Improving the Performance of Peer to Peer File Sharing Via BitTorrent Protocol.

S.Venkateswarlu [1] and Md khaja zikriya [2]

DEPT of CSE,KL University, Vaddeswaram ,Vijayawada, Andhra Pradesh, India.

Abstract - Even though many P2P file-sharing systems have been proposed and implemented, only very few have stood the test of intensive daily use by a very large user community. If the community of file-sharers cooperates appropriately, very high download rates can be achieved at virtually no cost to the injector of the content. We begin this with a brief history of P2P file-sharing, including Gnutella, and explain why so many file-sharing networks suffer from free riding. We then focus primarily on BitTorrent, the most successful file-sharing network to date, with more than 150 Millions of users worldwide. We describe the BitTorrent protocol in detail, and present some game-theoretic analyses. Finally, we describe multiple attacks on BitTorrent that increase an individual user's performance, and show how the BitTorrent protocol could be improved to increase overall performance.

Keywords-Peer to peer ,BitTorrent protocol, free-riding, BitTyrant.

Corresponding Author: S.Venkateswarlu

I. INTRODUCTION

Distribution of File via P2P Networks

Some of the most popular Internet-based services are in the online media domain. We can buy MP3s from the iTunes store, we can download ebooks from Amazon, and we can watch videos for free on Youtube and Hulu, or for a fee on Netix. All of these services are based on the “client-server" model, which means that the required tasks are clearly separated: the provider of the service, or more generally of a resource, is the server; the requester of the service, or the resource, is the client. Generally, the client is a standard home computer that does not share any of its resources, and only contacts the server whenever it wants to consume its service. Because millions of MP3s are downloaded and videos are watched every day, the server" is often a whole network of very powerful machines. In the case of Youtube, Hulu, Netix, etc., the number of servers easily exceeds multiple thousands. Obviously, these servers are very expensive and not many companies can afford to use such large-scale server networks for the distribution of media.

In the late 1990's, a very different information sharing paradigm has emerged: peer-to-peer (P2P) file-sharing networks. In a P2P network, the separation of servers and clients is removed: each computer can act as both a server and as a client, often simultaneously. Users of standard home computers around the world may connect to each other and form a P2P network. With the appropriate protocol in place, users can exchange (by uploading and downloading) files with each other. All of this generally takes place without any centralized system infrastructure or control. This decentralization leads to three advantages that P2P systems have over server-based systems:

1.Users enjoy a higher level of anonymity and thus privacy.
2.Large files can be distributed much more effciently (i.e., with much lower costs).
3.P2P systems are more resilient to various forms of failures.

P2P file-sharing networks have often received a bad reputation because a majority of files exchanged in these networks are copyrighted materials, and thus, exchanging them with other users is illegal in most countries. However, there are also many examples of legal uses of P2P file-sharing networks. For example, the newest Linux distributions are generally available in P2P file-sharing networks. Another example is the the game "World of Warcraft," which is distributed via the BitTorrent P2P file-sharing network instead of letting users download it from a centralized server. Finally, some Internet TV services like Joost stream their video data to the users via P2P networks.

II. RELATED WORK

Gnutella Network

The basic design and operation of Gnutella is very similar to other P2P file-sharing networks. A user who wants to participate downloads an application (or develops one himself) that adheres to the Gnutella protocol. Depending on the role that a user plays in a particular transaction, we will be speaking about servers and clients, but more generally we will just use the term peers to refer to the computers running the Gnutella application. To join the network, a new peer must first connect to one of several known peers that are almost always online (but that generally do not share any files). These peers then forward a list of IP and port addresses of other Gnutella peers to the new peer. Once connected to the network, a peer can either initiate an interaction by sending a broadcast message, or it can remain passive and simply rebroadcast messages from others. To find a desired file, a peer sends a Query Message describing the content it is looking for. These messages are forwarded (i.e., rebroadcast) from peer-to-peer until a peer with the desired content is found, or until some maximum number of re-broadcasts has occurred. A peer that has the desired file replies to a query message with a Query Response Message, containing that peer's IP and port address, a unique client ID, as well as other information necessary to download the file. These query response messages are propagated backwards along the path that the original query message took, until it reaches the original requester who can then contact the peer who has the file and download it.

Free Riding on Gnutella

A central feature of Gnutella and other similar networks is that there is no central server, users are not monitored, and no statistics about user behavior are maintained. Thus, for two peers interacting with each other, every interaction looks just like any other one: it's a one-shot game with anonymous players. This is in contrast to the way some users had shared files before the existence of P2P file sharing networks. For example, on Bulletin boards, users often knew each other from prior interactions and there was a strong sense of a close-nit community.

Unfortunately, the design features of Gnutella lead to an incentive problem inherent to many of the early file-sharing networks (including Kazaa, eDonkey, etc.): the majority of users were "free riding":Definition 3.1 (Free Riding). An agent is free riding when he consumes resources without contributing back to the community. Consider the simplified "P2P File Sharing Game" as shown in Figure 1. Each player can either cooperate, i.e., share files with other users in the network, or free ride, i.e., only download files without ever uploading any files. The numbers in the payoff table are just illustrative, but are meant to convey the fact that peers obtain positive utility from downloading a file (here 3), but that they have a small cost for uploading a file (here -1). User may incur a cost for uploading for a variety of reasons: they might have to pay for bandwidth, they might need their upload bandwidth for other applications (e.g., voice-over-IP), they might have a disutility from leaving their

computer on when files are being uploaded, or they might be concerned about the legal implications from uploading copyrighted material.

		Player 2	
		Share	Free Ride
Player 1	Share	2, 2	−1, 3
	Free Ride	3, −1	0, 0

Figure 1: The P2P File-Sharing Game.

No matter what the exact benefit-to-cost ratio is for a user, as long as a user obtains positive value from receiving a file, and incurs a cost for providing a file, the resulting game has the same structure as the one displayed in Figure 1: it is a game. Given the incentive structure of Gnutella, one might be surprised that the network hasn't collapsed very quickly. Some of the sharing activity in P2P file-sharing networks can be explained by users who leave the default settings of a software client in place, and many clients were configured such that downloaded files would automatically be stored in a folder that was also available for upload afterwards. Other users might simply enjoy sharing files with other people so much, that it outweighs their costs for uploading.
In Kazaa, for example, a client would collect local statistics about how much it had uploaded and downloaded, with this information transmitted to other peers, the idea being, that peers would then give preference to those peers with a better upload to download ratio. However, because this information was stored locally on the clients, it could easily be spoofed, and very quickly Kazaa clients were available where a user could simply enter a number that would be transmitted to the other clients instead of the true number of uploaded kilobytes. Thus, to this day, free riding remains a big problem in Gnutella, leading to a slow but steady decline in the network's performance and market share.

Repeated Games in Gnutella?
So far we have described the P2P file-sharing game" as a one-shot game. However, in reality, most peers download many files, possibly thousands, over their lifetime. Thus, it might be more appropriate to model the P2P file-sharing as an infinitely repeated game. We saw that for the infinitely repeated game, the tit-for-tat strategy is a Nash equilibrium, and that in the equilibrium, both players always cooperate. Given this positive result, we might hope that suitably-designed file-sharing clients could also implement strategies (like tit-for-tat) that leads to cooperative equilibria, and thereby provide agents with an incentive to share. Unfortunately, the problem with the design of early P2P file-sharing clients was that individual peers often only interacted once in their life time, i.e., the rendezvous probability was extremely low. And even if two peers i and j saw each other multiple times, it was unlikely that peer i would have a file that peer j was interested in obtaining right now. This problem was neatly solved by Bram Cohen in 2003, when he introduced a new file-sharing protocol that was based on a completely new paradigm, called BitTorrent.

III. BitTorrent

At a high level, one of the key differences between BitTorrent and Gnutella is that in BitTorrent, a client that is currently downloading a file, simultaneously uploads pieces of that same file to other peers who are also currently interested in obtaining the same file. On a " piece level," the peers that are concurrently downloading the same file are playing a repeated game with each other, and BitTorrent has implemented some version of a tit-for-tat strategy to encourage cooperation in that game. The protocol is designed such that the more a peer

uploads to other peers, the faster it will be able to download the same file. Thus, BitTorrent was designed with the goal to provide users with a clear incentive to upload to others, and to discourage free riding. In this section, we describe the BitTorrent protocol in detail, and in the following sections we will analyze its game-theoretic properties to see what incentives are actually at play.

BitTorrent Protocol

To participate in BitTorrent, a user must download (or develop) a software application that is compatible with the BitTorrent protocol. There exists a reference client that closely follows the original protocol as specified in 2003, and we will use that when describing the default client. How ever, many other clients have been developed and released to the public that implement strategies deviating from those of the reference client. In 2011, some of the most popular BitTorrent clients included µTorrent and Vuze.

The content that users download via BitTorrent can either be a single file, or a large aggregated collection of files (e.g., all MP3s from a new music album), but for simplicity we will always speak of "a file", going forward. The peers that are currently involved in uploading or downloading the same file are the swarm. A file in BitTorrent is divided into pieces, which are further divided into blocks (typically 64-512KB per block). To find content, a user generally goes to a website that maintains a searchable directory of torrents or torrent files. One of the most popular such websites is http://thepiratebay.org/, in particular because it provides access to some of the most popular, yet copyrighted movies currently available.The torrent files are often not hosted by the same website that provides the directory service, but only linked to.

The user downloads a .torrent file and opens it with his BitTorrent client. The .torrent file contains all the relevant metadata, including the name and size of the file, as well as a SHA-1 fingerprint of the data blocks.3 Note that the term ".torrent file" always refers to the file containing the metadata information, but that the term "torrent" is used ambiguously. Sometimes, it also refers to the .torrent file, and sometimes it refers to the file (or files) that the user is ultimately interested in downloading.

One key element of the metadata contained in the .torrent file is the address of a tracker, which is a computer that acts as a server for this particular torrent, responsible for coordinating the peers in the swarm. The client announces itself to the tracker to obtain a list of peers that are part of the swarm, and the tracker returns a random subset, typically containing 50 active peers of the swarm.

The peer re-announces itself to the tracker periodically, usually every 15 to 30 minutes, and also tells the tracker when it is leaving the swarm (i.e., when it becomes inactive). Upon receiving the list of peers from the tracker, a peer can connect to the other peers and start downloading pieces from them.

The peers who are currently downloading pieces of a file are called leechers, and the peers who already have the complete file and are still uploading the file are called seeders. Thus, the injection of new content into BitTorrent always begins with one seeder uploading the file to other peers. A peer is only interested in peers that have pieces that it does not currently have itself.

A peer a can either initiate a connection to another peer b from the list it received from the tracker, or it can receive a connection request from a peer b that has a in the list of peers that b received from the tracker. Either way, when two peers in a swarm connect for the first time, they become neighbors, and they exchange a bit field, i.e., a bit array informing each other about which pieces they already have. A peer maintains open connections to all of its neighbors, forming the local neighborhood of that peer, and only closes those connections when itself or the other peer leaves the swarm. Whenever a peer finishes downloading a new

piece, it sends per-piece have messages to the other peers in its local neighborhood, who can then update their information. Whenever a peer finishes downloading a new piece, it sends per-piece have messages to the other peers in its neighborhood, who can then update their information. Locally, each peer maintains an estimate of the availability of each piece of the file by counting how many of its neighbors have the piece. When a peer p starts uploading to a leecher l, then l informs p about which blocks l wishes to receive next. The standard strategy is rarest-first, where a leecher tries to download blocks from those pieces that currently have the lowest availability in the swarm.

A peer generally only uploads to a small subset of peers in its neighborhood. All peers that it doesn't upload to are called choked, and the few that receive some data are called unchoked, also sometimes called the active set. The number of unchoke slots k is variable, and can be set by the client software. The reference client suggests to set the number of upload slots proportional to p upload capacity, however, most implementations use a fixed number (often k = 4). A key aspect of the BitTorrent protocol is determining which peers to unchoke. We will soon see that this aspect is of particular strategic importance in BitTorrent.

The Unchoking Algorithm: Who to upload to?

A peer always tries to download from all of the peers in its neighborhood, but is very selective regarding who to upload to, i.e., which client to unchoke. In the reference client, a peer's decision in regard to who to unchoke is based on how much the other peers have uploaded to him in the past. To make that decision, each peer considers epochs of 10 seconds each, and the download rate for each neighbor is based on a rolling 20-second average. A peer unchokes the k peers with the highest upload rate, splitting its upload bandwidth equally among these k peers at the so-called equal-split rate. The client makes a new unchoking decision every 10 seconds. The reference client reserves one upload slot as an optimistic unchoking slot. Every 30 seconds, the client unchokes one peer at random from its neighborhood. This serves two goals. First, it helps a client to explore its neighborhood, i.e., find new peers that have higher upload speeds than the peers it currently has unchoked. Second, it helps peers that have just joined the swarm to get off the ground, i.e., to obtain their first pieces, before they have something they can reciprocate with, which is good from a social welfare perspective.

A Tit-for-Tat Analysis

BitTorrent is a large game, simultaneously played by hundreds or thousands of peers connected to each other in a swarm through random neighborhoods. Rather than modeling this entire system explicitly, we focus on a simpler repeated model.

As mentioned above, modeling P2P file-sharing systems like Gnutella as a repeated game is not suitable, because peers rarely interact with each other multiple times during their lifetime. However, the situation is drastically different in BitTorrent. Here, files are split into pieces and pieces into blocks, and leechers are exchanging blocks with each other, not just one, but possibly thousands, thus forming the basis for a repeated game. Of course, a peer is not playing a simple 2- player repeated game, as it is simultaneously connected to many different peers. But note that the low-level interactions in BitTorrent are purely bilateral. As long as two peers have pieces that the other doesn't have, they can continue their bilateral exchange. Effectively, a peer is simultaneously playing a repeated game with many different peers.

The unchoking algorithm implemented in the reference client can be interpreted as implementing a kind of tit-for-tat strategy. The unchoking algorithm makes a new unchoking decision every 10 seconds based on average performance of peers over the past 20 seconds. We can indeed model this as a repeated game, where all of the upload activity during one 10 second epoch is modeled as a single action in the stage game. For simplicity, let's assume a client only calculates the other peers' upload activity in the last 10 seconds, and let τ t denote

a threshold such that, if a peer uploaded more than τ then it was among the top 4 uploaders for time period t. In this case, the client will unchoke this peer in the next time period.
Thus, the tit-for-tat strategy for a peer i would look like this:
1.If peer j uploaded more than T^t in round t: upload to this peer in round t + 1.
2.If peer j uploaded less than T^t in round t: don't upload to this peer in round t + 1.
This is quite analogous to a tit-for-tat strategy: if player j cooperates in round t then i will cooperate in round t + 1, if player j does not cooperate then i will not cooperate. Coupled with the style payoffs for this system, we might then expect the reference strategy to form a Nash equilibrium in BitTorrent. Indeed, this strategy does provide another client with some incentive to upload. If client j does not upload to client i then client i will not unchoke client j, except via optimistic unchoking, which happens very rarely. On the other hand, by uploading to i with high speeds, then j gets unchoked by i. Thus, the general idea of "the more you give the more you get" is satisfied.
However, unlike in the game, BitTorrent peers have more than just two actions. They cannot only decide between sharing and not sharing, but they can vary how much bandwidth they upload to each individual peer.
However, it turns out that the strategy of the reference client is suboptimal and it is easy to find many "attacks", or in other words, "better strategies". Thus, using the reference client and sharing is certainly not a Nash equilibrium. In the next section we will explore some of these ways in which BitTorrent can be attacked.

IV Attacks on BitTorrent

As we have seen in the last section, a BitTorrent client has a very large degree of freedom in making decisions regarding how to interact with other BitTorrent clients. The default strategies are defined via the BitTorrent reference client, but can be improved upon for a client, selfishly. Here is a list of the main decisions a BitTorrent client can make:
1. How often to contact the tracker to receive a list of peers?
2. Which pieces to reveal to which agent?
3. How many unchoking slots to use?
4. Which agent to unchoke?
5. How much upload speed to give to each unchoked agent?
6. What data to upload to each unchoked agent?
The preferences of BitTorrent users may vary, where some may prefer to minimize their down-
load time, some may prefer to minimize the upload bandwidth they provide, and others may want to strike a good balance between the two. In the following sections, we will present different attacks on BitTorrent, each addressing one of these two goals, or both, often making different trade-offs between them.

Uploading Garbage Data

When a BitTorrent client joins a swarm it has no blocks to upload to other peers and must rely on optimistic unchoking by other peers. Thus, getting started in BitTorrent can be slow, in particular when there are many leechers and few seeders. One attack on BitTorrent that addresses this issue is to upload garbage data. The idea is to let the peer announce to all other peers that it has all pieces of the file, but when those pieces are actually requested, to then upload random data. However, using the SHA-1 fingerprint contained in the .torrent file, clients can detect incorrect data already on the block level, and can thus quickly identify a peer that uploads garbage data. Now, most clients usually ban any traffic coming from a peer that has uploaded garbage data before. Thus, this attack is no longer viable in practice.

BitThief: Exploiting Optimistic Unchoking

The second attack we consider was implemented and tested via a real-world BitTorrent client called BitThief. This client exploits the optimistic unchoking protocol that most BitTorrent clients follow, and as implemented in the reference client. The goal of BitThief is to download files via BitTorrent without uploading anything in return. Thus, BitThief must rely exclusively on the optimistic unchoking slots of other peers.

BitThief does two things differently than the reference client. First, it initially asks the tracker for a list of 200 peers instead of 50, which is the standard. Second, it re-announces itself to the tracker much more frequently than the reference client does (standard is between 15 or 30 minutes).

The goal of both these techniques is to grow a client's neighborhood as fast as possible. This is beneficial, because every additional peer in a client's local neighborhood means an additional chance of being optimistically unchoked in the next round. For typical torrents with a mix of seeders and leechers, BitThief can generally be used to download every torrent successfully, and that the completion time was on average between 2 and 4 times longer than with the reference client. For some torrents, in particular those with small files and a large number of seeders, BitThief can even be slightly faster on average than the reference client, without uploading a single bit of data itself. This is because BitThief's strategy of quickly opening many connections is particularly powerful in the first few minutes of a download process For small files, the additional optimistic unchoke slots obtained in the first few minutes may be enough to outweigh the lost reciprocation due to not uploading, such that the total download time is reduced.

Thus, BitThief shows an obvious vulnerability of the optimistic unchoking strategy of the reference client. However, the exploit relies on the fact that trackers are not very careful about this kind of behavior and don't protect against it. It would not be too complicated to prevent multiple requests from the same IP address in a 30 minute window, which would make this attack mute, unless the attacker has a way to obtain multiple IP addresses, which most users cannot easily do. Furthermore, the attack is only interesting to users who really care about minimizing their upload bandwidth. For most files using BitThief leads to significantly longer download times.

Strategic Piece Revelation

The reference client always announces truthfully which pieces it currently has, and follows the “rarest-piece first" policy when downloading from another peer. Being truthful in revealing what pieces a peer has is not necessarily optimal. The key insight is that a peer j is only interested in peer i as long as i has some pieces that j needs. Perversely, a peer i would like to maximize the amount of time other peers are interested in trading pieces. At first sight, it seems optimal for a peer to reveal all pieces it has, to maximize interest from others. However, this view is myopic, and it turns out that by under-reporting its pieces, a peer can benefit.

Consider Figure 2 where various scenarios from peer i's perspective are shown, and $\succcurlyeq_i$ indicates i's preferences (i.e., scenarios (a) through (f) are ordered in decreasing preference order). Agent I prefers having more peers interested in i rather than fewer, and thus (a) $\succcurlyeq_i$ (c) $\succcurlyeq_i$ (e), as well as (b) $\succcurlyeq_i$ (d) $\succcurlyeq_i$ (f). Next, it is disadvantageous for i, if other peers trade pieces among each other, because this potentially reduces the other peers' interest in i in the future. Thus, (a) $\succcurlyeq_i$ (b), and (c) $\succcurlyeq_i$ (d) and (e) $\succcurlyeq_i$ (f). Lastly, it is always better to have one additional peer being interested in

i, even if that peer is also interested in another peer in the swarm. Thus, (b) $\succcurlyeq_i$ (c) and (d) $\succcurlyeq_i$ (e).

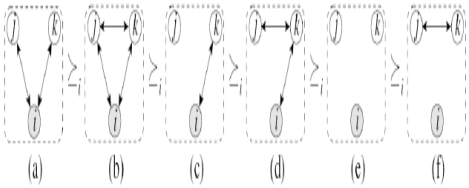

Figure 2: Strategic Piece : Peer i prefers to be as interesting Revelation .

Let βi represent i's bit-field, where βi(k) = 1 if peer i has piece k and 0 otherwise. As before, we assume that i reports its bit-field to new neighbors, and sends updates in the form of have-messages to existing neighbors. We let $\hat{\beta}$i denote the bit-field that i sends to neighbors, perhaps untruthful.

Consider the following algorithm:
Note that this piece-revelation strategy is counter to "rarest-piece first." Agent i only reveals a new piece to j when j is about to lose interest in i. And even then, i reveals the most common piece that it has and that j does not have. The idea is that providing j with a rare piece could increase

Algorithm1:Strategic Piece Revelation Algorithm
1.Let βi represent i's true bit-field , $\hat{\beta}$j denote j's bit field as j has announced it to i, and Li(j) the list of pieces that i has revealed to j.

2.If there does not exit any piece p such that $\hat{\beta}$j(p)=0 and βi(p)=1 then quit ;I cannot truthfully gain j's interest.

3.Find the piece p with $\hat{\beta}$j=0 and βj(p)=1 that maximizes the number of other neighbors l for which (a)l also has p, or (b)I has revealed p to l before.

4.Send a have-message to j, revealing that I has piece p, and add p to Li(j).

other peers' interest in j, thereby detracting interest from i. Thus, by following this algorithm, i tries to maintain the scenario from Figure 2 (a) as opposed to (b); (d) or (f). Note that in contrast to the attacks that involve uploading garbage data or contacting the tracker more frequently than the default client, the strategic piece revelation strategy cannot easily be defended against. It is private knowledge of each peer which pieces it has, and a single other peer cannot detect whether a peer underreports pieces or not.

Even though using strategic piece revelation is beneficial from an individual peer's point of view, a remaining question is its effect on overall social welfare. Experimental studies have shown that if all peers use strategic piece revelation, then the average download time for the whole population increases by 12%. All peers withhold information from each other, thereby leading to suboptimal allocations of bandwidth. It is perhaps surprising that the increase in download times isn't higher.
However, peers continue to reveal more and more pieces to each other, to garner each other's interest. Thus, the negative effects are particularly strong at the beginning of the download, but become less severe over time.

Yet, from a social welfare perspective, it is unfortunate that the BitTorrent protocol allows clients to benefit in this way. An interesting research question is whether a protocol could be designed that incentivizes truthful piece revelation.

BitTyrant: Strategic Unchoking

The last strategic attack we consider is also the most powerful one, in terms of decreasing a BitTorrent peer's average download time. The strategy was implemented in a BitTorrent client called BitTyrant in 2006, its effcacy has been experimentally tested, and the client is freely available for download.

The main idea used in BitTyrant is that the unchoking strategy, or more generally, the upload strategy implemented in most BitTorrent clients may be sub-optimal. Remember that most clients use a fixed number (often 4) of unchoking slots, upload to those clients that reciprocate with the fastest download rate, and split their bandwidth equally among all of the unchoked peers, called the equal-split policy.

The BitTyrant client exploits this insight by deviating from the reference client in three different ways:

1.How many upload slots to use? Instead of using a fixed number of upload slots, BitTyrant dynamically adjusts the number of upload slots, dependent on whether using an additional slot increases or decreases performance.

2.Who to unchoke? Instead of unchoking those peers with the fastest download speed, Bit-Tyrant unchokes those peers where the ratio of received download speed to provided upload speed is best. (Thus taking a `return-on-investment' perspective.)

3.How fast to upload to unchoked peers? Instead of using the equal-split strategy, BitTyrant dynamically adjusts the upload bandwidth provided to every unchoked peer, with the goal to upload at the minimum rate at which that peer is willing to reciprocate.

To perform this strategy, the BitTyrant client needs to do a little more bookkeeping than the reference client does and some estimation of certain parameters. In particular, for each peer j, the BitTyrant client i maintains an estimate of dj , the current download rate that j provides its unchoked peers, and uj , the upload rate required to become unchoked at j. If i is unchoked by j, then dj is simply the observed download bandwidth. Otherwise, i must estimate dj based on secondary information. Remember that peers send have-messages to all peers in their neighborhood, every time they complete a new piece, even to peers that are currently unchoked. Based on the frequency with which such have-messages arrive from peer j, i can estimate j's total download bandwidth. BitTyrant then uses this estimate as an estimate for j's total upload capacity. This heuristic is justified given the measurements , which suggests a roughly linear relationship between upload and download speed. Assuming an equal-split upload rate for peer j, the total estimated upload capacity is then divided by the total number of upload slots used at this speed (e.g., 4), which provides an estimate of dj , i.e., the download rate that i can expect to get from j. To estimate uj , i.e., the upload rate required to become unchoked at j, the BitTyrant client uses the distribution of equal split capacities observed in prior measurements, Now consider Algorithm 2, where the individual steps of the BitTyrant unchoking strategy are described:

Algorithm2:The BitTyrant Unchoking Algorithm

1. For each peer j, peer i maintains estimates of expected download rate dj and expected upload rate required for reciprocation uj .

(a) If client i is unchoked by j, then dj is the observed download bandwidth. Otherwise, dj is inferred indirectly from j's block announcement rate.

(b) Initialize uj using the distribution of equal split capacities observed in prior measurements.

2. Each round, rank order peers by decreasing ratio $\frac{dj}{uj}$ and unchoke those of top rank until the upload capacity is reached.
$\frac{do}{uo}, \frac{d1}{u1}, \frac{d2}{u2}, \frac{d3}{u3}, \ldots,$

Choose k|$k \sum_{j=0}^{k} uj \leq capi.$

At the end of each round, for each unchoked peer j:

(a) If peer j does not unchoke i: uj←(1+α)uj

(b) If peer j unchokes i: dj← observed rate.

(c) If peer j has unchoked i for the last r rounds: uj ←(1-γ)uj

By using this algorithm, a client i aims to download from those peers that reciprocate most for the least number of bytes uploaded to them. By ranking peers j in decreasing order of their $\frac{dj}{uj}$ ratio, it keeps adding new upload slots, filling them with the most valuable peers, until the upload budget capi is reached. Of course, the parameters α,y and r used in the update step are open for fine-tuning. In their experiments, decreased uj by y= 10% if the peer reciprocated for r = 3 rounds, and increased uj by α= 20% if the peer failed to reciprocate after being unchoked during the previous round.

V CONCLUSION

As a conclusion, it is worth noting that BitTorrent works amazingly well, despite all of the possible ways it can be attacked. As we have seen, many BitTorrent clients that can significantly improve a user's download performance have been proposed, and are freely available for download. Yet, almost nobody uses those clients in practice. The most-widely used clients implement strategies very similar to the reference client. It seems that most BitTorrent users are relatively happy with the performance they obtain using those clients. It is also worth noting that the user experience of those clients is also much better compared to the strategic clients we described in this paper, with graphically appealing user interfaces, integrated search, and much more. Many (rational) users may decide that a nice user interface is more important than a slightly faster download experience. However, some users purposefully select very “altruistic" settings, i.e., they program their clients to upload at least as much as they download. Thus, there are many open questions regarding what motivates users to share in P2P file communities, even when they don't benefit directly from those actions.

ACKNOWLEDGEMENTS

We are greatly delighted to place my most profound appreciation to Dr.K.Satayanarayana Chancellor of K.L.University, Dr.K.Raja Sekhara Rao Principal, S.Venkateswarlu Head of the Department , and Dr Subramanyam in charge for M.Tech under their guidance and encouragement and kindness in giving us the opportunity to carry out the paper .Their pleasure nature, directions, concerns towards us and their readiness to share ideas enthused

us and rejuvenated our efforts towards our goal. We also thank the anonymous comments references of this paper for their valuable

REFERENCES

[1]Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang. Exploiting bittorrent for fun (but not profit). In 5th International Workshop on Peer-to-Peer Systems (IPTPS), 2006., is cheap. In Fifth Workshop on Hot Topics in Networks, 2006.

[2]Laurent Massoulie and Milan Vojnovic. Coupon replication systems. In Proceedings of APiatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in BitTorrent? In 4th USENIX Symposium on Networked Systems Design & Implementation, 2007.

[3]J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS), 2005.

[4]Mike Ruberry and Sven Seuken. Sharing in bittorrent can be rational. Working Paper, Harvard
University, 2011.

[5]Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. First Monday, 5(10), 2000.

[6]R Axelrod and WD Hamilton. The evolution of cooperation. Science, 211(4489):1390{1396, 1981.

[7]Bram Cohen. Incentives build robustness in BitTorrent. In Proceedings of the First Workshop onEconomics of Peer-to-Peer Systems, 2003.

[8]Daniel Hughes, Geooff Coulson, and James Walkerdine. Free riding on gnutella revisited: The bell tolls? IEEE Distributed Systems Online, 6(6), 2005.

[9]N.Immorlica, B.Lucier, and B. Rogers. Emergence of cooperation in anonymous social networks through social capital. In Proceedings of the 11th ACM Conference on Electronic Commerce (EC), 2010.

.