy for allocating upload which seems most likely to make peers happy with their download rates is to make each peer's download rate be proportional to their upload rate. In practice it's very difficult to keep peer download rates from sometimes dropping to zero by chance, much less make upload and download rates be correlated. We will explain how BitTorrent solves all of these problems well.

1.1 BitTorrent Interface

BitTorrent's interface is almost the simplest possible. Users launch it by clicking on a hyperlink to the file they wish to download, and are given a standard "Save As" dialog, followed by a download progress dialog which is mostly notable for having an upload rate in addition to a download rate. This extreme case of use has contributed greatly to BitTorrent's adoption, and may even be more important than, although it certainly complements, the performance and cost redistribution features which are described in this paper.

1.2 Deployment

The decision to use BitTorrent is made by the publisher of a file. Downloaders use BitTorrent because it's the only way to get the file they want. Frequently downloaders cease uploading as soon as their download completes, although it is considered polite to leave the client uploading for a while after your download completes. The standard implementation continues to upload until the window is closed, which frequently results in uploads continuing until the user gets back to their machine.

In a typical deployment, the number of incomplete downloaders, (meaning ones which do not have the

whole file), increases very rapidly after the file is made available. It eventually peaks and then falls off at a roughly exponential rate. The number of complete downloaders increases slowly, peaks some time after the number of incomplete downloaders does, then also falls off exponentially. The peak of incomplete downloaders passes as downloaders complete. The peak of incomplete downloaders passes as finished downloaders stop uploading. The exponential falloff of both reflects the rate of new downloaders joining after the initial rush is over.

file made available by host, downloaders use BT because they want the file.downloaders upload while downloading, then leave.

2 Technical Framework

2.1 Publishing Content

To start a BitTorrent deployment, a static file with the extension .torrent is put on an ordinary web server. The .torrent contains information about the file, its length, name, and hashing information, and the url of a tracker. Trackers are responsible for helping downloaders find each other. They speak a very simple protocol layered on top of HTTP in which a downloader sends information about what file it's downloading, what port it's listening on, and similar information, and the tracker responds with a list of contact information for peers which are downloading the same file. Downloaders then use this information to connect to each other. To make a file available, a 'downloader' which happens to have the complete file already, known as a seed, must be started. The bandwidth requirements of the tracker and web server are very low, while the seed must send out at least one complete copy of the original file.

2.2 Peer Distribution

All logistical problems of file downloading are handled in the interactions between peers. Some information about upload and download rates is sent to the tracker, but that's just for statistics gathering. The tracker's responsibilities are strictly limited to helping peers find each other.

Although trackers are the only way for peers to find each other, and the only point of coordination at all, the standard tracker algorithm is to return a random list of peers. Random graphs have very good robustness properties [2]. Many peer selection algorithms result in a power law graph, which can get segmented after only a small amount of churn. Note that all connections between peers can transfer in both directions.

In order to keep track of which peers have what, BitTorrent cuts files into pieces of fixed size, typically a quarter megabyte. Each downloader reports to all of its peers what pieces it has. To verify data integrity, the SHA1 hashes of all the pieces are included in the .torrent file, and peers don't report that they have a piece until they've checked the hash. Erasure codes have been suggested as a technique which might help with file distribution [3], but this much simpler approach has proven to be workable.

Peers continuously download pieces from all peers which they can. They of course cannot download from peers they aren't connected to, and sometimes peers don't have any pieces they want or won't currently let them download. Strategies for peers disallowing downloading from them, known as choking, will be discussed later. Other suggested approaches to file distribution have generally involved a tree structure, which doesn't utilize the upload capacity of leaves. Simply having peers announce what they have results in less than a tenth of a percent bandwidth overhead and reliably utilizes all available upload capacity.

2.3 Pipelining

When transferring data over TCP, like BitTorrent does, it is very important to always have several requests pending at once, to avoid a delay between pieces being sent, which is disastrous for transfer rates. BitTorrent facilitates this by breaking pieces further into sub-pieces over the wire, typically sixteen kilobytes in size, and always keeping some number, typically five, requests pipelined at once. Every time a sub-piece arrives a new request is sent. The amount of data to pipeline has been selected as a value which

can reliably saturate most connections.

2.4 Piece Selection

Selecting pieces to download in a good order is very important for good performance. A poor piece selection algorithm can result in having all the pieces which are currently on offer or, on the flip side, not having any pieces to upload to peers you wish to.

2.4.1 Strict Priority

BitTorrent's first policy for piece selection is that once a single sub-piece has been requested, the remaining sub-pieces from that particular piece are requested before sub-pieces from any other piece. This does a good job of getting complete pieces as quickly as possible.

2.4.2 Rarest First

When selecting which piece to start downloading next, peers generally download pieces which the fewest of their own peers have first, a technique we refer to as 'rarest first'. This technique does a good job of making sure that peers have pieces which all of their peers want, so uploading can be done when wanted. It also makes sure that pieces which are more common are left for later, so the likelihood that a peer which currently is offering upload will later not have anything of interest is reduced.

Information theory dictates that no downloaders can complete until every part of the file has been uploaded by the seed. For deployments with a single seed whose upload capacity is considerably less than that of many downloaders, performance is much better if different downloaders get different pieces from the seed, since redundant downloads waste the opportunity for the seed to get more information out. Rarest first does a good job of only downloading new pieces from the seed, since downloaders will be able to see that their other peers have pieces the seed has uploaded already.

For some deployments the original seed is eventually taken down for cost reasons, leaving only current downloaders to upload. This leads to a very significant risk of a particular piece no longer being available from any current downloaders. Rarest first again handles this well, by replicating the rarest pieces as quickly as possible thus reducing the risk of them getting completely lost as current peers stop uploading.

2.4.3 Random First Piece

An exception to rarest first is when downloading starts. At that time, the peer has nothing to upload, so it's important to get a complete piece as quickly as possible. Rare pieces are generally only present on one peer, so they would be downloaded slower than pieces which are present on multiple peers for which it's possible to download sub-pieces from different places. For this reason, pieces to download are selected at random until the first complete piece is assembled, and then the strategy changes to rarest first.

2.4.4 Endgame Mode

Sometimes a piece will be requested from a peer with very slow transfer rates. This isn't a problem in the middle of a download, but could potentially delay a download's finish. To keep that from happening, once all sub-pieces which a peer doesn't have are actively being requested it sends requests for all sub-pieces to all peers. Cancels are sent for sub-pieces which arrive to keep too much bandwidth from being wasted on redundant sends. In practice not much bandwidth is wasted this way, since the endgame period is very short, and the end of a file is always downloaded quickly.

3 Choking Algorithms

BitTorrent does no central resource allocation. Each peer is responsible for attempting to maximize its own download rate. Peers do this by downloading from whoever they can and deciding which peers to upload to via a variant of tit-for-tat. To cooperate, peers upload, and to not cooperate they 'choke' peers. Choking is a temporary refusal to upload; It stops uploading, but downloading can still happen

and the connection doesn't need to be renegotiated when choking stops.

The choking algorithm isn't technically part of the BitTorrent wire protocol, but is necessary for good performance. A good choking algorithm should utilize all available resources, provide reasonably consistent download rates for everyone, and be somewhat resistant to peers only downloading and not uploading.

3.1 Pareto Efficiency

Well known economic theories show that systems which are pareto efficient, meaning that no two counterparties can make an exchange and both be happier, tend to have all of the above properties. In computer science terms, seeking pareto efficiency is a local optimization algorithm in which pairs of counterparties see if they can improve their lot together, and such algorithms tend to lead to global optima. Specifically, if two peers are both getting poor reciprocation for some of the upload they are providing, they can often start uploading to each other instead and both get a better download rate than they had before.

BitTorrent's choking algorithms attempt to achieve pareto efficiency using a more fleshed out version of tit-for-tat than that used to play prisoner's dilemma. Peers reciprocate uploading to peers which upload to them, with the goal of at any time of having several connections which are actively transferring in both directions. Unutilized connections are also uploaded to on a trial basis to see if better transfer rates could be found using them.

3.2 BitTorrent's Choking Algorithm

On a technical level, each BitTorrent peer always unchokes a fixed number of other peers (default is four), so the issue becomes which peers to unchoke. This approach allows TCP's built-in congestion control to reliably saturate upload capacity.

Decisions as to which peers to unchoke are based strictly on current download rate. Calculating current download rate meaningfully is a surprisingly difficult problem; The current implementation essentially uses a rolling 20-second average. Former choking algorithms used information about long-term net transfer amounts, but that performed poorly because the value of bandwidth shifts rapidly over time as resources go away and become available.

To avoid situations in which resources are wasted by rapidly choking and unchoking peers, BitTorrent peers recalculate who they want to choke once every ten seconds, and then leave the situation as is until the next ten second period is up. Ten seconds is a long enough period of time for TCP to ramp up new transfers to their full capacity.

3.3 Optimistic Unchoking

Simply uploading to the peers which provide the best download rate would suffer from having no method of discovering if currently unused connections are better than the ones being used. To fix this, at all times a BitTorrent peer has a single 'optimistic unchoke', which is unchoked regardless of the current download rate from it. Which peer is the optimistic unchoke is rotated every third rechoke period (30 seconds). 30 seconds is enough time for the upload to get to full capacity, the download to reciprocate, and the download to get to full capacity. The analogy with tit-for-tat here is quite remarkable; Optimistic unchokes correspond very strongly to always cooperating on the first move in prisoner's dilemma.

3.4 Anti-snubbing

Occasionally a BitTorrent peer will be choked by all peers which it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers. To mitigate this problem, when over a minute goes by without getting a single piece from a particular peer, BitTorrent assumes it is 'snubbed' by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

3.5 Upload Only

Once a peer is done downloading, it no longer has useful download rates to decide which peers to upload to. The current implementation then switches to preferring peers which it has better upload rates to, which does a decent job of utilizing all available upload capacity and preferring peers which noone else happens to be uploading to at the moment.

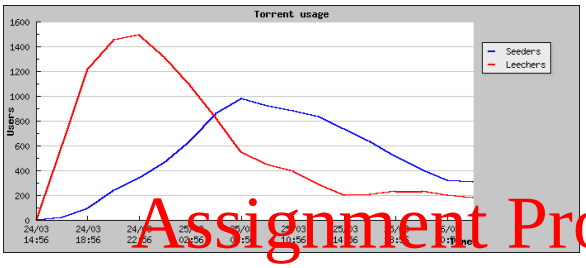

Figure 1: The number of complete downloaders ('seeders') and incomplete downloaders ('leechers') of a large deployment of an over 400 megabyte file over time. There must have been at least 1000 successful downloads, since at one time there were that many complete downloaders. The actual number of downloads completed during this period was probably several times that.

4 Real World Experience

BitTorrent not only is already implemented, but is already widely deployed. It routinely serves files hundreds of megabytes in size to hundreds of concurrent downloaders. The largest known deployments have had over a thousand simultaneous downloaders. The current scaling bottleneck (which hasn't actually been reached) appears to be the bandwidth overhead of the tracker. Currently that's about a thousandth the total amount of bandwidth used, and some minor protocol extensions will probably get it down to a ten thousandth.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in cooperative environments. In *Proceedings of IPTPS03*, Berkeley, USA, Feb. 2003.

[4] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02*, Cambridge, USA, Mar. 2002.