Multiple Identities in BitTorrent Networks

Jin Sun, Anirban Banerjee, and Michalis Faloutsos

Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
{jsun, anirban, michalis}@cs.ucr.edu

Abstract. Peer-to-peer (P2P) file sharing systems have become ubiquitous and at present the BitTorrent (BT) based P2P systems are very popular and successful. It has been argued that this is mostly due to the Tit-For-Tat (TFT) strategy used in BT [1] that discourages free-ride behavior. However, Hale and Patarin [2] identify the weakness of TFT and hypothesize that it is possible to use multiple identities to cheat. To test this hypothesis we modify the official BT source code to allow the creation of multiple processes by one BT client. They use different identities to download the same file cooperatively. We experiment with several piece selection and sharing algorithms and show that BT is fairly robust to the exploitation of multiple identities except for one case. In most cases, the use of multiple identities does not provide siginificant speedup consistently. Interestingly, clients with multiple identities are still punished if they do not maintain a comparable upload rate with other normal clients. We attribute this to the robust way that the Tit-For-Tat policy works. From our experiments we observe that the BT protocol is rather resilient to exploits using multiple identities and it encourages self-regulation among BT clients.

Key words: Peer-to-peer networks, BitTorrent, fairness, resource allocation

1 Introduction

Peer-to-peer file sharing systems enable large-scale content distribution, by allowing users to cooperate with each other and voluntarily share their resources, mostly files. There are numerous P2P clients available nowadays, such as KaZaA [3], GNUtella [4], eDonkey [5] and BitTorrent [1]. P2P applications have been known to contribute significantly to Internet traffic as indicated by some measurements of the Internet backbone recently [6]. Among all these file-sharing applications, BitTorrent seems to be the most popular one and has evolved to account for a large portion of the P2P traffic on the Internet [7].

BitTorrent achieves a higher level of robustness and resource utilization than most currently known cooperative techniques [1]. It works by grouping users, who have common interests in downloading a specific file, together into swarms

to cooperate with one another to speed up the download process. BitTorrent is distinctive for its "choking/unchoking" algorithms that promote high-level reciprocation among users. The "Tit-for-Tat" (TFT) strategy, an integral part of BT, encourages users to cooperate: A BT client uploads more generously to its peers which reciprocate and provide it with the data it needs. Those who do not share run the risk of being *neglected* by the swarm.

Some people believe that the TFT strategy is the key strength for BT's success. However, Hale and Patarin [2] highlight some weaknesses of the TFT policy and state that it is possible to fake identity in BitTorrent and get free ride because there is no mechanism to provide trust-based identification authentication. Although it has been pointed out in [2] that it is possible to use fake peer IDs in BitTorrent networks, to the best of our knowledge there is no detailed description or implementation of this approach in related literature.

In some sense, it is natural to expect that selfish users would try to get free-rides, at the cost of other benevolent users in the swarm given the fact that there is no monitoring mechanism such as reputation management in P2P networks [8, 9]. In addition to the tendency to free-riding, a user may also wish to use multiple identities to speed up the download process. For example, a peer may have two different IPs and want to create two BT processes that can work cooperatively to download the same file without any overlap. This is especially attractive when the file being downloaded is very large. However, this is not currently supported by the official BT client and other compatible implementations. The questions we attempt to answer through our research effort are:

1. Does use of multiple identities in BT help speed up downloads?
2. How is the swarm effected by such selfish behaviors?

Motivated by these interesting questions, we have developed and implemented a modified BT client based on the official BT source code. Using the modified BT client, a user can (1) *create multiple processes with different IDs* and these (2) *processes can cooperatively download the same file* using different piece selection and sharing algorithms. Our implementation seems to be the first to explore this idea of multiple identities for BT users. To answer these questions we conducted extensive experiments with the modified BT client in the CS labs at UC Riverside. We list our contributions below:

1. **Using multiple identities does not provide consistent benefits:** From our experiments, we find that the modified client can only achieve limited speedup in very specific cases. We find that BT is fairly robust to the exploitation of multiple identities.
2. **Modified clients are penalized like normal clients:** We observe that modified clients need to maintain a comparable upload rate with other normal clients, otherwise they suffer from high file download latency.

We attribute this robustness of BT to the Tit-For-Tat policy, which is an effective mechanism to implement fairness in the swarm. The rest of the paper is

organized as follows. In Section 2, we provide some more background information on BitTorrent systems. In Section 3, we describe how to create multiple processes with different IDs to download the same file cooperatively and propose a few different piece selection and sharing algorithms. In Section 4, we describe our experiments with the modified client and analyze the results. In Section 5, we conclude this paper.

2 BitTorrent file sharing system

Before describing our modifications to the BT client, it is necessary to give some more details on the BitTorrent protocol.

Data: BitTorrent achieves efficient content distribution by swarm download. The basic idea is to split a file into equal-size *pieces* and have clients download pieces from different peers simultaneously [1]. Each piece is further split into *blocks*, which is the basic transmission unit in BitTorrent. Each node sends a request for each block it does not have and also advertises the blocks it has to its peers. To download a target file, a user first needs to find a *torrent* file which contains enough information about the file to download, especially the URL of a tracker.

Network: A tracker is a centralized machine that keeps track of all the peers participating in a swarm. Active peers report their status once a while to the tracker and also get up-to-date information from the tracker about one another. Then these peers can download/upload blocks among themselves and share the burden of file distribution cooperatively. In the BT protocol, individual peers choose their own Peer IDs arbitrarily. Peer IDs are not used for authentication. Trackers track these IDs only to facilitate peer-to-peer connections. So it is possible to use multiple peer IDs on behalf of a user without being detected by either the tracker or other peers. It should also be noted that peers cannot be differentiated by IP addresses alone in BT networks for two reasons. One is that BT supports multiple connections behind *Network Address Translator*s (NATs), which means that several peers behind the same NAT can have the same global IP address. The second reason is that connections from behind a *proxy* are also accepted in BT.

To summarize, BT does not prevent a user from launching multiple cooperative processes to download the same file. The question is whether users can take advantage of this seeming weakness.

3 BT client implementation with multiple identities

In this section, we describe the changes we have made to the original BT client implementation. These changes enable the creation of N BT processes (with N different IDs) for a user to download one file cooperatively. If one process has already downloaded some pieces, then the other processes do not need to request and download the same pieces again. We can use different approaches to assign individual pieces to these process.

When starting to download a file, a user can specify the total number of processes to be created, say N. Each process has a unique Peer ID and can register with the tracker successfully and request for a peer list.

Once a process receives the peer list from the tracker, it acts as a normal BT client following the BT protocol. For other peers, these processes do not exhibit any abnormal behavior other than that they may have the same IP address.

Our approach differs from the standard mechanism regarding piece selection. A process sends a new request to its peers for a piece, only picking one from its own "task list", instead of from the entire piece list. By splitting the target file into several parts and assigning different parts to different processes, we cut down the actual download size by N for individual process. Naturally, we expect that this approach may reduce the overall download time for one file although we will soon see that some BT dynamics prevent this from happening easily. There can also be some variations of the basic scheme. For example, rather than creating fixed "task lists" for individual processes, it is also possible for these processes to exchange information among themselves and decide which pieces to download next by skipping those pieces that have been downloaded or are being downloaded by other processes. Another variation is whether these processes can upload to their peers those pieces that have been downloaded or are being downloaded by other processes.

Limitations: It should be noted that we cannot create an arbitrary number of processes without limit for several reasons. First, the download capacity (bandwidth) is a limited resource and shared by all the processes concurrently if they run on the same machine. When the total number of processes increases, the download bandwidth for each process decreases, thus the download time will be affected. This can be more conspicuous when both upload and download traffic are multiplexed on the same physical link and the download traffic is also affected by the upload traffic. Second, BitTorrent protocol is built on TCP and usually all the processes have to share the same TCP buffer and too many concurrent network connections would degrade TCP performance. Third, TCP flow control might also affect the performance when too many connections are established.

Next, we describe the four different algorithms for piece assignment and sharing among concurrent processes in detail.

3.1 Fixed-range piece assignment with no piece-sharing among processes

This is the most straightforward approach to share the download task among concurrent processes. The file to be downloaded (the target file) is divided evenly into N parts. Each process is assigned a part, which is the "task list" that needs to be accomplished.

When a process sends a new request to its peers, it only chooses a piece from its own "task list", instead of from the entire piece list. This approach is simple and incurs the least overhead and these processes can run on different machines.

However, in this simple approach, there is no information exchange between different processes. Each process behaves independently and there is no piece sharing among processes. One possible limitation of this approach is that each single process may have higher probability of being choked by others, because it can only offer a subset of the file pieces that others might want.

3.2 Fixed-range piece assignment with piece-sharing among processes

In the second approach, when generating a new request, the N concurrent processes still use the same piece assignment algorithm as discussed earlier. However, here processes also try to coordinate with one another. They exchange information among themselves about what pieces have been downloaded as a whole. That is, when each process announces what pieces they have downloaded, it will also include those pieces that have been downloaded by other processes. Thus each modified process is also able to upload pieces downloaded by other processes. This may help to reduce the chance of these modified processes being choked by others because they can offer to upload more pieces.

To share pieces among these different processes, there are two ways. One is to set up dedicated connections among these processes and then they transfer the actual pieces they have downloaded to one another. However, this incurs more communication overhead and implementation complexity. The other way to to have all the processes run on the same machine and each process can copy the pieces it has downloaded as individual files to a common directory accessible to all. This allows each process to check this directory and also upload to its peers pieces downloaded by other processes.

One caveat with this approach is that since these processes need to run on the same machine, they may be prevented from connecting to the same peer and they also need to share the available the bandwidth and the available TCP buffer. However, because of its simplicity we choose this approach in our implementation.

3.3 Random piece assignment with no piece-sharing among processes

The third approach differs in terms of its *random* piece selection algorithm: Whenever one process is about to select a piece to request, it no longer chooses from the fixed subset of pieces like the two aforementioned approaches. Instead, it randomly picks one among all the pieces that are not requested/downloaded by any other processes. Each process needs to request the piece which the BitTorrent protocol returns (surely that piece needs to be among those that other connecting peers can provide), otherwise it may lose the opportunity to get pieces from others if they stick to the fixed piece assignment discussed earlier. When they generate less requests to other peers, they may not make the best use of the available download bandwidth.

Obviously, it is necessary for each process to be aware of what pieces other processes have already had. Each process needs to check (via file) what pieces the other processes are downloading and avoid downloading the same ones. Therefore, we can use the same approach by copying every piece to a common directory and then any process needs to check this directory first before generating a new request.

Similar to the previous approach, this simplifies implementation although it requires all processes to run on the same machine.

3.4 Random piece assignment with piece-sharing among processes

In this approach, the major difference with the previous scheme is that each process also advertises to its peers pieces downloaded by other processes. So each process can also upload to its peers pieces downloaded by other processes and may reduce the possibility of being choked by its peers.

Due to space related constraints, we refrain from describing changes made to the original BT client code. However, our source code and commentary are available on our web site [10] to benefit the research community.

4 Experiments and Discussions

To validate our methodology, we have conducted experiments in swarms created in the CS labs at UC Riverside. In all experiments, we run both the modified clients alongside the original BT clients to compare their performance.

4.1 Experiment Setup

Experiments in UCR intranet are conducted in an easily controlled environment, so they can help us to assess the impact of different approaches precisely. We deploy 40 nodes to download a 30MB file. In this swarm, there is one tracker and 1 upto 6 seeders. Only one leecher runs our modified client (which actually creates multiple processes as described in Section 3), other leechers and seeders use the original BT client. We use BitTorrent's default parameter settings for all nodes except:

- *Node Degree*
  This parameter determines the neighborhood size, i.e., the number of concurrent connections a node will maintain. It is set to 10 in our experiments.
- *Maximum upload rate*
  We only change it for the modified client. For normal clients and seeders, this parameter defaults to 20 Kbps.
- *Number of processes created*: This parameter applies to the modified BT client only. One modified BT client can create 2–5 processes to download the same file cooperatively.

4.2 Experiment Results and Analysis

Although we have experimented with four different approaches, we first show the results for the two approaches that do not do piece-sharing and then later we discuss the results for the approaches that use piece-sharing.

Ranged piece assignment with no piece-sharing among processes To evaluate this approach, we compare the performance of our modified client with the original BT client in terms of download completion time by varying:

1. the number of seeders in the swarm
2. the maximum upload speed for the modified client
3. the number of concurrent processes in the modified client

respectively.

The results are shown in Figs. 1, 2 and 3 respectively.

Comparison of download time: We present Fig. 1, which compares the download time between normal BT clients and the modified BT client. The latter uses two concurrent processes to download the file cooperatively. We vary the number of seeders to see how it can affect the download speed. In this set of experiments, both normal BT clients and the modified BT client have the same default upload speed of 20Kbps. This graph clearly shows that when there is only one seeder in the swarm, the modified client can hardly download faster than the original client. When the number of seeders increases to 2, there is a significant speedup achieved by the modified client. However, when the number of seeders increases further, the speedup is more level for the modified client.

It is easy to explain the speedup for the original client because more seeders translates to more upload capability offered by the swarm. For the modified client, the case is a bit more complicated. When there is only one seeder in the swarm, the effect of the rarest-first piece selection algorithm is the most conspicuous: The availability of any piece is mostly limited by the upload capability of the seeder which is only 20 Kbps. Most of the time the two processes that are created by the modified client are blocked by the availability of the requested pieces and they in fact spend much time waiting for the pieces to become available in the swarm.

Effect of varying number of seeders: When there are more than one seeders in the swarm, the modified client can achieve much higher speedup than the original client. The reason is that the availability of the pieces is much less constrained by the upload capacity of the seeders and that each process only needs to download one half of the file so the download time is much shorter.

When more seeders are available, they do not help the modified client much because the upload capacity of seeders are shared by both original clients and modified client and increasing one seeder does not increase the capacity much.

Because modified client benefits from more than one seeders as discussed above, in later experiments we focus on the case when there are three seeders in the swarm.

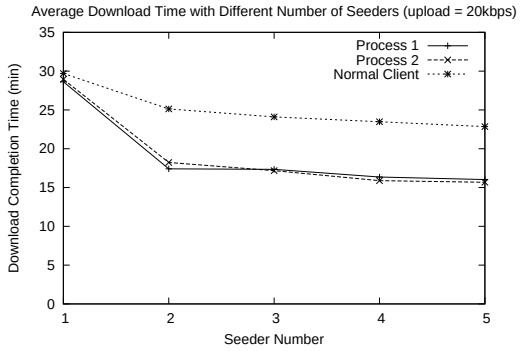

Fig. 1. Average download time with different number of seeders in the swarm (upload rate = 20 Kbps)

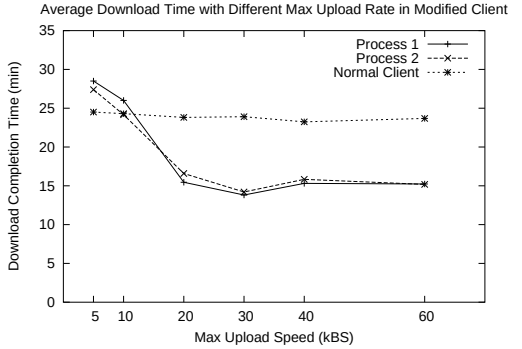

Fig. 2. Average download time with different maximum upload speed in our modified BT client

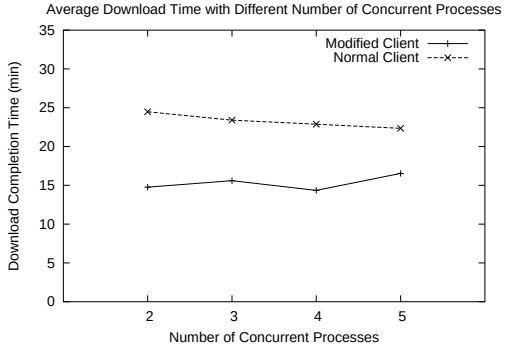

Fig. 3. Average download time with different number of concurrent processes in our modified BT client. Processes number varies from 2 to 5

Effect of varying upload speed: In Fig. 2, we show the effect of varying upload speed for the modified client. It can be observed that for the modified client the download time curve follows three stages. When the modified client decreases its upload speed below the norm, its download time increases even though each process only needs to download only one half of the file. The reason is that these processes can offer only half of the pieces that other peers are interested in and they may be choked more often by other peers. When they are unwilling to upload more data to other peers, then they are punished more severely. So they should not decrease upload rate below the norm.

When modified client increases its upload rate, the benefit of the need to download only one half of the file is more conspicuous so the download time decreases sharply. This also partially compensates for the deficiency of the limited availability of pieces from these processes. They can increase their upload rate to help reduce the possibility of being choked by others. However, further increasing the upload rate will not help much beyond a certain point because by that time these processes cannot necessarily achieve the maximum upload rate as they have limited pieces to offer and other peers will not request pieces from them so often.

It is also interesting to note that increasing upload rate may negatively affect the download speed as shown from the slight tip at 40 Kbps. This is due to the reason that these processes have their upload and download traffic multiplexed and too much upload traffic can reduce their download speed and seeders may also reduce upload rate to these processes further.

This shows that the modified client should not cheat by decreasing upload rate below the norm and it should not be too generous either.

Effect of varying number of processes: Fig. 3 shows the results when we vary the number of processes created by the modified client. As usual the number of seeders is still three.

We can see that for normal clients, the download time decreases slightly with the increase of processes created by the modified client. This is easy to explain because with more processes participating in the swarm, they offer more upload capacity and normal clients can benefit from this. However, for the modified client the benefit is not so clear-cut. Increasing processes reduces the workload of individual processes, however this also reduce the number of pieces that these processes can offer to others in exchange for higher download speed. This also increases the possibility that these processes are choked by others. So it is not to the modified client's advantage to increase the number of processes arbitrarily and creating two processes has already helped a lot.

We also note that in a swarm full of free-riders who refuse to upload anything, our modified client performs much better than the other normal clients. However, the overall download time for all the clients in the swarm increases a lot and everyone suffers. In reality, users have to behave to avoid the slowdown. So BT protocol effectively curbs selfish behavior and encourages self-regulation among users.

Random piece assignment with no piece-sharing among processes In this approach, each process does not stick to a fixed-range of pieces to download, instead it randomly picks one piece that are not requested by other processes. Please note that in this approach the processes created by the modified client need to run on the same machine and have to share the available download and upload bandwidth.

To evaluate this approach, we also vary the number of seeders, maximum upload speed for the modified client and the number of concurrent processes and compare the download time between the modified client and the original BT client. The results are shown in Figs. 4, 5 and 6.

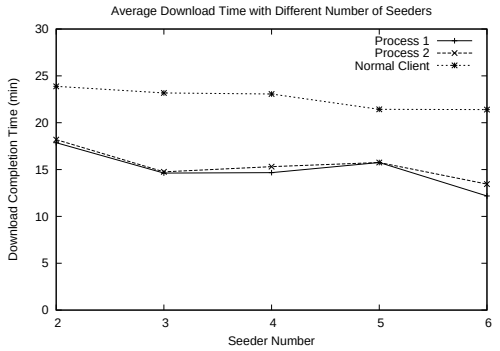

Fig. 4. Average download time with different number of seeders

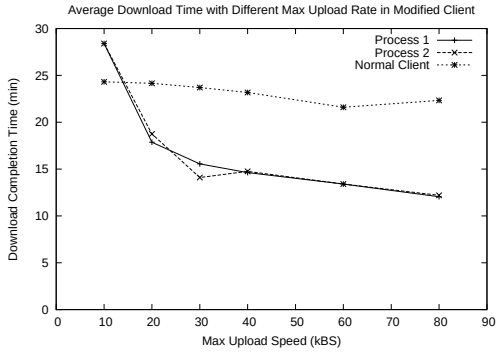

Fig. 5. Average download time with different upload speed in our modified BT client

Comparison of download times: Evaluating Fig. 4 alongside Fig. 1, we can observe that the download time varies a lot more in this approach. The main

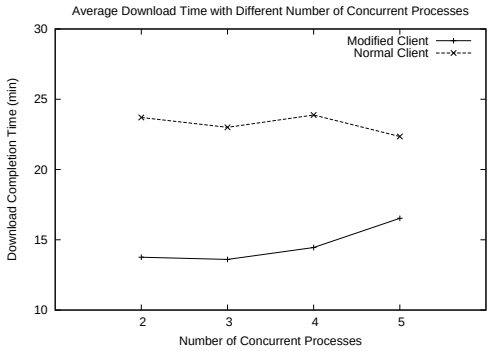

Fig. 6. Average download time with different number of concurrent processes in our modified BT client

reason is that the load for the two processes may be uneven: One process may download more pieces than the other process because once it starts downloading the rarest piece according to BT's piece selection algorithm, the other cannot request it and have to wait for the next available one. The other process may be choked more often by other peers. However, when there are more seeders in the swarm, then the other process may still benefit a lot from the increased upload capability offered by these seeders and the download time is reduced.

Effect of varying upload rate: Fig. 5 shows that this approach benefits more by increasing the upload rate in comparison with Fig. 2. This can be explained as follows. Because these two processes may have uneven load, one process that has less pieces to offer may need to increase its upload rate to offer more data to its peers to reduce the possibility of being choked by others. Fig. 6 shows that with the default upload rate, the download time cannot be reduced by increasing the number of processes. This is easy to explain because when there are more processes, the load of each process is distributed more unevenly and usually one process may shoulder most of the load and other processes have little to download and hence cannot contribute much to the download task.

Approaches that do piece-sharing Earlier we have shown the results for the approaches that do not do piece-sharing. We have also experimented with approaches that do piece-sharing. However, these **piece-sharing approaches also yield only satisfactory results under a few cases**:

1. When there are *more* seeders in the network.
2. When maximum upload rate is increased.

The speedup is limited due to the following reason. Although these processes may upload more pieces to their peers, this does not increase the number of pieces that are available to themselves even though their peers are willing to reciprocate with more pieces. In this case, their upload and download rate are unbalanced and they end up with faster uploading but without faster download.

To summarize, *there is no easy way to achieve consistent speedup under any case by using multiple identities* with either piece selection and sharing algorithm. We do not mean to say that all possible strategies to achieve download speedup have been exhausted. However, it should be noted that given the highly dynamic nature of BT networks, it would be extremely challenging if ever possible to devise a strategy that works in most if not all scenarios.

5 Conclusion

In this paper, we have described the seeming weakness in BT systems, i.e., the possibility of using multiple identities to cheat which was first identified by Hale and Patarin [2]. However, there was no implementation that exploit such weakness. Then we describe our modification to the official BT client implementation to allow the creation of multiple processes in one BT client to download the same file cooperatively. Our extensive experiments with different piece selection and sharing algorithms show that it is possible to achieve speedup in a few selected cases. However, no strategy can guarantee a speedup. In fact, increasing the number of processes may even hurt the performance of the modified client. The modified BT client helps only when the user is a free-rider, however then the overall download speed of that user will be low and the the overall network performance degrades with many such users. This shows that the BT protocol is rather resilient to the exploit of using multiple identities. We argue that if such exploits were easily achievable, then BT systems would have suffered a breakdown long time ago.

References

1. Cohen, B.: Incentives Build Robustness in BiT Torrent. http://www.bittorrent.com/bittorrentecon.pdf
2. Hale, D., Patarin, S.: How to cheat BitTorrent and why nobody does. In: Technical Report UBLCS 05/12/05. (2005)
3. Liang, J., Kumar, R., Xi, Y., Ross, K.K.: Pollution in P2P File Sharing Systems. In: IEEE INFOCOM. (2005)
4. GNUtella. http://en.wikipedia.org/wiki/Gnutella
5. EDonkey. http://en.wikipedia.org/wiki/EDonkey2000
6. Karagiannis, T., Broido, A., Brownlee, N., Faloutsos, M.: Is P2P dying or just hiding? In: Globecom, Dallas, TX, USA. (2004)
7. Mennecke, T.: BitTorrent Remains Powerhouse Network. In: Slyck News. (2005)
8. Buragohain, C., Agrawal, D., Suri, S.: A Game-Theoretic Framework for Incentives in P2P Systems. In: International Conference on Peer-to-Peer Computing. (2003)
9. Golle, P., Leyton-Brown, K., Mironov, I., Lillibridge, M.: Incentives For Sharing in Peer-to-Peer Networks. In: Proceedings of the 3rd ACM conference on Electronic Commerce. (2001)
10. Sun, J.: Multiple Identities in BitTorrent Networks. http://www.cs.ucr.edu/~jsun/BitTorrent.html