The Third International Multi-Conference on Computing in the Global Information Technology

[KSKS08] *Sebastian Kaune, Jan Stolzenburg, Aleksandra Kovacevic, Ralf Steinmetz;* **Understanding BitTorrent's Suitability in Various Applications and Environments.** In: The Third International Multi-Conference on Computing in the Global Information Technology (ComP2P '08), IEEE Computer Society Press, July 2008. Seite 256-262

Understanding BitTorrent's Suitability in Various Applications and Environments

Sebastian Kaune, Jan Stolzenburg, Aleksandra Kovačević, Ralf Steinmetz
Multimedia Communications Lab, TU Darmstadt
{kaune, stolzenburg, sandra, steinmetz}@kom.tu-darmstadt.de

Ruben Cuevas
Universidad Carlos III de Madrid
{rcuevas}@it.uc3m.es

Abstract

Peer-to-peer systems have recently emerged as an attractive alternative to client/server approaches, especially in the area of content distribution. By efficiently leveraging the available upload bandwidth of the end users, BitTorrent becomes a de facto standard for scalable content distribution. Inspired by its success, many companies try to shift the major upload burden from their rented source servers to end users by using this protocol, since many hosting sites charge them based on the used egress capacity.

In this paper, we perform an in-depth study of the overall performance of BitTorrent in its entirety, in order to get a broader understanding of its suitability for different applications domains. We analyze its performance from a bifocal perspective, namely that of the content providers and that of the end users. In this context, we find that the decrease of the source server's upload capacity has a highly negative impact on the overall protocol performance. In addition, it is shown that giving incentives to peers to stay online after completing downloading does not pay off.

1 Introduction

Peer-to-peer (p2p) systems have recently emerged as an attractive alternative to client/server approaches, especially in the area of content distribution. By leveraging the available upload bandwidth of the end users, or so called *peers*, these systems overcome the limitations of centralized approaches in that they have the potential to scale to large network sizes.

Especially in the domain of file sharing, BitTorrent [9] becomes a de facto standard for scalable content-distribution justified by its efficiency in spreading content to interested parties. Accordingly, the inventor of BitTorrent, *Bram Cohen*, recently extended the web portal of the original client, also known as mainline client, to a managed platform for commercial-grade content delivery [2].

However, despite of the fact that BitTorrent is proven to be highly effective in distributing content over the Internet, users often have to wait long time periods when downloading a very popular file. In addition to this, many hosting sites charge content providers based on the use of the egress capacity. Thus, many companies are highly interested in shifting the major upload burden from their rented source servers to the end users; to this end, the peer-to-peer paradigm, and especially BitTorrent, seems to be the most appropriate technology. For instance, Blizzard Entertainment spreads patches, videos and demos of the successful online game "World of Warcraft" to end systems by using a specific BitTorrent client named Blizzard downloader [3].

Inspired by these demands, there are still open questions that need to be answered in order to get a broader understanding of the suitability of BitTorrent in different application domains. Within this paper we give answers to the following questions which we feel are, amongst others, of major importance in the context of content distribution:

- What are the effects on BitTorrent's overall performance if the source server has only a very limited upload capacity? Can content providers reduce their costs in terms of saved upload bandwidth or will end-users suffer from highly increased download times?
- Does it pay off for content providers to encourage users to stay online for a predefined time interval after they have finished their downloads? If so, are there any correlations to the download times of the end users?
- Does BitTorrent work well solely with large files? How appropriate is BitTorrent for small files, for example, patches, small software updates or MP3's?

978-0-7695-3275-2/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCGI.2008.24

- How much of the uploaded data is served by the source server? In general, do peers have to provide the same amount of uploaded data? Is it evenly balanced among all peers or does a small group has to carry the major burden?
- Are there any correlations between the download times and the point in time at which a peer joins a torrent? Furthermore, does the network joining time influence a peer's carried upload capacity?

To answer these questions, we present a detailed study of the overall performance of BitTorrent in two different use case scenarios. The first scenario describes a so called *flash crowd*. That is, this scenario reflects a common real world situation in which a very popular file is made available by a content provider and a large number of peers try to download it within a short time frame.

The second download scenario, named *constant stream*, reflects the situation in which one specific file is stored on a web server together with many other files over several weeks or months. This scenario covers the case in which rented web servers are combined with software archives or mirror servers offering reams of different video files, software updates, and patches. However, due to the large number of files, the upload bandwidth of the content server is very limited, meaning that only a limited upload capacity is available per file.

The main contribution of this paper is the analysis of the overall performance of BitTorrent in *its entirety*. That is, we investigate its behavior and effectiveness under various influential factors by using the proposed use case scenarios as fundamentals of our research. Previous work, e.g. [11] [10] [8], focuses, instead, on specific mechanisms and algorithms applied in BitTorrent. In particular, they solely explore the impact of those algorithms on the performance of BitTorrent by comparing them to existing or newly introduced alternatives.

The rest of this paper is structured as follows. Section 2 provides a brief background on BitTorrent. In Section 3 and 4, we present our evaluation methodology and the results of our analyses. Discussion and remarks are given in Section 5. Related work is in Section 6, while in Section 7 we conclude the paper.

2 Overview of BitTorrent

BitTorrent is a P2P file distribution protocol whose goal is to enable fast and efficient content replication over the Internet. It leverages the upload bandwidth of the downloading peers and thereby increases the global system capacity with increasing network size. In particular, the basic idea is to split a file into pieces of typically 256 KB, and then to further divided each piece into blocks of 16 KB. All information necessary to download a specific file can be obtained from a meta info file, also called a *torrent*, which is usually stored on a web server. Each torrent also contains the IP address and port of a central component called the *tracker*. This component periodically receives statistics from nodes currently involved in the download process and keeps track of when peers join and leave the system.

To join the system, each peer downloads a torrent file and then contacts the corresponding tracker to obtain a list containing a random subset of the peers (typically 50 nodes) currently in the system. The new node then tries to establish a connection to its so called *neighbors* in order to be incorporated into the current download process. Nodes replicate data among each others at a block-level by using swarming techniques. For further details about the applied algorithms of BitTorrent, we refer the interested reader to [2].

3 Evaluation Methodology

Inspired by the increasing demand of BitTorrent in commercial applications, it is essential to analyze its performance in different use case scenarios in more detail. In particular, a distinction has to be made between commercially driven closed source clients such as the Blizzard Downloader, and open source variants like Azureus [1], and μTorrent [7].

Analyzing the latter comes with many influential factors which can highly falsify the results of measurement, especially in environments consisting of diverse descendants of the mainline client. For instance, reason for this might be user-manipulated software clients, different parameter setups or even specific choking algorithms applied by each compatible client. Hence, conclusions about the real performance of BitTorrent might be misleading in these heterogeneous environments.

Instead, a closed source variant allows the creation of a homogenous environment that is, on the one hand, much more controllable for developers, and that has, on the other hand, a positive side effect that the underlying algorithms of BitTorrent are harder to manipulate. Not surprisingly, the mainline client of BitTorrent is closed source as of a few months ago.

3.1 Simulation Model

For this reason, we assume for our experiments, a homogeneous environment consisting entirely of peers using the mainline client 5.2.0. By inspecting the source code of this client, we have accurately implemented it in PeerfactSim [5], a large scale simulator for p2p systems. The protocol settings, e.g. the maximum number of outgoing connections, the maximum peer set size and the number of peers initially returned by the tracker, are adjusted to the default values to be as close to reality as possible.

BitTorrent uses TCP Streams to transfer data between two peers; therefore, in order to reflect the underlying network topology, several gigabytes of internet measurement data derived from the CAIDA [4] and PingER [6] project are integrated into the core simulation framework of PeerfactSim. This allows us to realistically model concurrent and competing TCP streams, and thus to capture the real world behavior of BitTorrent. In addition to this, the integration of the measurement data also allows us to realistically model round-trip-times of overlay messages, packet loss probabilities, and inter packet delay variations (also known as *jitter*) for inter, and intra country, and regional connections from all over the world.

The use of simulation as a means of evaluation enables to study particular situations reflecting different use case scenarios that would be hard to realize in real word experiments.

3.2 Metrics

The *performance* criterion refers to the download time of users. In particular, we analyze in detail:

- *Mean download time.* This metric specifies the average time it takes for each user to download a demanded file. We compare the mean download time to a simulated optimum in which users are able to fully utilize their download within the entire download process.

  Absolute download finish time. This metric specifies the absolute point in time at which each peer's download finishes relatively to the start of the scenario at time point zero. E.g. if a peer joins the network at time point 30 and his download takes 60 minutes, the absolute download finish time would be 90. By using this metric, we are able to quantify correlations between the download times and the network joining times.

The *costs* criterion stands for the effort measured in uploaded data that both the users and the source server rented by content providers have to make in order to spread a file. We differentiate between the following two metrics:

- *Load on the source server.* This is defined as the amount of uploaded data (e.g. in GB's) provided by the source server. In our presentation, we divide this load by the aggregated upload data of the whole network. Thus, we are able quantify the fraction of data sent out by the source server.
- *Load on the peers.* This is defined by total amount of data uploaded by the users relative to the aggregate upload data of the whole network. This metric directly depends on the previous one. That is, the more load carried by the source server, the less data uploaded by the peers, and vice versa. However, it is also important to consider the load-balancing in terms of provided upload data between the users, e.g., do a few peers carry the whole load or is it evenly distributed between all participants.

3.3 Scenarios

The following two use case scenarios were evaluated in this work. In each scenario, we used the default settings as stated below, although we do vary these settings to reflect different situations of interest as noted later on.

1) Scenario "Flash Crowd": In this scenario, a popular software update is made available by a content provider. The basic setups has the following settings:

- File size: 100 MB
- Number of source servers: 1 (which stays on throughout the duration of the experiment)
- Source server upload bandwidth: 1000 Kbps
- Peer download/upload bandwidth: 1000/128 Kbps
- Join/leave process: A flash crowd where nodes join the network in 40-second intervals. Each peer leaves the network as soon as he finishes the download
- Duration of Experiment: 1 hour of joining, and then until all peers have finished their downloads
- Number of peers that join the system: 100

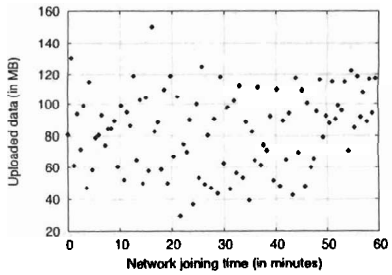
Additionally to this basic setup, we varied specific parameters within the different simulation runs according to Tab. 1.

Variation	Description
Slow seed	Up-/download bandwidth of the source server is halved
Altruistic clients	Peers stay online 30 minutes after they have finished their downloads
Small file	File size is 20 MB instead of 100 MB

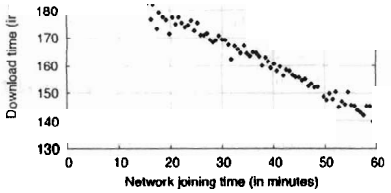
Table 1. Variations used in our simulations for the "Flash Crowd" scenario.

2) Scenario "Constant Stream": This scenario reflects a particular situation in which a demanded file is stored on a web server among several other files, thus constituting a file archive. For this reason, only a limited upload bandwidth of the source server can be reserved by the content provider for this particular file. The basic setup of this scenario has the following settings:

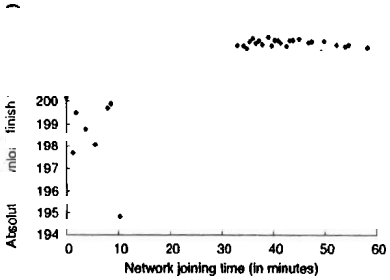
- File size: 40 MB
- Number of source servers: 1 (which stays on throughout the duration of the experiment)
- Source server upload bandwidth: 256 Kbps
- Peer download/upload bandwidth: 1000/128 Kbps



(a) The amount of uploaded data of each peer dependent on the network joining time (*basic setup*).



(b) The download time of each peer dependent on the network joining time (*basic setup*).



(c) The point in time at which each peers download finishes dependent on the network joining time (*variation "altruistic clients"*).

Figure 1. Results of the "Flash Crowd" scenario

- Join/leave process: Peers are joining the network in 2-minute intervals. Each peer leaves the network as soon as he finishes the download
- Duration of Experiment: 24 hours of joining, and then until all peers have finished their downloads
- Number of peers that join the system: 2880

Also in this scenario, we varied specific parameters according to Tab. 2.

Variation	Description
Slow seed	Up-/download bandwidth of the source server is halved
Altruistic clients	Peers stay online 5 minutes after they have finished their downloads

Table 2. Variations used in our simulations for the "Constant Stream" scenario.

4 Results

4.1 Scenario "Flash Crowd"

The outcomes of this scenario using the basic setup are as follows. The source server has to carry 15% of the total amount of uploaded data in the whole network (cf. Tab. 3). The remaining 85% is carried by the leeching peers. Fig. 1(a) depicts the average amount of bytes uploaded by each peer, depending on the time he joined the network. It can be seen that there are large fluctuations in the amount of uploaded data of each peer. Some peers only provide 40 MB whereas a small group of peers provide 120 MB. Keeping in mind that all peers have the same bandwidth capacities, we conclude that the load balancing of BitTorrent in this scenario is far from the optimum in which the load would be evenly distributed between all clients. Moreover, the graph also suggests that there is no dependency between the joining time of a peer and the amount of upload he provides.

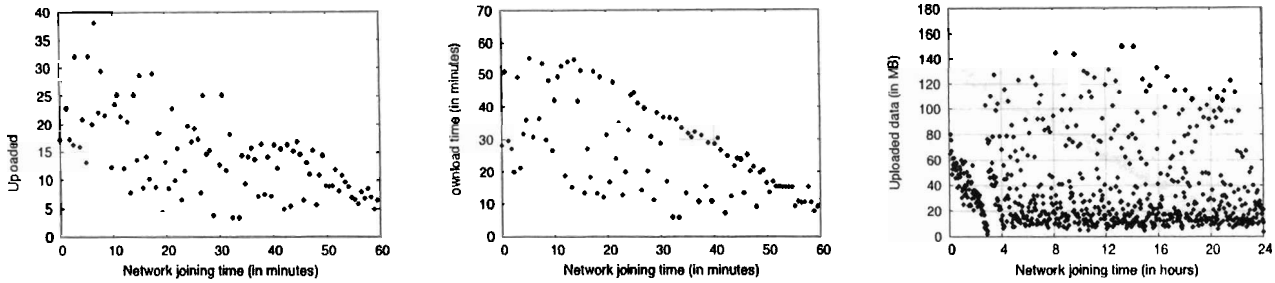
The analysis of the download times show that the later a peer joins the network, the less time he needs to receive the complete file (cf. Fig. 1(b)). This observation seems to be logical since the more time proceeds the more potential seeding peers are available in the network. Peers joining later into the network benefit from this situation. The simulations showed further that the absolute download finish times of all leeching peers are close together[1]. That is, irrespective of the network joining times, all peers finished their downloads nearly at the same point in time.

In conclusion, the content server has to carry only a seventh part (15%) of the load he would have to carry without BitTorrent! On the other hand, as seen in Tab. 3, the mean download time in this scenario was 169 minutes. If we compare this time, with the theoretical optimum of 13 minutes which is achieved if each peer is able to fully utilize its download bandwidth, the leeching peers have to wait on average thirteen times longer compared to this optimal download time.

1) Variation "Slow Seed". The bisection of the source server's bandwidth capacity leads to surprising results. On the one hand, the aggregated upload data that the source server has to carry is further reduced to 12% (cf. Tab. 3). However, the download times are now twice as long as compared to the basic setup of this scenario. Our simulations showed that in the worst case, some peers need 6 hours to finish their download. Furthermore, the simulations revealed that the fluctuations of the amount of upload data each peer has to carry on average ranges now between 25MB and 200 MB. Thus, the ratio of the upper and lower bounds grows from 1:3 to 1:8 when compared to the basic setup before. We conclude that the upload capacity of the source server has a high impact on the overall performance of BitTorrent in a flash crowd scenario.

2) Variation "Altruistic Clients". In this setup, the leeching peers stay online 30 minutes after they have finished their downloads. One might suggest that this circumstance

[1]The correspondent graphs are not shown due to space constraints.



(a) The amount of uploaded data of each peer dependent on the network joining time in the flash crowd scenario (*variation "small file"*).

(b) The download time of each peer dependent on the network joining time in the flash crowd scenario (*variation "small file"*).

(c) The amount of uploaded data of each peer dependent on the network joining time in the constant stream scenario (*basic setup*).

Figure 2. Results of the "Flash Crowd" and "Constant Stream" scenario.

would boost the download times. However, the measured results show the opposite (cf. Tab. 3). Neither the mean download times nor the amount of aggregated upload carried by the source server changes significantly compared to the basic setup. In addition to this, Fig. 1(c) confirms again the observation already mentioned in the basic setup, that nearly all peers finish at the same point in time. Our simulations showed further that the download times only decreases for peers joining at the end of the simulated scenario, and that this was not even a significant decrease, compared to the basic setup. Thus, if we compare the effort to encourage users to stay online 30 minutes after their downloads have completed, we have to conclude that it does not pay off in a flash crowd.

	Basic setup	**Slow seed**	**Altruistic clients**	**Small file**
Duration (hours)	3,5	6	3,4	1,2
File size (MB)	100	100	100	20
Bytes sent (GB)	10	10	10	2
% from server	15%	12%	13%	29%
% from peers	85%	88%	87%	71%
Avg. download time (minutes)	169	332	166	27

Table 3. Overview about results ("Flash Crowd" scenario).

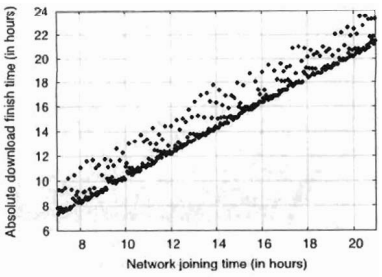
3) Variation "Small File". In this scenario, the leeching peers tried to download a relatively small file, amounting to 20 MB. The results show interesting new insights into the performance of BitTorrent. First, the load on the source server in terms of provided upload data was as twice as that of the basic setup. That is, the seed has to carry about 29% of the aggregated amount of uploaded data (see Tab. 3). Furthermore, peers joining at the beginning of the simulated flash crowd scenario have to bear a high upload burden sometimes reaching over 30 MB (cf. Fig. 2(a)). In contrast, peers joining at the end only have to carry around 5-10 MB. Hence, it can be observed that the load-balancing of BitTorrent in terms of provided upload data is bad for small files. Second, Fig. 2(b) shows very high fluctuations in the download times of the leeching peers, ranging from 5 minutes to 1 hour. Remarkably, there is a clear trend that peers joining at the end of the simulated scenario never need more than 20 minutes to finish the download. In contrast, some peers, joining at the beginning of the scenario, needed an average of 45 minutes to receive the file.

In summary, we conclude that BitTorrent performs suboptimally for small file sizes. Peers joining at the end of the flash crowd scenario have, on the one hand, only to carry a very limited amount of the total upload burden, but benefit from very low download times; whereas peers joining at the beginning suffer from both high download times and the major upload burden.

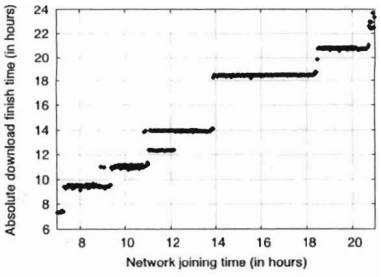
4.2 Scenario "Constant Stream"

The results of the constant stream scenario in its basic configuration, derived from a measurement period of 24 hours, showed that the source server has to carry only a very limited load, namely 8% of the total amount of upload data (cf. Tab. 4). Fig. 2(c) depicts the amount of uploaded data of each peer. It can be seen that many of the leeching peers contribute around 10-30 MB, while a small fraction has to bear the brunt of the load, around 150MB. Compared to the previous scenario (flash crowd), we observe a much better load-balancing in terms of send data by each peer.

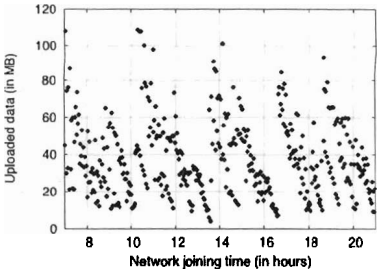
Also, the analysis of the download times offers good results. The optimum time by fully utilizing the download bandwidth would be 5 minutes, which no peer has achieved. However, the simulation showed that some of them managed to finish their downloads in 10 minutes, whereas the measured average was 29 minutes (see Tab. 4). Further, Fig. 3(a) shows a linear increase of the absolute download finish times of the leeching peers. This indicates that the majority need the same time to download the file; only some of them suffer from a higher deviation of the average.



(a) The point in time at which each peers download finishes dependent on the network joining time (*basic setup*).



(b) The point in time at which each peers download finishes dependent on the network joining time (*variation "altruistic clients"*).



(c) The amount of uploaded data of each peer dependent on the network joining time (*variation "altruistic clients"*).

Figure 3. Results of the "Constant Stream" scenario.

We conclude that BitTorrent is very appropriate in this use case scenario. First, it efficiently leverages the available upload bandwidth of the leeching peers, and at the same time it provides download times being partly close to the theoretical optimum of 5 minutes. Only a very small fraction suffers from the unbalanced load distribution in terms of provided upload bandwidth.

	Basic setup	Slow seed	Altruistic clients
Duration (hours)	24	24	24
File size (MB)	40	40	40
Bytes sent (GB)	28,8	28,8	28,8
% from server	8%	5%	6%
% from peers	92%	95%	94%
Avg. download time (minutes)	29	53	27

Table 4. Overview about results ("Constant Stream" scenario).

1) Variation "Altruistic Clients". In this variation the peers stayed online five minutes after finishing their downloads. At first glance, this slight variation should not strongly affect the outcomes of the measurements; however, Fig. 3(b) and Fig. 3(c) give some interesting insights.

From Fig. 3(b), it can be seen that, in terms of absolute download finish times, the equilibrium is heavily disturbed compared to Fig. 3(a). In particular, the peers cluster together in small groups and finish their downloads at nearly the same point in time. Peers which are online in the beginning of each cluster formation have to wait the longest to finish their downloads and have additionally to carry the highest amount of upload data (cf. Fig. 3(c)). On the other hand, peers that are joining at the end of a cluster formation finish their downloads in a nearly optimal time (5-10 minutes) and provide only very little upload data.

In conclusion, content providers can only reduce 2% of the costs in terms of provided upload data when encouraging users to stay online for a short time period (five minutes in this variation) after they have finished the download. Moreover, the average completion time to download the file does not significantly decrease (cf. Tab. 4).

2) Variation "Slow Seed". Also, the experiments in the constant stream scenario have shown that the bisection of the source server's upload bandwidth capacity to only 128 kbps leads to a negative impact on the overall performance of BitTorrent. The results reflect the same picture as already observed in the flash crowd scenario (cf. Tab. 3 and Tab. 4), except in that the load on the source server decreases. Our simulations also showed that download completion times dramatically fluctuate. The results ranged from 10 minutes to 5 hours. Hence, these outcomes act again as an indicator that the bandwidth of the source server has to be well-proportioned in order to achieve an optimal overall performance in BitTorrent.

5 Discussion and Remarks

The simulations showed that the decrease of the source server's upload bandwidth had a highly negative impact on the overall performance of BitTorrent. More precisely, in both use case scenarios, the bisection of the content servers upload bandwidth lead to mean download completion times that were double the norm, and to a significant imbalance in the amount of upload data provided for each peer. In contrast to this, the load on the source server was only reduced by 3% when compared to the basic setup variation of both scenarios. We conclude that the reduction of the source's upload bandwidth is only somewhat recommendable as it can highly impair the overall performance of BitTorrent. A further interesting insight is that the intention to encourage users to stay online after they have finished their download, does not significantly boost the mean download completion times, even though the load on the source server is only reduced by 3-4% compared to the basic setup of both use case scenarios. If we compare the effort to encourage users

to stay online for a further amount of time, we have to conclude that it does not pay off for either the content providers, in terms of save costs, or the leeching peers, in terms of decreased download times.

To answer the question whether BitTorrent is also appropriate for small sized files, we studied the flash crowd scenario with two different file size variations (100 MB / 20 MB). In the variation with 20 MB, the simulations showed an increase from 15% to 29% in the load on the source server compared to the variation with 100 MB. The results further showed a high imbalance in terms of provided upload data of each peer. In particular, the peers joining at the end of the flash crowd scenario have only to carry a very limited amount of the total upload burden, but get to benefit from very low download times. For this reason, BitTorrent performs sub-optimally for small file sizes.

In general, it can be said that the load on the source server in all variations of both use case scenarios never exceeded the 15% threshold, except the small size variation as mentioned before. The analyses of the constant stream scenario, in which the load never exceeded 8%, particularly emphasized that BitTorrent is a very nice alternative to client/server approaches. Further, in all scenarios, we observed an imbalance in the amount of upload data each peer had to carry. A few peers always had the bad luck to carry a major burden of it, irrespective of their joining time.

6 Related Work

Recently, a lot of research has been done in order to analyze the performance of BitTorrent. Most of it is only focusing on specific algorithms of the protocol; however, there are no conclusions about the overall effectiveness of the protocol.

In [11] [12], the main functionality of both the rarest-first and choking algorithm are under study. The authors find that the rarest-first algorithm is crucial for a high diversity of the pieces. They also point out that the choke algorithm in its last version is fair, fosters reciprocation and is robust against free-riders. In [8], different mechanisms to improve the performance of BitTorrent are proposed. They observe that BitTorrent's rate based tit-for-tat mechanism does not prevent unfairness in user populations having heterogeneous bandwidth. That is, end users with higher upload link capacities cluster together and are therefore able to finish downloads in less time. In [10], the authors find that BitTorrent lacks fairness. More precisely, they claim that neither contributing nodes are properly rewarded nor free-riders are effectively punished. [12] argues that the optimistic unchoke mechanism strengthens the robustness by giving leeching peers the possibility to find seeders with high upload capacities.

7 Conclusion

In this paper, we provided a depth-in study of the overall performance of BitTorrent in two representative use case scenarios, and thereby we were able to answer the questions posed in the beginning of this work.

We find that BitTorrent efficiently leverages the available upload bandwidth of the participating peers, and therefore significantly reduces the load of the source server. However, the measured mean download times were often far away from the theoretical optimum since peers were not able to fully utilize their available download bandwidth due to the widely-spread asymmetric DSL connections.

Valuable future work should further investigate our finding that giving incentives to peers to stay online after finishing downloading does not pay off in BitTorrent.

Acknowledgement

The collaboration on this paper has been funded through the European Network of Excellence CONTENT, FP6-0384239

References

[1] Azureus. http://azureus.sourceforge.net.

[2] BitTorrent. http://www.bittorrent.com.

[3] Blizzard Downloader. http://www.worldofwarcraft.com/info/faq/-blizzarddownloader.html.

[4] Cooperative Association for Internet Data Analysis (CAIDA). Macroscopic Topology Project. http://www.caida.org/analysis-/topology/macroscopic/.

[5] PeerfactSim.KOM: A simulator for large-scale peer-to-peer networks. http://www.peerfactsim.com/.

[6] The PingER Project. http://www-iepm.slac.stanford.edu/pinger/.

[7] uTorrent. http://www.utorrent.com.

[8] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *INFOCOM*. IEEE, 2006.

[9] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[10] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *P2PECON '05: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 116–121, New York, NY, USA, 2005. ACM.

[11] Legout, Urvoy-Keller, and Michiardi. Rarest first and choke algorithms are enough. In *ACM SIGCOMM Internet Measurement Conference*, 2006.

[12] A. Legout, G. Urvoy-Keller, and P. Michiardi. Understanding bittorrent: An experimental perspective. Technical report, July 08 2005.