

# Verification of BitTorrent Seeder Utilization

Mark Siner

2 May 2007

## 1 Abstract

The BitTorrent content distribution mechanism has been tested on the Internet. It has shown great performance which has convinced many people of its effectiveness. Since it is already known that BitTorrent is an effective content distribution mechanism, the challenge is to understand it better. What is needed first is verification of the specified behavior of a BitTorrent swarm. The purpose of this paper is to examine the usage of seeders in BitTorrent swarms through controlled experimentation.

## 2 About BitTorrent

BitTorrent is a content distribution protocol that is typically used for file distribution. It uses a peer-to-peer(P2P) approach to distribute the workload and bandwidth utilization. A major difference from typical P2P approaches is that there is no "BitTorrent P2P Network". Instead of a single network for all content, the BitTorrent protocol creates P2P networks, called "swarms", centered around specific content. The actions of clients in the swarm are coordinated by a central server known as a "tracker". The tracker is not involved in downloading or uploading content.

Clients in the swarm fall into one of two classifications, "leechers" and "seeders". Leechers

are clients who have not finished downloading the content. Because of this, leechers are both downloading and uploading pieces. Seeders are clients who have all of the content have no need to download, so they are not uploading. Usually at least one client starts as a seeder because that client is the content distributor. Other clients become seeders when they have finished downloading.

The protocol does rely on standard content distribution mechanisms to get a swarm started. A client wanting to joining a swarm needs two pieces of data, a description of the content and the location of the tracker. This information is put into a "torrent" file and typically hosted on a web site. The torrent includes a division of the content into numbered "pieces". Clients in the swarm use these piece numbers as a standard with which to communicate about the content.

BitTorrent clients employ a *tit-for-tat* algorithm. Generally, this means that client A will only upload to client B as much as it will upload to client A. This provides incentives for clients to upload data to other clients, which helps distribute the workload.

## 3 Behavioral Verification

This study is not focused on the performance of the BitTorrent protocol or any specific im-

plementations. The goal is to pick a specific claim of the description of BitTorrent behavior and perform experiments in order to verify that this behavior is present. After verifying certain behaviors, users of BitTorrent can have a better understanding of what is happening and possibly use it to their advantage.

## 4 Seeder Behavior

One of the most valuable assets to a BitTorrent swarm are seeders. A swarm needs at least one copy of every content piece present among the clients to fully function. If the swarm is missing any piece, leechers will never be able to complete the download of the content. Seeders are holding all pieces of the content, which means that a swarm with at least one seeder has all pieces present. So a swarm should have at least one seeder at any time.

Since a seeder could leave at any time, a swarm needs to make full utilization of the seeders while they are part of the swarm. To accomplish this, seeders do not utilize any *tit-for-tat* algorithm. Instead, a seeder will try to utilize all available upload bandwidth ([1] page 5).

## 5 Hypothesis

This research will take a scientific approach to these tests. To do so, there must first be a prediction or hypothesis. So, focusing on seeders, the following is the hypothesis:

*A seeder in a BitTorrent swarm will fully utilize all available upload bandwidth.*

There is value in this hypothesis whether it is falsified or not. If the experiments are not able to falsify it, then users can know how a BitTorrent

swarm will treat a seeder. If the experiments are able to falsify the hypothesis, then there is a possibility that the design of the BitTorrent protocol has not been implemented in the way that it has been described.

## 6 Experimental Procedure

To conduct experiments on BitTorrent behavior, this study will require full control over all clients being measured. Some studies [3] have taken approaches of monitoring clients in wild or uncontrolled swarms. On page 2 of [3] the authors describe how BitTorrent communications protocol was "abused" in order to collect data from uncontrolled clients. That kind of approach is useful for collecting real-world performance numbers, but could be unreliable for observing individual client behavior.

The first thing that had to be picked was the BitTorrent implementation that would be used. This study specifically used the Mainline BitTorrent client and tracker implementations at version 4.4.0. The term Mainline refers to the implementation made and maintained by Bram Cohen (the creator of the BitTorrent protocol) and his company BitTorrent Inc. In the absence of an official protocol standard, the Mainline implementation serves as a reference implementation that defines the protocol. The client software is available for free and under a derivative of an Open Source license at [bittorrent.com](http://bittorrent.com).

The experiments utilize five desktop computers running Fedora Core Linux. The machines are on the same local network, but the network is *not isolated*. Obtaining an isolated network would cost too much in time and work. Furthermore, network isolation should not be needed for the tests because they are not analyzing the per-

formance of the protocol or implementation. Instead, this study is observing the decisions made by the clients of which small network interference should have minimal effect upon.

Slight alterations were made to the Mainline BitTorrent client to be used in the experiments. The change made the console client, `bittorrent-console`, create a local log file. The client would update the log file for every time it updated its output, which will be set to happen every ten seconds. Each update to the log file contained the following information; an ISO 8601 format date/time of the update, the percent of the content downloaded, and peer information. Each peer entry contained the IP address of the peer and the upload and download rates. This log file was in JavaScript Object Notation(JSON) for ease in parsing. The client would continue updating the log until the program terminated.

A script, `sshlauncher.pl`, was created to parse and run experiment configurations. An experiment can be configured in a text file using JSON. The configuration contained a list named "jobs" which contained objects of information for each job that needed to be run in the experiment. A job configuration consisted of a machine name, user name, directory to run from, command to execute, a delay in seconds to wait before executing the command, and a duration in seconds before the command should be terminated. One part of the experiment configuration that was not automated was setting up the directories for each client. This involved manually creating an empty directory for each client and the tracker and then copying the entire file to be downloaded into the directories corresponding to the clients that will start as seeders. Another script that was necessary was a script called `timedlaunch.pl` that

would wait the delay time, execute the command, and then wait the duration time before terminating it. So `sshlauncher.pl` would parse an experiment configuration, log into each machine via SSH, and then execute an instance of `timedlaunch.pl` with the delay, duration, and command as parameters. This setup allows a whole experiment to be started with the execution of a single command without any central coordination needed to make sure all of the timing was correct. Note that because the logs are timestamped with the time reported by the operating system, the machine clocks must be synchronized within at least the update period to be able to use the log data.

When an experiment ended, the next step was to collect all of the client logs. Another script, `aggregatelogs.pl`, was created to do this job. Given an experiment file, it would use `scp` to transfer each log file from the specified working directory and then concatenate it to an aggregate file, `agg_log.json`, and keep the new file also as valid JSON.

The next step to the post-experiment process is generating graphs to represent the state of the swarm. Such graphs would include an vertex for every known client and directed edges for every upload relationship. Each vertex would have a name or IP address, depending on if the name is known yet, and the percent downloaded. Each edge would go from uploader to downloader and be labeled with the upload speed reported by the uploader( notice that the graphs are not using the speed reported by the downloader as it could be affected by network traffic ). Furthermore, since this study target seeders, the seeder vertices will be in a visible named cluster. All of these graphs will be generated using Graphviz, a program used to create well layed-out graphs. To track the progress of the swarm, a graph will

need to be generated for every update of the aggregate log. The algorithm to generate all of these graphs is actually rather simple.

To generate the graphs, each update was tagged with its owner name and interleaved into a single sorted list. Besides this list, there will be an associative list that maps any known client address to its last update. The algorithms is to scan through the list and, for every update, set the update in the associative list and then generate a graph using all of the states in the associative list. As a further step, the script, `generategraphs.pl`, will also generate a small HTML page to display the image and link to the next page in the timeline.

The amount of work involved in constructing the framework was non-trivial, but not beyond reach of being recreated by any computer science researcher. Below is the breakdown in lines of code for the work done.

`bittorrent-console`: 36 lines of Python

`sshlauncher.pl`: 31 lines of Perl

`timedlaunch.pl`: 36 lines of Perl

`aggregatelogs.pl`: 50 lines of Perl

`generategraphs.pl`: 289 lines of Perl

Furthermore, all of these scripts were constructed with scalability in mind. The amount of data generated by an experiment is quite large and can result in generating around 600 graphs for a 15 minute experiment. The graphs generated by the Graphviz software are adequate for this study, but lack some capacity for representing more complex sets of data that might be needed for more in-depth research.

The above framework provides the ability to configure, execute, and analyze a complete experiment. The analysis will consist of walking

through the timeline of graphs and observing the behavior. The experiments should be configured so that an experimenter would know where they should start looking based on events configured into the experiment.

## 7 Experiments and Results

The experiments in this study will involve five client swarms. Each experiment will vary in the seeder to leecher ratio in the swarm. With always at least one seeder and one leecher. All clients will join the swarm at the same time and have an upload speed cap of 20 kB/s. The cap limits the sum of all upload speeds.

### 7.1 Experiment A

This experiment will involve a single seeder. Figure 1 shows the state of the swarm about 2 minutes after the swarm started. The upload edges to the seeder increased evenly until they hit the cap shown in this picture.

The graphs presented in this study will all have similar features. A graph is essentially a snapshot of the swarm at a moment during the experiment. Each node represents a client. These nodes have a percentage on them representing the percent of the file downloaded (note that seeders will have this set to 100%). Each edge from node A to node B represents node A uploading to node B at the rate posted on the edge in kB/s. All seeder nodes will be in a cluster labeled "seeders".

### 7.2 Experiment B

Experiment B will start the swarm with two seeders. Figure 2 shows the state of the swarm after about 2 minutes.

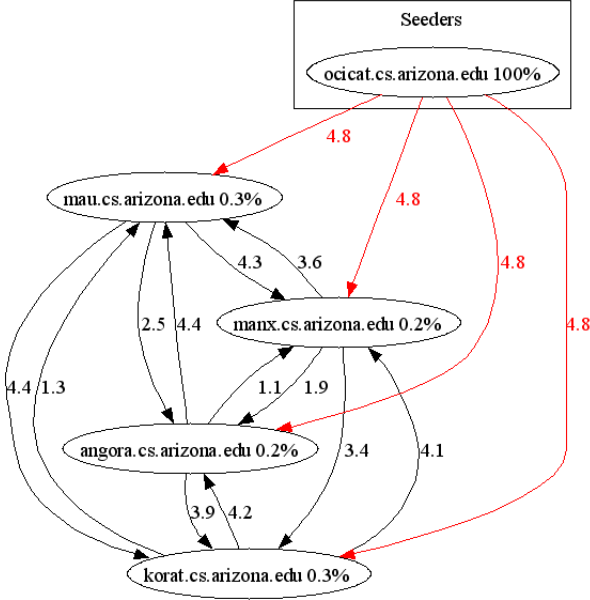


Figure 1: Experiment A swarm

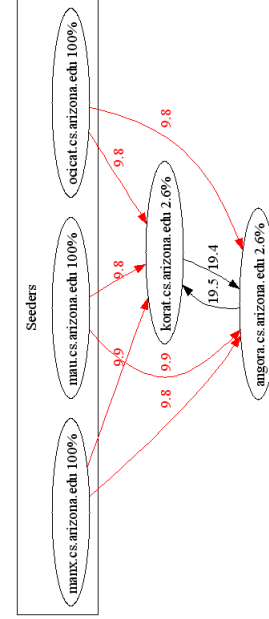


Figure 3: Experiment C swarm

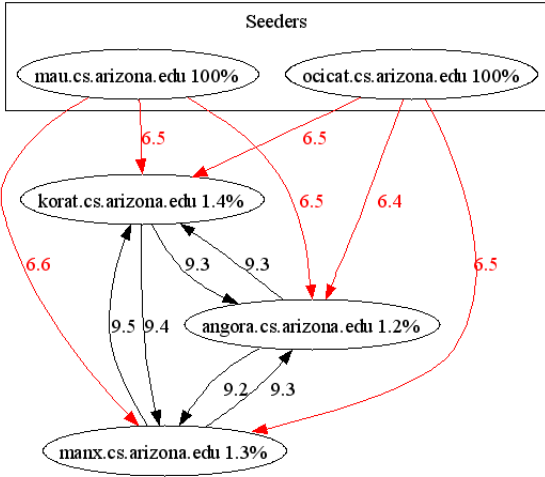


Figure 2: Experiment B swarm

### 7.3 Experiment C

This experiment will start the swarm with three seeders. Figure 3 shows the state of the swarm after about 2 minutes.

### 7.4 Experiment D

This experiment will start the swarm with four seeders. Figure 4 shows the state of the swarm after about 2 minutes.

## 8 Evaluation

### 8.1 Uniform Asymptotic Increase

One observation that was made in the experiments was a uniform increase in seeder upload

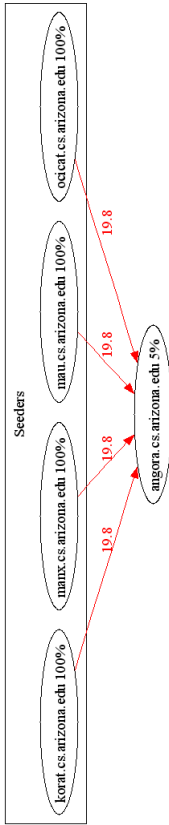


Figure 4: Experiment D swarm

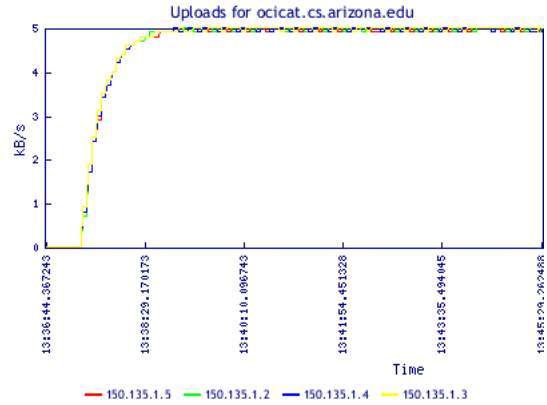


Figure 5: Seed upload speeds from Experiment A

speeds. This means the seeder seemed to increase all upload speeds equally in a balanced way. Furthermore, the speed increase looks asymptotic. Figure 5 shows the upload speeds from the single seeder in Experiment A to the other four clients. The chart shows upload speed on the y-axis and time on the x-axis with a line for the upload speed from the client to all other clients. It can be observed that all lines follow the same curve that has an upper bound of 5 kB/s.

## 8.2 Full Utilization

The graphs shown in the Results section should be self explanatory when discussing the hypothesis. Each seeder in the graphs is shown utilizing its full bandwidth capabilities as dictated by the 20 kB/s restriction. The seeders remained in these states until the leechers finished downloading the content and became seeders themselves. Because of the uniform distribution described in the previous section, this happened with all of the leechers near simultaneously. Fig-

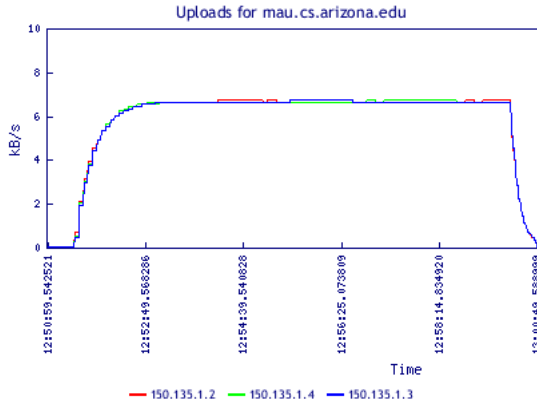


Figure 6: Seed upload speeds from Experiment B

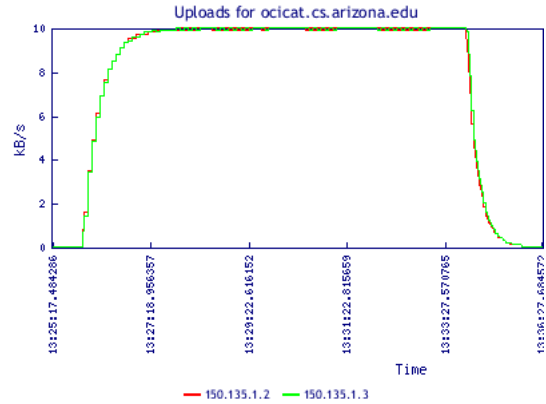


Figure 7: Seed upload speeds from Experiment C

Figure 6, Figure 7, and Figure 8 show this dropoff occurring. All of these charts have a span of 15 minutes. Logically, the more seeders involved in the swarm, the sooner this dropoff happens. The dropoff did not occur in Experiment A because none of the leechers finished downloading in the duration of the experiment.

## 9 Implications

Now that it has been observed that seeders will always be fully utilized, what can be done with this information. It is the belief of this author that this information is essential knowledge for any content distributors using BitTorrent or considering using it as a distribution mechanism. When using BitTorrent, a distributor should plan on capping its seeding clients according to the bandwidth and number of servers available. At a bare minimum, BitTorrent can provide an efficient load balancing when using multiple servers as opposed to using traditional direct hosting and downloading. The problem is,

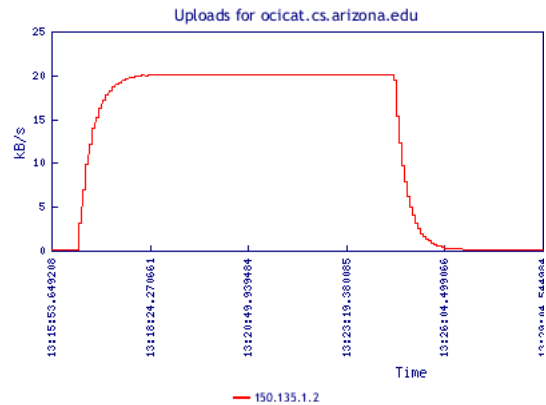


Figure 8: Seed upload speeds from Experiment D

when the swarm is large, the seeding servers will still be utilizing all of their bandwidth. So how can distributors offload the work to the swarm so that the seeding servers will be used only when they are needed? A possible solution that has been proposed is called Super-Seeding that was proposed by the developers of the BitTorrent client in [2]. The Super-Seed mode is not part of the mainline client, but it is being implemented in other clients besides BitTorrent. What Super-Seeding does is makes a seeder hide the fact that it is a seeder. It will advertise rare pieces and not advertise any more pieces until it sees that other clients are sharing the piece that it just shared. This serves the purpose of creating seeders faster with the consequence of slower download speeds until seeders are established.

## 10 Future Work

This study has been only preliminary research into the different aspects of the BitTorrent protocol. More work needs to be done to understand the behavior of swarms. Of particular interest is the relations between leechers that is governed by various choking algorithms. Of note in the experiments is the leecher upload relation in Experiment A. Figure 9 shows the upload speeds from a leecher from Experiment A, while Figure 10 shows the upload speeds from a leecher in Experiment B. Each of these results was typical for leechers in the corresponding experiment, but they are wildly different from each other. The behavior in Experiment A seems to suggest influence from the choking algorithm, but that influence does not seem to exist in Experiment B.

There are two logical next steps for this specific work. The first is to experiment using different BitTorrent clients that do have common

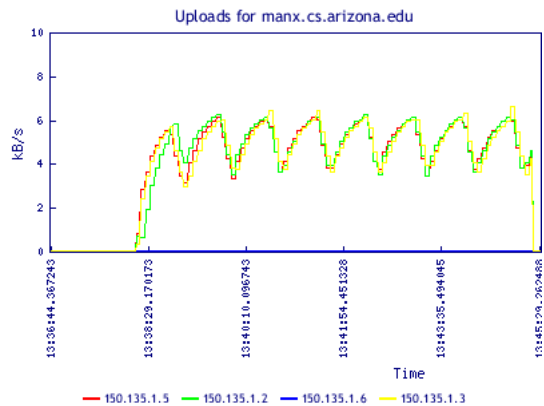


Figure 9: Leecher upload speeds from Experiment A

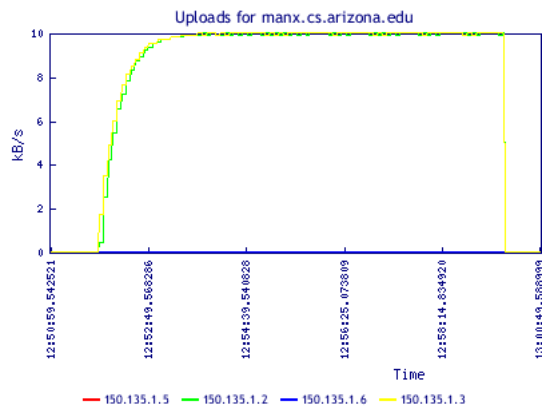


Figure 10: Leecher upload speeds from Experiment B

ancestry (many BitTorrent clients are derivatives of the mainline client). This approach has the possibility of revealing inconsistencies between implementations that could lead to poor swarm performance. Such an inconsistency could be caused by either poor implementation or vagueness in the de facto standard.

The other approach is to investigate Super-Seeding. Because Mainline does not support this feature, another client such as BitTornado would need to be used. Running the same tests using Super-Seeding should give us different results and hopefully lead to lower bandwidth demand being put onto the initial seeders.

## 11 Conclusions

This study has served its purpose. That is, it has not falsified a key proposed behavior in BitTorrent swarms. It serves as evidence that a BitTorrent swarm will fully utilize seeder bandwidth. More research needs to be conducted to verify BitTorrent behavior so that it can be used as a reliable and predictable content distribution mechanism.

## References

- [1] B. Cohen. Incentives build robustness in bit-torrent. In *Proc. of Workshop on Economics of Peer-to-Peer Systems*, May 2003.
- [2] BitTornado Developers. About super-seed mode. Algorithm description published on the BitTornado website [bittornado.org](http://bittornado.org) at [bittornado.org/docs/superseed.txt](http://bittornado.org/docs/superseed.txt).
- [3] Garbacki P. Epema D. Pouwelse, J. and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proc. of*

*the 4th International Workshop on Peer-to-Peer Systems*, February 2005.