Summer 2011

Improving Performance of BitTorrent Network through Incentive Mechanism

Mingzhe Li
San Jose State University

Improving Performance of BitTorrent Network through Incentive Mechanism

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Mingzhe Li

Spring 2011

Abstract

Content Distribution via Internet has become increasingly popular right now. Compared with centralized file distribution system using FTP and HTTP protocols, P2P (Peer to Peer) is more cost-effective. Furthermore, it could help save on bandwidth costs and to handle pick demands. Among many P2P protocols, BitTorrent is one of the most popular protocols right now. The BitTorrent network uses tit-for-tat as a method of seeking Pareto efficiency. However, seeders who contribute more to BitTorrent network than leechers are not incentivized to stay online and upload to others. This paper discusses an incentive mechanism which rewards seeder, who stays in a BitTorrent network and uploads to other peers, with better download speed in its further download process. Experimental results with the incentive mechanism and an analysis of the result are also discussed in this paper.

Acknowledgements

I thank my advisor, Dr. Robert Chun, whose support and dedication guide me finish the writing project. Dr. Chun is always there to listen and to give technical and editorial advice. I also thank him for giving me the freedom to choose my own research topics.

A very special thank to my committee member, Dr. Xiao Su, who gives me the access to PlanetLab. I also like to thank Dr. T.Y. Lin for participation as my thesis member and a role for me being lively, enthusiastic, and energetic.

It has been a challenging, yet rewarding journey which I could not have completed alone and am grateful for your support.

Thank you.

Table of Contents

List of Tables

List of Figures

1.0 Introduction

This section argues the advantages of P2P network and gives brief introductions of BitTorrent protocol. The problem with regular BitTorrent network and the solution to this problem are discussed in section 1.3.

1.1 The Case for Peer-To-Peer

Peer-to-peer network is a system with elements that both provide services to and request services from other elements. There are various areas which Peer-to-peer architecture could work better than other architectures. For example, content distribution system that uses client-server model always needs strong central servers and enormous bandwidth to ensure client's download speed. Illustrated in Figure 1, the burden on the server will increase with the number of clients because the server is sending one copy to each client.

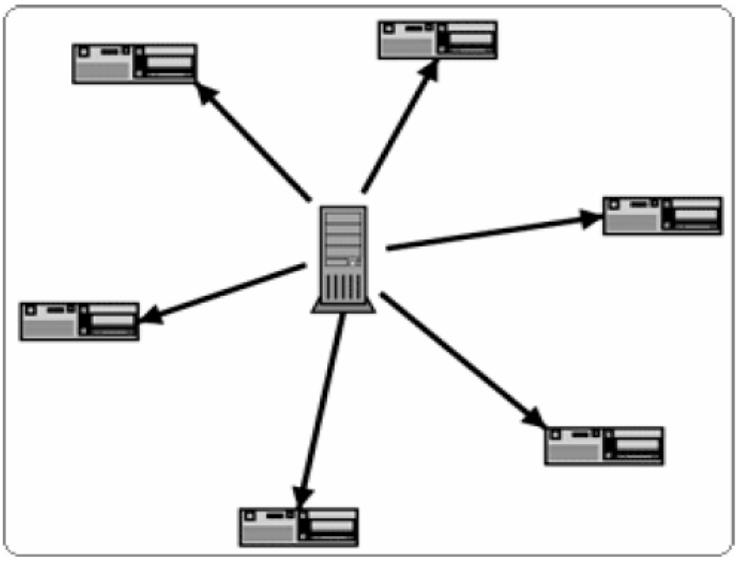

Figure 1: Centralized File Distribution System.

Figure source: (bittorrent.org 2006)

In contrary with centralized file distribution system, P2P(Peer-toPeer) file distribution system enables the network to increase indefinitely without investing costly central resource because of extra processing power and bandwidth brought by new users to the network. It enables nodes to download resources from others and upload resources to others simultaneously, illustrated in figure2.

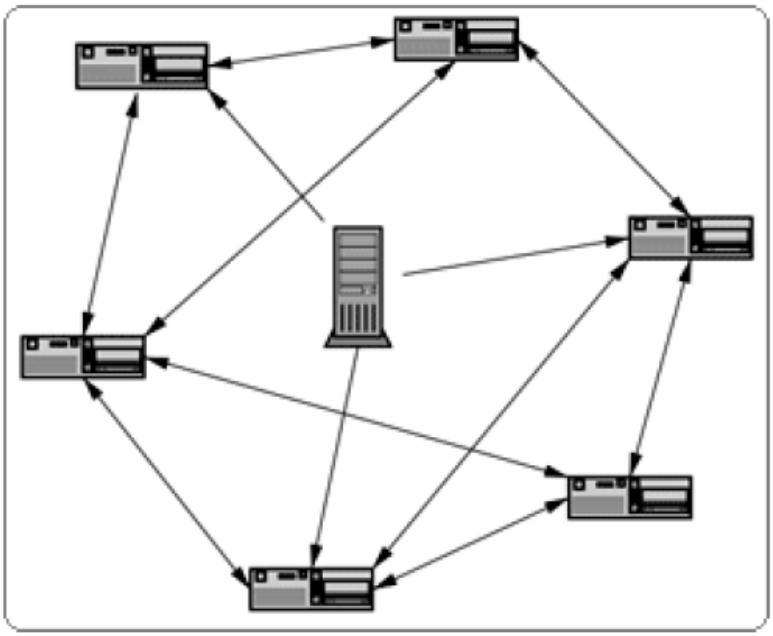

Figure 2: File Distribution System with Peer-to-Peer.

Another advantage of P2P is robustness of the network. In traditional centralized file distribution system, if one component of the server fails, the whole network will collapse. However, in P2P file distribution system, if components of nodes fail, the whole network is still robust.

1.2 BitTorrent Introduction

BitTorrent is a peer-to-peer file sharing protocol used for distributing large amount of data. It is one of most commonly used protocols for transferring large files,

and it has been estimated that it accounted for 27% to 55% of all Internet traffic as of February 2009[1]. One major advantage of BitTorrent protocol is distributing large files without adding heavy load on the source computer and network.

The basic idea of BitTorrent is to divide the file into equal-sized blocks and have nodes download blocks from multiple peers concurrently [2]. Nodes in BitTorrent network are either seeder or leecher. A seeder refers to a client that uploads to its peers after it has completed its download. A leecher is a client that has not completed its download [2]. In order to establish connections between nodes in BitTorrent network, tracker, a central server of BitTorrent network that keeps record of nodes currently in the system, is necessary. An example is given to briefly introduce the complete download process in BitTorrent network. Assumed there are one seeder and three leechers in this example. A BitTorrent client AAA wants to share a novel named “Gone with the Wind” of 2 Megabyte. AAA first needs to create a .torrent file which contains metadata (assuming AAA set the piece size to be 512 KB) of the file and the tracker to use. Secondly, AAA could upload the torrent file to a public domain site which enables for others to download the torrent file. Another BitTorrent client BBB, who is interested in the novel, needs to obtain the .torrent file created by AAA from that public domain site. After BBB gets the torrent file and run on its BitTorrent client, information in the .torrent file will help establish connection between AAA and BBB, illustrated in Figure 3. Once the connection between AAA and BBB is established, BBB could start download piece1 to piece4 from AAA. Suppose another client CCC begins to download “Gone with the Wind” after BBB finishes downloading piece1. Since AAA and BBB both have piece1 of that novel at this moment, CCC could download piece1 from BBB and get piece 2, 3 and

4 from AAA. When new user DDD comes to the network, DDD could connect with AAA, BBB and CCC. Because AAA, BBB and CCC all have pieces of the novel, DDD could download four pieces from 3 different nodes. In this way, the seeder AAA's network and performance will not decrease as a result of the increasing number of users. If more users join the network, the download completion time will decrease significantly. Compared to traditional Internet hosting, BitTorrent network reduces great burden imposed on the original distributor's hardware and bandwidth.

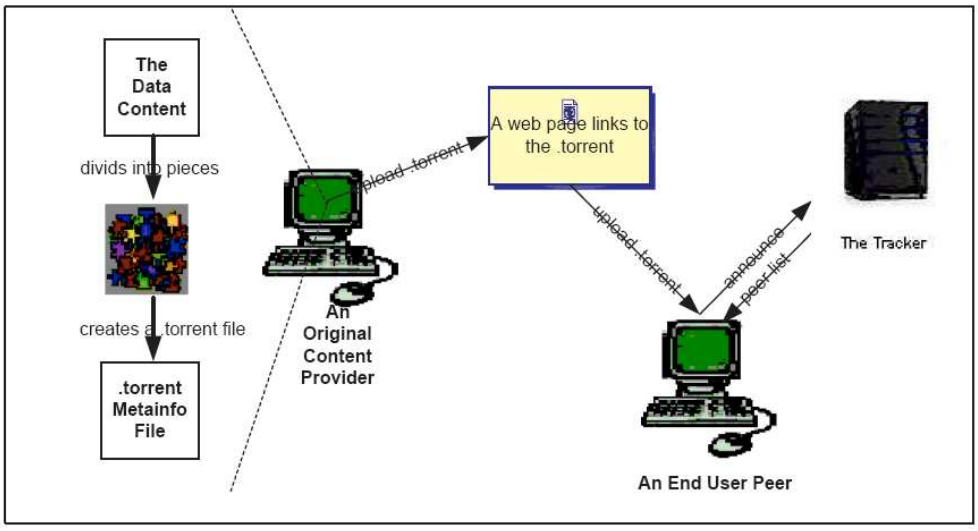

Figure 3: BitTorrent Download Process. Source [3]

1.3 Problem and Solution

In contrast with traditional file distribution techniques' enduring availability, a source for the file shared in a BitTorrent network is generally temporary and therefore harder to trace. To illustrate the problem, let's consider the previous section's example. Assuming BBB is the only seeder who completed downloading the novel "Gone with the Wind". If AAA is not in the network anymore and BBB exits the network right after it

finishes downloading, other clients will have no other choice but to wait for BBB's return to the network to finish downloading. This is one of challenges of BitTorrent network. Because of lacking incentives for seeders to upload resources for others, BitTorrent seeders choose to exit the network right after they finish downloading.

To enable leechers in BitTorrent network completing download process, seeders need to stay online and share resources. If there are more seeders available in the network, leecher's download completion time will be decreased significantly. In order to solve this problem, we propose to add an incentive mechanism for seeders. Not only will the incentive mechanism incentivize seeder to stay online, but it will also increase the performance of the whole network. This paper discusses the design, implementation of the incentive mechanism. Furthermore, we conducted experiments on PlanetLab to test performance of the incentive mechanism.

2.0 BitTorrent Background

This section describes known algorithms for BitTorrent system to work efficiently. A description of piece selection mechanism, peer selection mechanism, and pipelining is presented. Moreover, two limitations of current BitTorrent system are discussed at the end of this section.

2.1 Overview of Piece Selection Mechanism

BitTorrent protocol transfers files piece by piece. Different from traditional network protocols which download files from start to end, BitTorrent first allocates space for a download process and then gets pieces from peers. An inefficient piece selection

mechanism can lead to low performance of the whole BitTorrent network. For example, if a leecher is not interested in pieces its associated nodes have to offer, this leecher could stop uploading any piece to its associated nodes. If most of the nodes in a BitTorrent network stop uploading any piece to others, the whole network will collapse. So, an efficient piece selection mechanism not only helps the system work properly but also enables it to reach its optimal performance.

In a download process of BitTorrent network, there are three different stages. BitTorrent implies three separate algorithms in these stages. These three algorithms are illustrated in Figure 4.

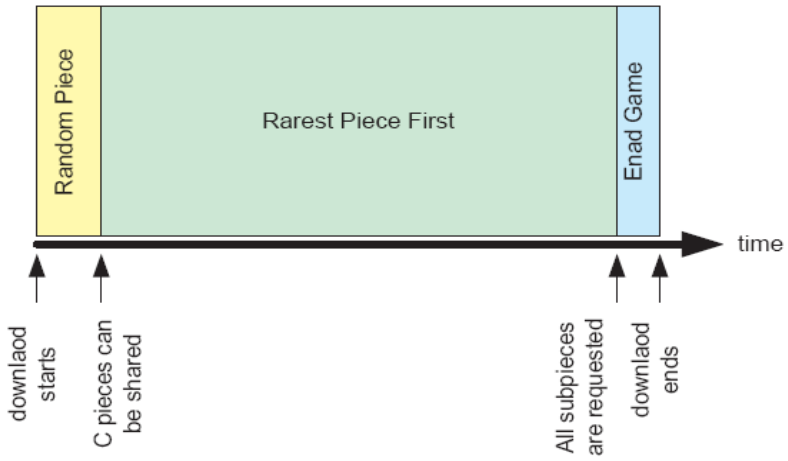

Figure 4: Mechanism to Select Piece

2.2 Rarest Piece First

Rarest piece is the least amount of copies of one particular piece in a swarm of nodes [3]. Nodes in a downloading process store a table containing a list of pieces existing in the swarm and the number of each piece. With the rarest piece first algorithm, the next piece for a peer to download is always the rarest missing piece. In some cases,

there will be more than one piece which is the rarest piece. If this happens, a random piece for those will be selected by the peer to download.

Advantages of rarest piece first algorithm are:

1. Each peer always has piece that its associated peers want to download.
2. Increase the possibility of finishing downloading all pieces by starting downloading the rarest piece first. Furthermore, if only one seeder is available for system, this algorithm is necessary for download completion.

2.3 Random Piece First

When a new peer starts a download process, it should get pieces as soon as possible in order to reciprocate for others. As to get pieces faster, peers should randomly select pieces to download. Compared with rarest first piece algorithm, random piece first algorithm that downloads from more than one peer at the same time is more likely to download faster. After it has downloaded I pieces(I is a constant which is different in different BitTorrent client implementations), it will stop using the random piece first algorithm and start to apply the rarest first strategy.

2.4 End Game

Sometimes a peer will download a piece at a very low download rate. If this happens in the middle of a download process, it's not a problem. However, if it happens at the end of a download process, it will delay the process of finishing downloading. BitTorrent prevents this condition by applying end game piece selection algorithm. This algorithm is applied once a peer has requested every piece of the file. In that situation, the

peer will send requests for all sub pieces which are missing to all its peers who have those corresponding sub pieces. If a sub piece is downloaded in the end game phase, the peer sends “cancel” messages to its associated peers that have the corresponding appending requests. In this way, bandwidth could be saved from redundant sends.

Advantages of end game algorithm are:

1. Downloading the end of the file at a faster rate
2. The possibility of finishing a download process is increased

2.5 Pipelining

Normally, HTTP requests are issued sequentially, with the next request being issued only after the response to previous requests have been received. Depending on the network latencies and bandwidth limitations, this can result in significant delay before the next request is seen by the server[2]. Pipelining enables HTTP requests to be written out to a single socket without waiting for the corresponding responses, illustrated in Figure 5.

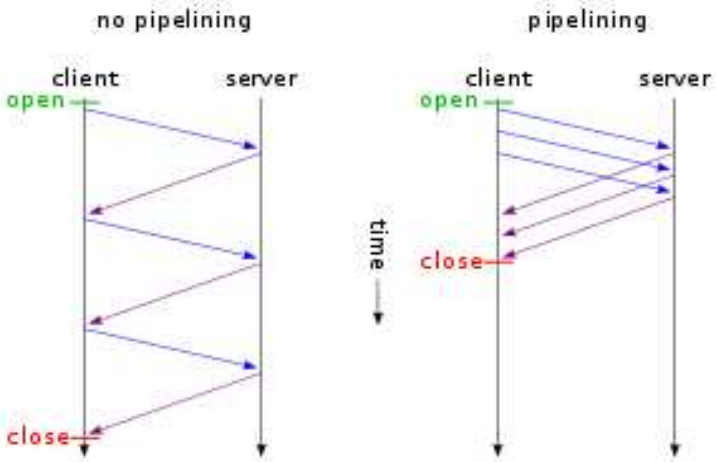

Figure 5: Schema of Non-Pipelined Connection VS Pipelined Connection

BitTorrent which depends on TCP enables the mechanism of Pipelining. In order to maximize the advantage of pipelining, BitTorrent protocol breaks down pieces into sub pieces the size of which is from 32KB to 1MB. In order to decrease the latency between pieces in a BitTorrent download process, a peer always has five requests pending at one time. In this way, the peer could send out five requests for sub pieces simultaneously. Furthermore, in a good network environment, all five sub pieces will be returned sequentially. With pipelining, BitTorrent helps peers decrease download completion time significantly.

2.6 Strict Priority

Different from the previous three algorithms which are used at separate phases, strict priority is used from the beginning of a download process to its end. BitTorrent applies the strict priority policy for sub piece selection. As we have explained in the pipelining, when BitTorrent protocol transfers pieces over the network, it breaks pieces into sub pieces. Strict Priority is that once a sub piece of a piece has been requested, the remaining sub pieces of that piece will be requested before sub-pieces from any other piece [2]. With this mechanism, a peer could always get the copy of a piece as soon as possible. Furthermore, it's the crucial algorithm which enables rarest piece first, random piece first and end game piece selection working efficiently to finish downloading.

2.7 Seeder's Peer Selection

Different from peers who upload to and download from others, who only upload content to others apply different peer selection algorithms. Generally, seeds try to distribute data uniformly to peers, so they imply the algorithm in a round-robin manner.

2.8 Tit-for-Tat

In BitTorrent protocol, peers download from whom they can, and upload simultaneously to a constant number of peers. Because of TCP congestion control behavior which prevent sending data over too many connections at the same time, a peer only uploads to a limited number of associated nodes. Thus, peers need to select peers from a swarm of peers to upload to. In BitTorrent, this behavior is named unchoke. The default number of peers to unchoke is four. A peer makes the decision regarding whom to unchoke and whom to choke every 10 seconds. Since each peer needs to follow the choke algorithm, an efficient choking algorithm will influence the performance of the whole system.

BitTorrent incorporates a tit-for-tat(TFT) peer selection mechanism which nodes preferentially upload to peers from whom they are able to download at a fast rate in return[2]. Although seeds don't download anything, they follow the same algorithm policy which enables them to upload to up to 5 nodes that have the highest download rate.

Advantages of this mechanism are:

1. Motivate peers to contribute to the network.
2. Utilize all available bandwidth of the system.

2.9 Optimistic Unchoke

Tit-for-Tat works well in the middle of downloading. But, when a peer starts to download, it doesn't have anything to upload to others. Based on the Tit-for-Tat algorithm, the new peer will never be unchoked. In order to avoid this problem,

optimistic unchoke mechanism is applied in BitTorrent protocol to work together with tit-for-tat algorithm. Optimistic unchoke mechanism could boost a new peer who does not have any piece of content. This mechanism chooses to unchoke a peer randomly regardless of its current upload rate. It is applied every optimistic unchoke period (typically 30 seconds). Optimistic unchoke mechanism is to unchoke one connection, while tit-for-tat is to unchoke the resulting 4 connections. With the optimistic unchoke algorithm, new peers joining the BitTorrent network could get resources as soon as possible.

2.10 Anti-Snubbing

In BitTorrent, one important rule is that total download speed should be approximately equal to the total upload speed. So, each peer should be encouraged to contribute to the system and get the same amount in return. However, there are some peers who only download resources from others. To prevent this problem, BitTorrent implies the anti-snubbing mechanism.

If a peer has not received anything from a specific peer for a certain amount of time (typically 60 seconds), it will mark the particular peer's connection as snubbed. In this case, the peer will continue to get poor download speed until the optimistic unchoke mechanism finds better peers.

3.0 Related Work

This section describes known solutions to solve seeder promotion problem. A description of single torrent incentive mechanism and multi-torrent incentive mechanism is presented.

3.1 Dandelion

Dandelion is a system which provides robust (provably non-manipulable) incentives for nodes to upload to others in a paid content distribution system [18]. Based on Dandelion, a client who honestly uploads to its associated peers is rewarded with credit, which is a kind of monetary reward. A client that does not upload or uploads garbage to its peers cannot claim credit. A client cannot download resources from selfish (rational) peers without the client being charged and the peers rewarded. Based on the Dandelion system, peers are incentivized to upload to its peers even if they do not have content that interests the client. Figure 6 shows the Dandelion system:

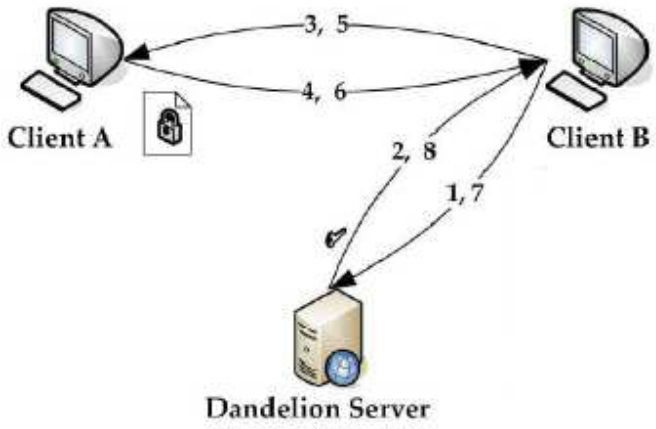

Figure 6: Dandelion System

In Figure 6, the numbers on the arrows corresponds to the download process of Dandelion system. Message in each arrow is show:

1. Request for content from server
2. Send Back a list of peers and tickets
3. Chunk Announcements
4. Request for chunk
5. Encrypted chunk, encrypted key and commitment
6. Request for decryption key
7. Decryption key

Each client's credit is managed by the Dandelion server. Also, each client has a shared symmetric key with the server. In this way, the system could prevent known attacks such as Sybil attack.

3.2 Team Incentives

Although Dandelion could incentivize peers to upload resources to others, it sacrifices the scalability of P2P system. Team-Enhanced BitTorrent protocol enables peers with similar upload bandwidth to form a team which will collaborate for mutual benefit. Figure 7 shows the steps for team formation.

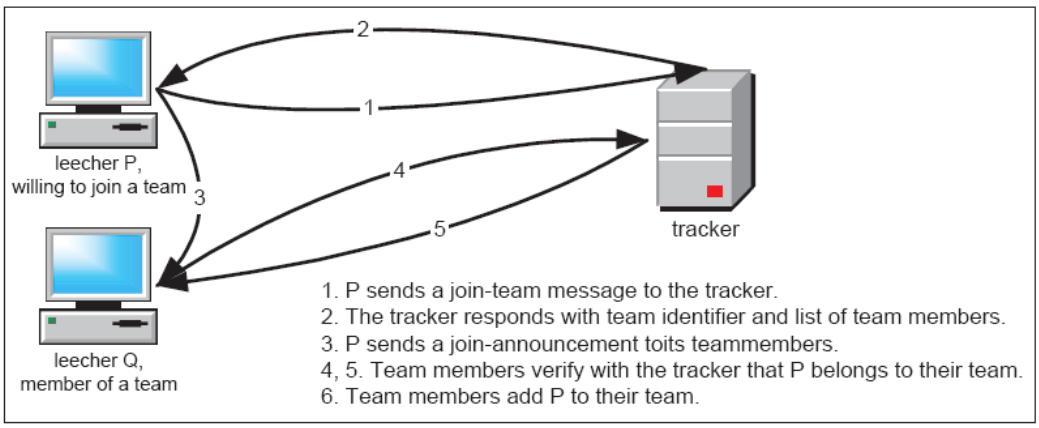

Figure 7: Team Formation

Based on team-enhanced protocol, the total number of optimistic unchoke will be reduced. Furthermore, In a team of symmetric peers, each one will make its prioirity to serve other team members as a replacement to the optimistic unchokes. In this way, a peer will get improved download rates compared to being independent.

3.3 Buddy Incentives

Team incentive mechanism could improve the performance of BitTorrent, but it requires revisions to the tracker of a BitTorrent system. Buddy incentives which adopt the similar idea as team incentive does not require revisions to the central tracker. The notion of buddy means pairs of peers having similar upload capacity, collaborating for mutual benefit. Buddy incentives could significantly reduce the number of optimistic unchokes which may force high capacity node to work with low capacity node. Figure 8 illustrates the formation of buddy incentive:

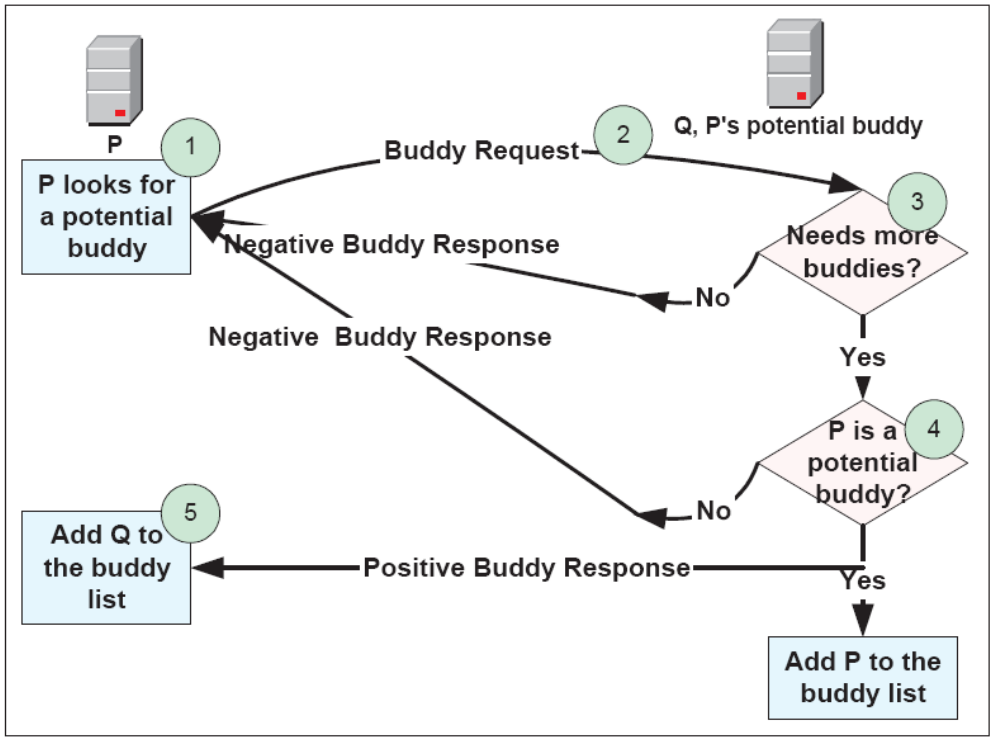

Figure 8: Buddy Formation

In Buddy incentives, a leecher P is willing to maximize the number of buddies that have a similar upload rate as itself. Leecher P reserves an unchoked slot for each buddy to which it can upload data to in order to minimize buddy chokes, which can lead to the termination of buddy relations. Based on the buddy incentives, peers will save bandwidth previously used for optimistic unchoke for their buddies. In this way, peer's download completion time will be decreased.

3.4 Multitorrent Solution

Most BT studies focus on single torrent solution, while measurements in [19] suggest that 85% of users participate in multiple torrents. In [19], it proposes a solution to solve the seeder promotion problem in a multitorrent environment. Multitorrent means

that a node participating as a leecher in a particular torrent is willing to serve as a seed in a torrent in which it has participated some time earlier in its lifetime. In [19], when node chooses peers to unchoke, it will base its choice on total contribution of each peer. Contribution of each peer is computed in formula (1): $D_i(x, Y)$ is the downloading rate of node N_x from Node N_y and $w_i(y)$ is the weight we assign to that downloading rate. If N_y is not a seeder in torrent I, then $w_i(y) = 1$, otherwise, $w_i(y)$ can be set to a value that is larger than one. Based on the formula, the node that is seeding files in one torrent file will get more contributions which are computed in the formula. Based on the multitorrent solution, more contributions mean more bandwidth. So, seeder could be incentivized to stay online and upload to others.

$$\sum_{i=1}^{\#\text{Torrents}} w_i\ (y) * D_i(x, y) \quad (1)$$

4.0 New Incentive Mechanism

This section discusses a new incentive mechanism which aims to incentivize seeders stay in the network and upload to other peers. A description of design, implementation of the other incentive mechanisms is presented.

4.1 A Problem with BitTorrent

BitTorrent protocol strives to ensure fairness: peers who contribute data to the system should be able to achieve high download throughput. However, fairness itself is not enough to enable the BitTorrent system to get the best system performance. There are always some peers who contribute more data to the system than others.

One major incentive mechanism for leechers to upload to other peers is tit-for-tat incentive mechanism which facilitates the continuous discovery of better peers. However, the TFT peer selection mechanism could not incentivize seeders uploading to others. Although the number of seeders is far less than that of leechers in a BitTorrent, a large proportion of upload bandwidth is from seeders, illustrated in Figure 9. Because of the seeder promotion problem, a massive proportion of torrents (about 40%) achieve extremely low performance with few users being able to download the file successfully. Furthermore, if more seeders are willing to upload to other peers, mean download completion time of nodes will decrease significantly. In order to incentivize more seeds to stay online after they finish downloading, we propose an incentive mechanism for seeders.

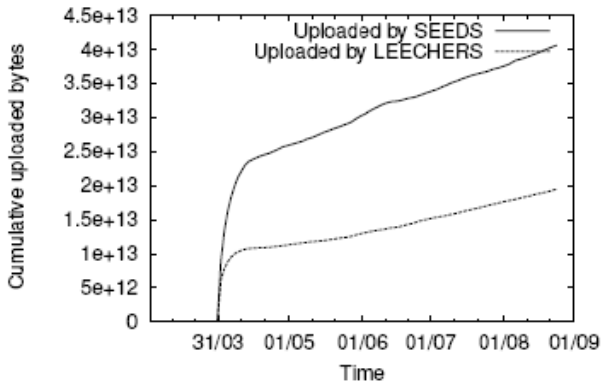

Figure 9: Volume Uploaded by Seeders and Leechers

4.2 Design

The purpose of the incentive mechanism in this paper is to incentivize seeder stay longer in the BitTorrent network. The incentive works to ensure seeders who stay longer than others in the BitTorrent network have better download speed than others when it joins another download process in the future as a leecher. Download bandwidth of a

leecher is the sum of all its associated peers' upload bandwidth, so redistributions of its associated peers' upload bandwidth could manage the leercher's download bandwidth. The mechanism consists of three components:

1. Counter component: if a seeder stays in BitTorrent network and uploads to other peers, the counter component will compue a value for the seeder based on the number of seeding files and number of minutes uploading for others.
2. Client component: it establishes TCP channels with all associated peers in the same download process and gets each one's value returned from server component. Finally, it will rearrange upload speed for each associated peer based on the value returned from each client.
3. Server component: it binds to the port 59500 and waits for future requests from associated peers' client component. When new request arrives, the server will get the value computed by counter component and send it back to requested client.

4.3 Deluge

Deluge [20] is a BitTorrent client selected to test the incentive mechanism. Advantages of deluge over other BitTorrent clients are:

1. It is a lightweight, cross-platform BitTorrent client, which makes its portable from one platform to another.
2. Deluge and its dependent library libtorrent[5] are all open source which is able for us to change the source code.
3. Most functionalities of deluge are achieved by plugins. Also, it provides complete documentation for API of deluge.

One crucial part of the incentive mechanism is to manage each peer's upload speed. However, in the latest deluge client, that function doesn't exist. In order to add a new function to the Deluge client, we need to be familiar with the structure of Deluge, illustrated in Figure 10. As explained in the former section, deluge depends on libtorrent to implement the BitTorrent protocol. Since libtorrent has the function to limit peer's download and upload speed, we only need to make revisions to source code of Deluge client. In the Deluge client, there are three components: core, UI and plug-in. In this paper, we only make revision to the core section of Deluge client. Structure of the core section is illustrated in Figure 11. Torrent.py and Torrentmanager.py contains functions to limit speed of torrent. So, we add the function to limit upload speed of peers in these two files.

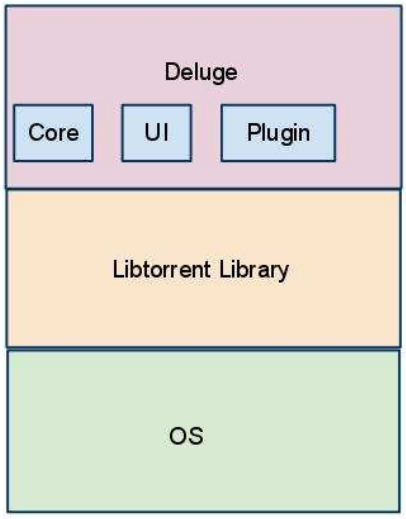

Figure 10: Deluge and Libtorrent

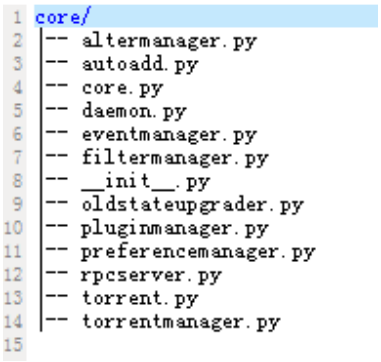

Figure 11: Structure of Deluge Core

4.4 Counter

The counter component works to count the number of seconds a seeder has stayed in BitTorrent network. Generally, one seeder may be seeding multiple files to others. In order to provide fairness for those seeders who are seeding more than one file in the local machine, the counter component will not only count the number of seconds it has stayed in BitTorrent network but also the number of seeding files on the local machine. The formula used in the counter component is $T_n = ((F-1) * 0.1 + 1) * S$. Parameters used in this formula are: F: the number of files seeding in local machine. S is the number of seconds local node has stayed online. The result T_n represents the contributions of local node. The more contributions one node has, the more bandwidth it will get in its next download process.

4.5 Server

A node's server component works to process requests from associated peers' client components. It binds to the port number of 59500. Once a new request arrives, the server will get the contribution computed from counter component and send that

information back to that peer. Pseudo Code of server component is illustrated in Figure 12.

```
1def on_connect_success(result):
2    def on_get_config(result):
3        usertime = result
4    client.myplugin.get_config().addCallback(on_get_config)
5
6def on_connect_fail(result):
7    print "Connection failed!"
8
10def bind(dict):
11    finalstr = ''
12    finalstr = finalstr +'total_time'+ str(dict['total_time']) + 'user_name' + dict['user_name']
13    return finalstr
14
15class EchoProtocol(basic.LineReceiver):
16    d = client.connect()
17    d.addCallback(on_connect_success)
18    d.addErrback(on_connect_fail)
20    def lineReceived(self, line):
21        timestr = bind(usertime)
22        if line == 'quit':
23            self.sendLine(timestr)
24        else:
25            self.sendLine("You said: " + line)
26
28class EchoServerFactory(protocol.ServerFactory):
29    protocol  = EchoProtocol
```

```
30
31if __name__ == "__main__":
32    host = "localhost"
33    reactor.listenTCP(port, EchoServerFactory( ))
34    reactor.run( )
```

Figure 12: Pseudo-Code of Server Component

4.6 Client

The client component is the crucial part of the incentive mechanism. It computes and sets the upload bandwidth for every associate peer. There are 3 steps in the client , illustrated in Figure 13.

1. When a new download process starts, the client component will get the list of associated peers via BitTorrent protocol.
2. With associated peers' IP addresses and port number 59500, the client establishes TCP channels with each of its associated peers.
3. After the client gets each associated peer's contributions, the client will first convert each peer's contribution to its corresponding upload bandwidth. Then, the client will set the upload bandwidth for each one.

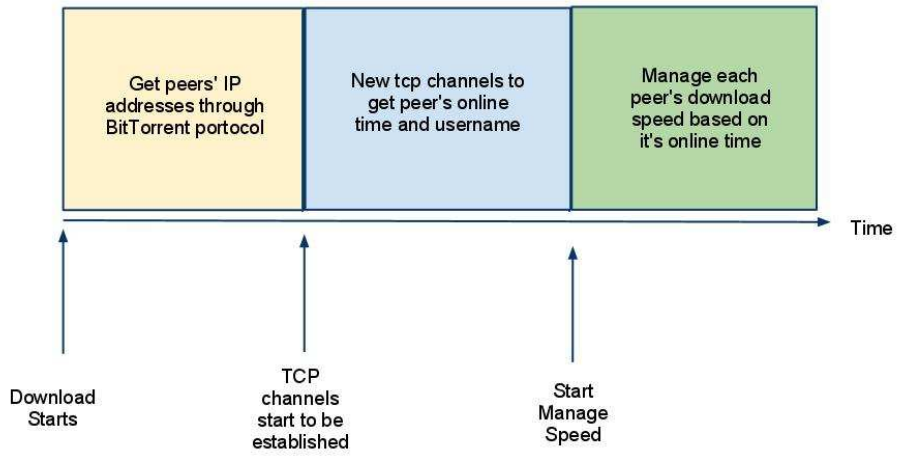

Figure 13: Client Structure

In order to mange download speed of associated peers, it's necessary to use an algorithm which could provide fairness for all peers. In this paper, the algorithm consists of two formulas (Parameters used in these two formulas are: N represents peers in the same download process, L represents the upload bandwidth per torrent, T_n represents each peer's contribution returned from corresponding server.

$$\mathrm{Y_n} = (1 + \frac{1}{\mathrm{T_n}})^{\mathrm{T_n}} \qquad (2)$$

$$D_n = \frac{Y_n}{Y_1 + Y_2 + Y_3 \ldots + Y_n} * \mathrm{L} \qquad (3)$$

Our Incentive mechanism highly depends on online time which means long time users who prefer to seeding files to others will get high contribution value. However, for newcomers to the incentive mechanism, their contributions will be very low. In order to prevent the local node from distributing too much bandwidth for long time users, we use the first formula the value of which is between 1 and e, illustrated in Figure 14. Based on this graph, newcomer to the network will not be severely punished. Furthermore, nodes who stayed longer in the network will be rewarded.

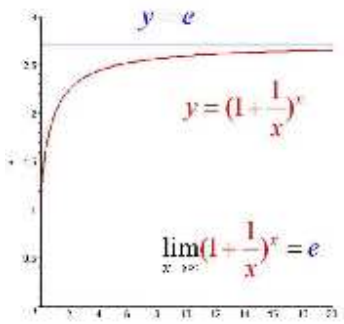

Figure 14: Function Graph

4.7 Usage

In this paper, we test the incentive mechanism on PlanetLab. Steps to run the incentive mechanism:

1. Execute the incentive server component:

   zack@ubuntu:~$*python server.py*

2. Execute the incentive client component:

   zack@ubuntu:~$*python client.py*

3. Execute the incentive client component:

   zack@ubuntu:~$*python counter.py*

5.0 Software Tools, Development Kits Used

Deluge 1.3.0 is selected to test the incentive mechanism. All components of the mechanism were developed using Python 2.7 and Twisted network engine. IDE used for this paper is Eclipse with PyDev plug-in.

6.0 Experimental Results

In this section, we conducted five groups of experiments to validate the property of our incentive mechanism and explained their results in detail.

6.1 Experiment Design

In order to evaluate the performance of the incentive mechanism, several experiments are conducted on PlanetLab [3]. In PlanetLab, we simulate a BitTorrent network of 20 nodes which are from 20 different sites. On each node, we install the Deluge software and all its dependent libraries. Counter script and Client script are running on each node's crontab. Server script is running in the OS's background. Those five experiments are distinguished by whether selected nodes have incentives installed or not. Those five groups of experiments are shown in table 1. Y means the incentive mechanism is installed on that node. N means the incentive mechanism is not installed on that node.

	Seeder	Leecher
Experiment One	Y	Y
Experiment Two	Y	N
Experiment Three	N	Y
Experiment Four	N	N
Verification Experiment	Y	Y

Table 1: Experiments in this section

6.2.1 Experiment 1:

In this experiment, both leechers and seeders have incentive mechanism installed. We conducted three different test cases:

1. One seeder and 19 leechers.

In order to prevent hogging limited network bandwidth of nodes in PlanetLab, we set each node's upload bandwidth per torrent to 100KB/s and download bandwidth per torrent to 200KB/s. Each node's online time in BitTorrent network is illustrated in Table 2.

Node Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 2: G1T1 Node Online Time

Download completion time of all leechers is illustrated in Figure 15. Since only one seeder is available in the whole BitTorrent network, the upload bandwidth set by the seeder doesn't have much influence on others. So, the download completion time of every node is approximately the same except Node 17 and Node 19. Compared with other nodes in the BitTorrent network, Node 17 and Node 19 have much slower network bandwidth. So their download completion time is much longer than others. In summary, incentive mechanism does not have much influence on each node's download completion time. The revised BitTorrent network with one seeder works almost the same as regular BitTorrent network.

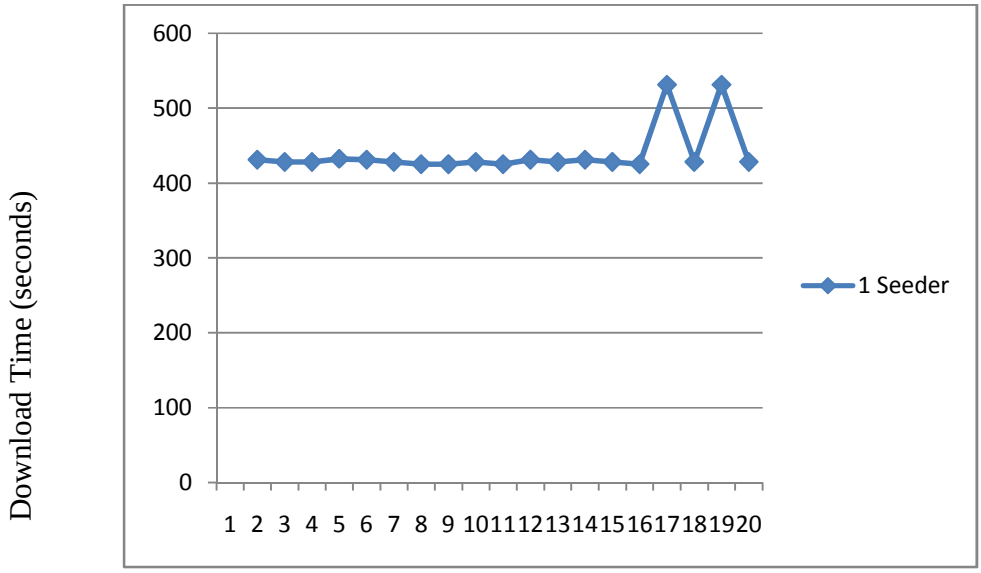

Figure 15: Download completion time of Nodes in G1T1

2. Two seeders and 18 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Each node's online time in BitTorrent network is illustrated in Table 3.

Node Number	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Table 3: G1T2 Node Online Time

Figure 16 shows leechers' download completion time in this test case. Compared with the previous test case which has one available seeder, download completion time of each node decreases significantly. Generally, leechers who contribute more to others could finish their download faster than others. In summary, the incentive mechanism

works to ensure better download speed for those who stay online longer and contribute more to others.

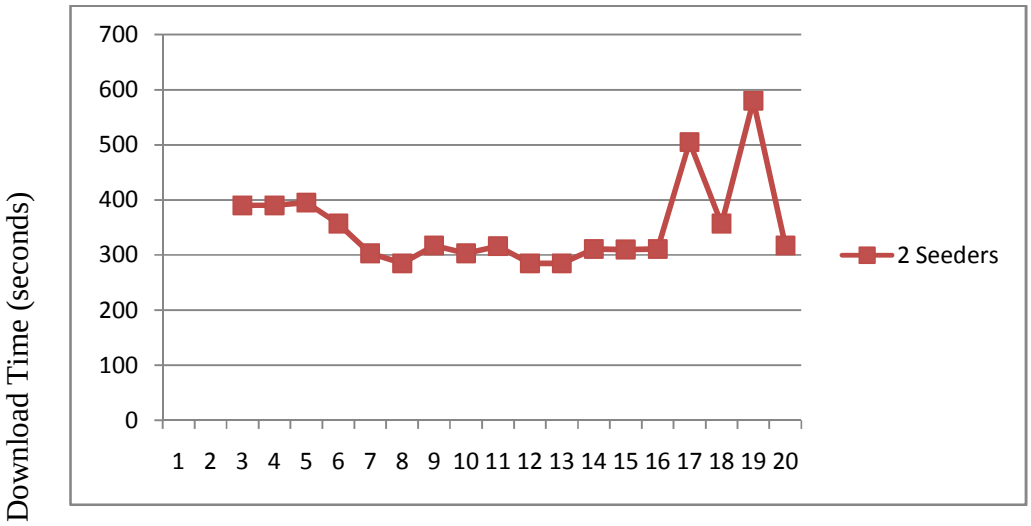

Figure 16: Download Completion Time of Nodes in G1T2

3. Three seeders and 17 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Nodes' online time is illustrated in Table 4.

Node Number	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Table 4: G1T3 Node Online Time

Figure 17 shows leechers' download completion time in this test case. Comparing with the test case 2 of this experiment, each node's download completion time decreases. More importantly, majority of those nodes who contribute more to others finish their download faster than others.

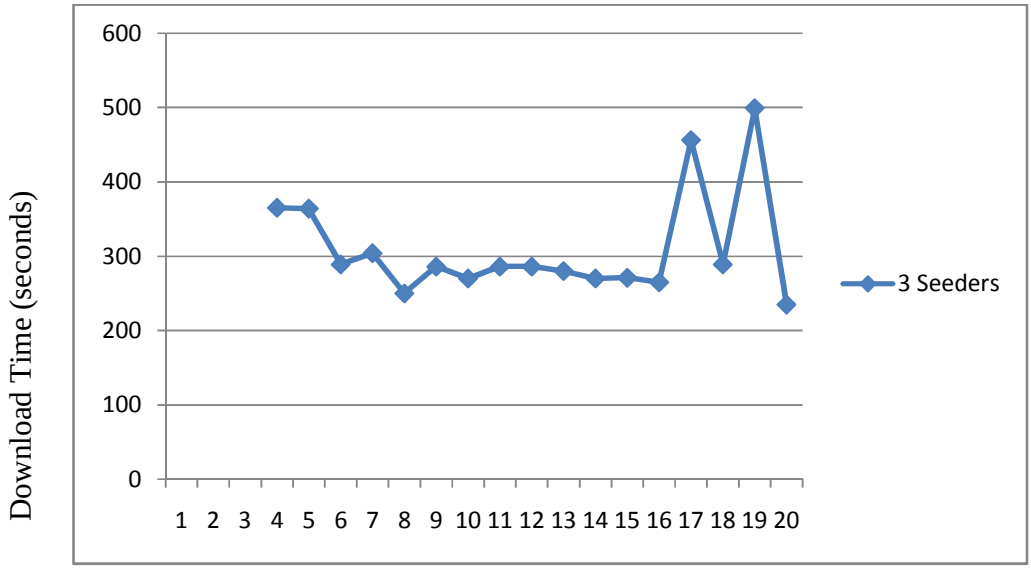

Figure 17: Download Completion Time of Nodes in G1T3

In summary, our first experiment prove that the incentive mechanism could help nodes who contribute more to others get better download speed in a BitTorrent network if there is more than one seeder from the beginning of the download process. If there is only one seeder from the beginning of the download process, the performance of the revised BitTorrent system is almost the same as a regular BitTorrent system.

6.2.2 Experiment 2:

In this experiment, only seeders have incentive mechanisms installed. Leechers don't install incentive mechanisms. Leechers without our incentive mechanism will follow regular BitTorrent TFT incentive uploading to others. There are also three different test cases in this experiment:

1. One seeder and 19 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Node' online time is illustrated in Table 5.

Node Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 5: G2T1 Node Online Time

Figure 18 shows leechers' download completion time. Download completion time of each node is approximately the same. Since only one node has incentive mechanism installed and majority of nodes follow regular BitTorrent, the upload bandwidth set by the seeder will not have much effects on every leecher's download bandwidth. In summary, our incentive mechanism almost does not influence the download completion time of each node in this test case.

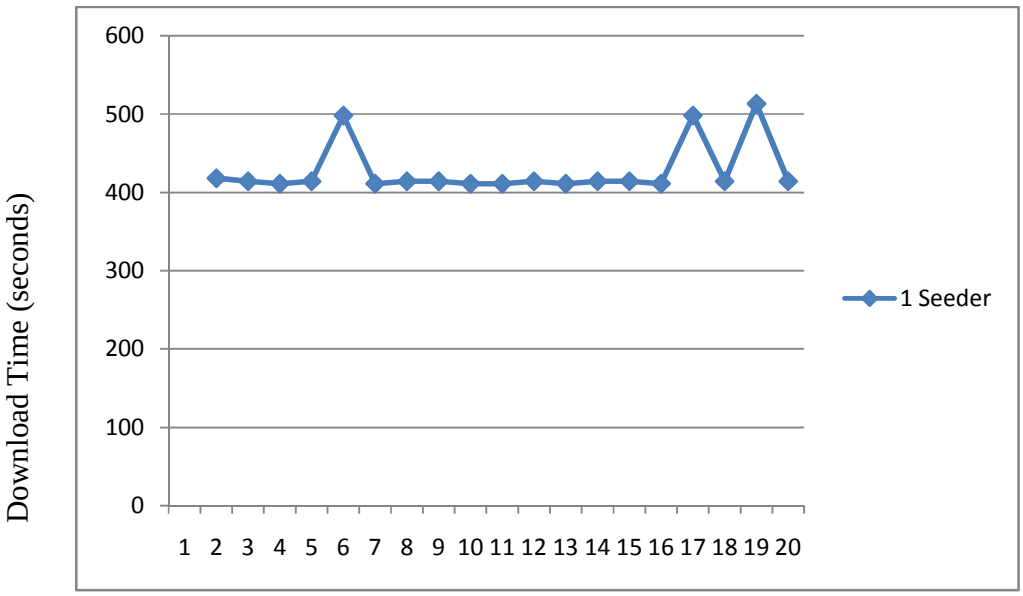

Figure 18: Download Completion Time of Nodes in G2T1

2. Two seeders and 18 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Each node's online time in BitTorrent network is illustrated in Table 6.

Node Number	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Table 6: G2T2 Node Online Time

Each leecher's download completion time is illustrated in Figure 19. The result is similar to the previous test case of this experiment. Since only those two seeders have incentive installed, majority of the nodes will follow the regular BitTorrent protocol to upload to others. The upload bandwidth set by minority of nodes in the BitTorrent will not have many effects on majority of nodes in it. In summary, the incentive mechanism will not influence much on each node's download completion time in this test case. But, each node's download completion time in this test case decreases significantly compared to the previous test case with one seeder.

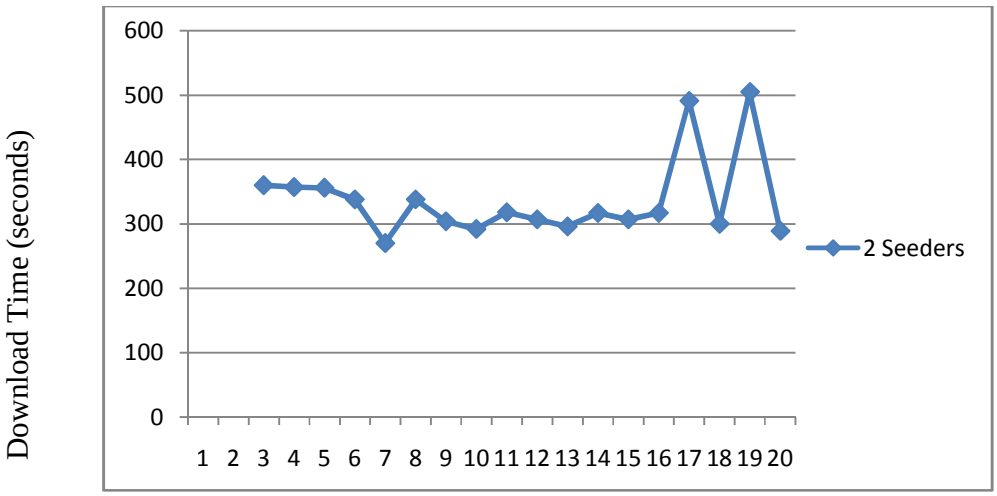

Figure 19: Download Completion Time Nodes in G2T2

3. Three seeders and 17 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Node's online time in BitTorrent network is illustrated in Table 7.

Node Number	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Table 7: G2T3 Node Online Time

Each leecher's download completion is illustrated in Figure 20. In this graph, nodes who contribute more to others don't get better download speed, because most nodes' download bandwidth will not be influcenced by the upload speed set by three seeders.

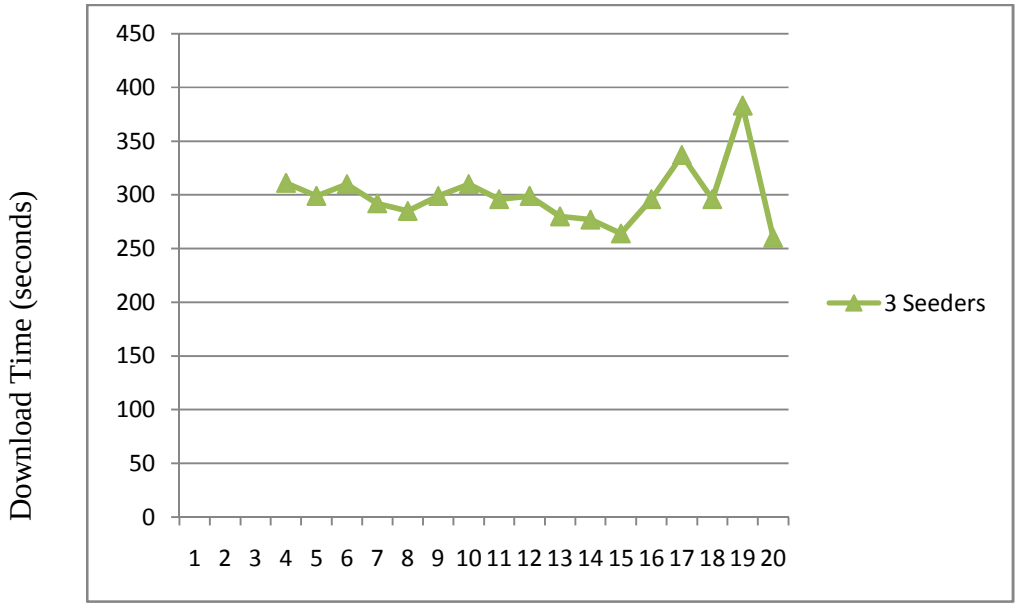

Figure 20: Download Completion Time of Nodes in G2T3

In summary, if majority of nodes in the BitTorrent network don't have incentive mechanism installed, then node who contribute more to others will not be guaranteed to get better download speed than others.

6.2.3 Experiment 3:

In this experiment, we install the incentive mechanism on leechers but not on seeders. There are also three different test cases in this experiment:

1. One seeder and 19 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Each node's online time in BitTorrent network is illustrated in Table 8.

Node Number	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 8: G3T1 Node Online Time

Figure 21 shows leechers' download completion time in this test case. The result is similar to the previous two experiments. Each node's download completion time in the revised BitTorrent system is almost the same as that in regular BitTorrent system.

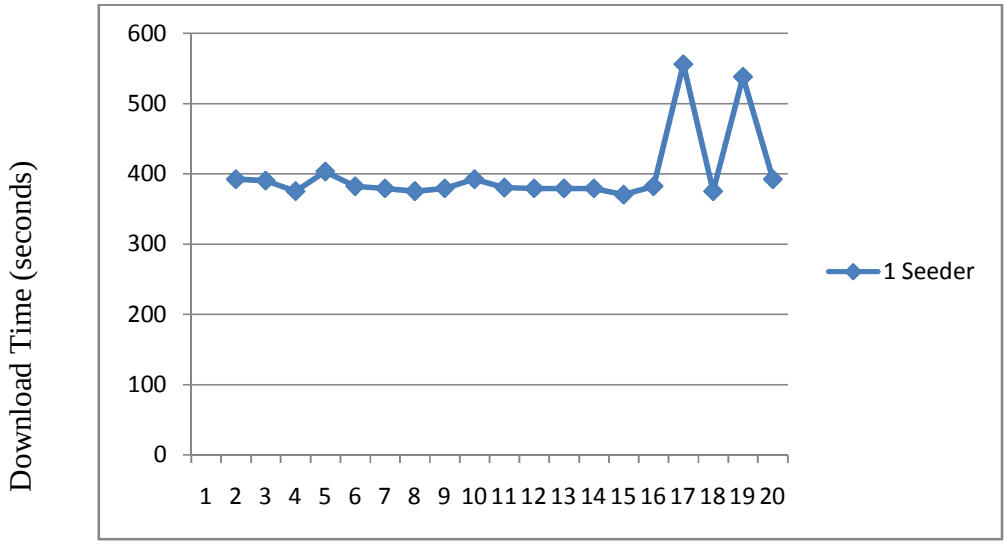

Figure 21: Download Completion Time of Nodes in G3T1

2. Two seeders and 18 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s. Each node's online time in BitTorrent network is illustrated in Table 9.

Node Number	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Table 9: G3T2 Node Online Time

Each leecher's download completion time is illustrated in Figure 22. Compared with the previous test case in this experiment, the download completion time for each node decreases significantly. Furthermore, nodes that stay longer and contribute more to others will get better download speed than others.

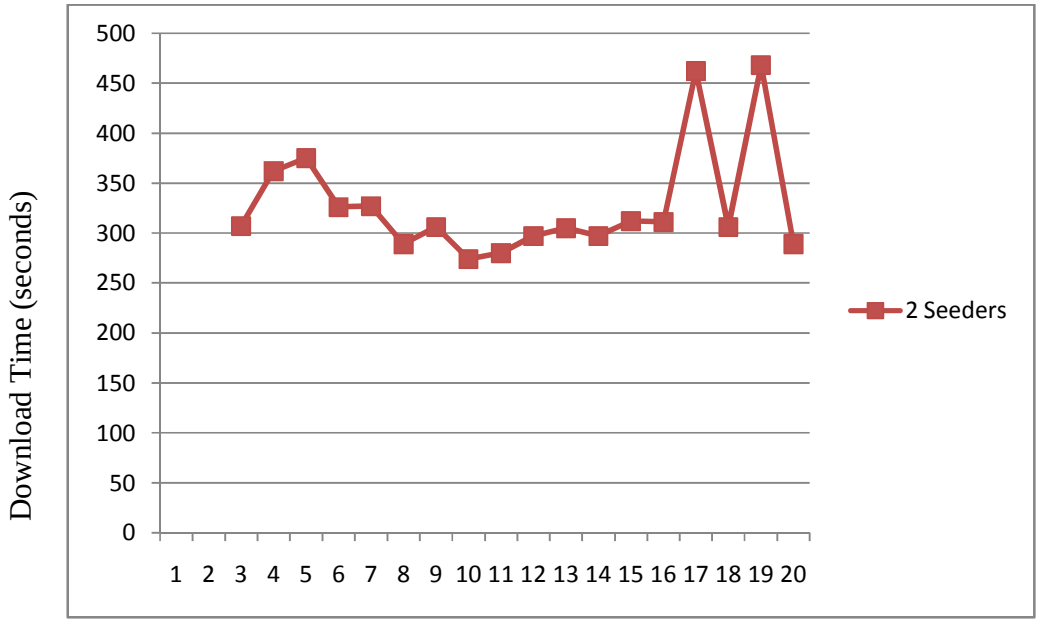

Figure 22: Download Completion Time of Nodes in G3T2

3. Three seeders and 17 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to to 200KB/s. Each Node's online time in BitTorrent network is illustrated in Table 10.

Node Number	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Online Time(day)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Table 10: G3T3 Node Online Time

Each leecher's download completion time is illustrated in Figure 23. From the graph, we could see that some nodes who contribute more to others will finish downloading faster than others. However, some nodes who also contribute more to others don't get better download speed. The reason is that the network situation of each node

varies a lot in PlanetLab and those three seeders disbtribute resource evenly. In experiment 2, all of those test cases almost don't have influence on each leecher's download completion time. But in this experiment, generally, nodes who contribute more to others will get better download speeds. In Summary, this experiment shows that the number of nodes that have incentive mechanisms installed will influence the final result.

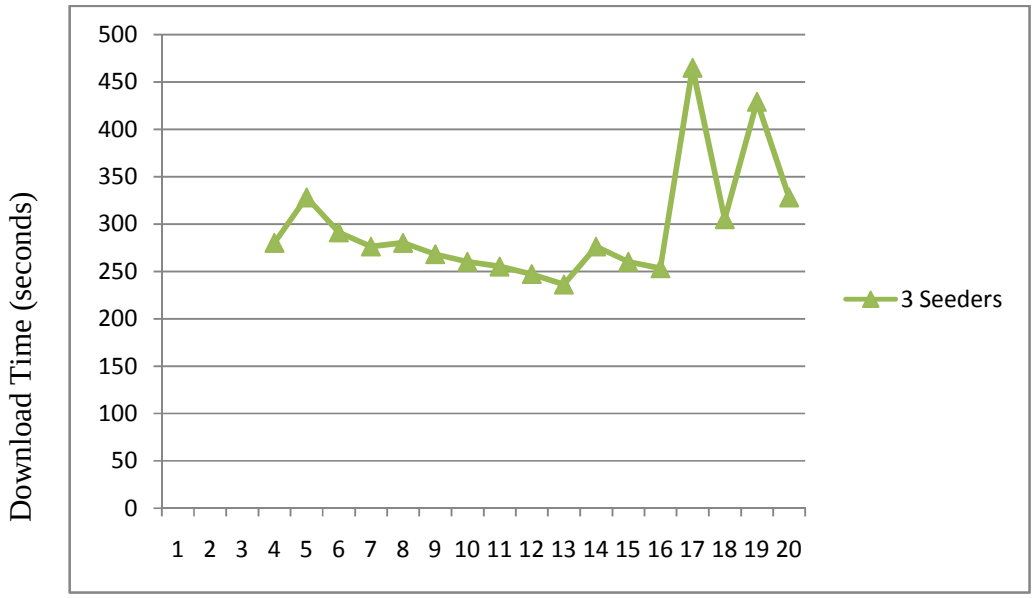

Figure 23: Download Completion Time of Nodes in G3T3

6.2.4 Experiment 4:

In this experiment, we run a regular BitTorrent test which doesn't have incentive mechanisms installed on leerchers nor on seeders. We conducted three different test cases:

1. One seeder and 19 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s.

Figure 24 shows each leecher's download completion time. In regular Bittorrent networks, the seeder tries to ensure fairness which means to distribute resource evenly to each client. So, the download completion time for each node is approximately the same.

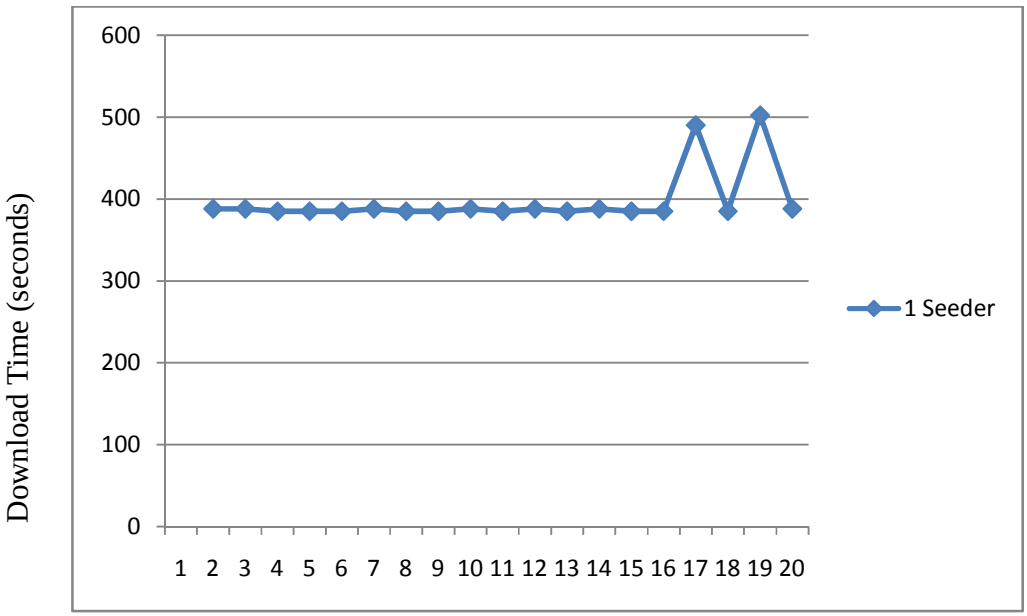

Figure 24: Download completion time of Nodes in G4T1

2. Two seeders and 18 leechers.

Each node's upload bandwidth per torrent is set to 100KB/s and download bandwidth per torrent is set to 200KB/s.

Figure 25 shows leecher's download completion time. In regular BitTorrent, nodes that have better download bandwidth will complete their downloading process faster than others. Compared with the previous test case, each node's download completion time decreased significantly.

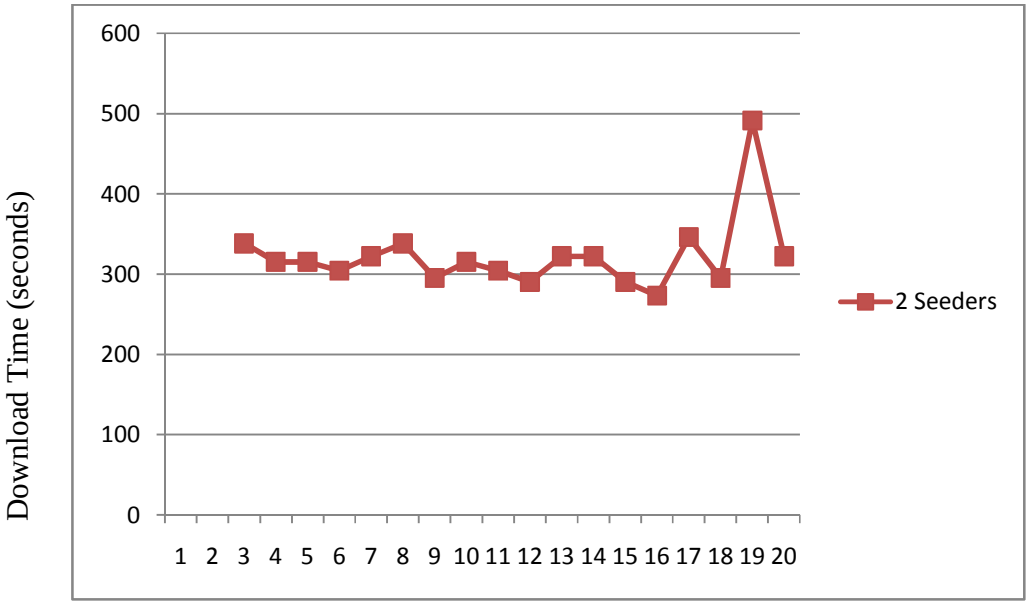

Figure 25: Download Completion Time of Nodes in G4T2

3. Three seeders and 17 leechers.

Each node's upload bandwidth is set to 100KB/s and download bandwidth is set to 200KB/s.

Figure 26 shows leechers' download completion time. Compared with the previous test case which has two seeders, some nodes' download completion decreased significantly. However, most nodes' download completion time is the same or even increased a little bit.

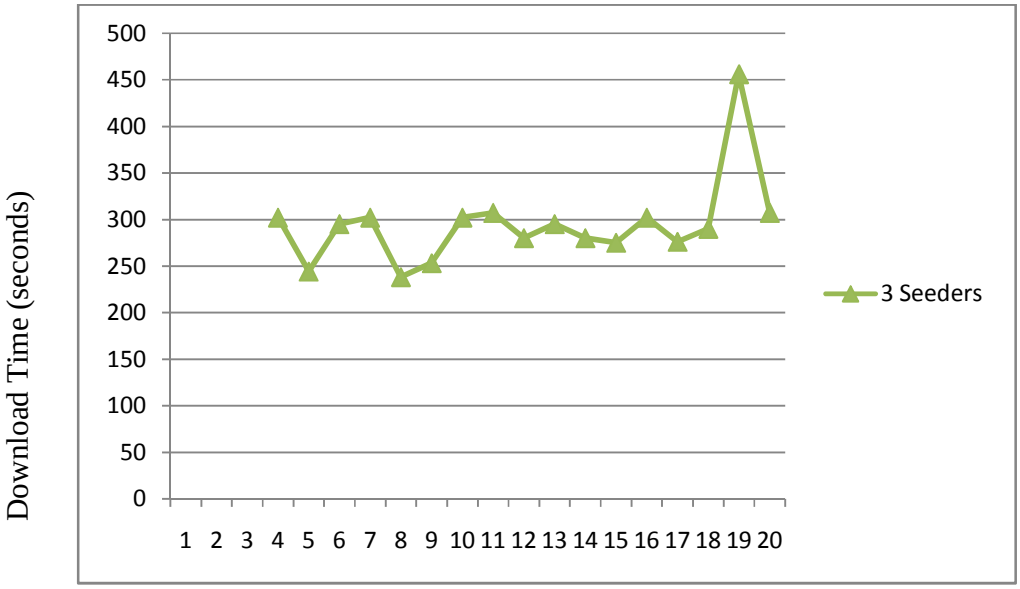

Figure 26: Download Completion Time of Nodes in G4T3

In summary, in regular BitTorrent, more seeders don't mean faster download completion for all nodes.

6.3 Verification Experiment

In all three experiments for revised BitTorrent system, online time set for each node is the same. In order to validate that it is the incentive mechanism instead of any other factors that help nodes stay online longer and contribute more to others getting better download speeds. In this section, we conducted two test cases which have reversed order of online time, illustrated in Table 11.

Node Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Time Increased(day)	1	2	3	4	5	6	7	8	9	10	11	12	13
Time Decreased(day)	13	12	11	10	9	8	7	6	5	4	3	2	1

Table 11: Verification Test Online Time

Download completion time of each node of two test cases are illustrated in Figure 27. From the graph, we could see that the download completion time of nodes is an increasing order from left to right in the time decreased test case. And, the download completion time of nodes is a decreasing order from left to right in the time increased test case. From this experiment, we could prove that it is because of the incentive mechanism, instead of other factors that make those nodes stay longer finish downloading process faster than others.

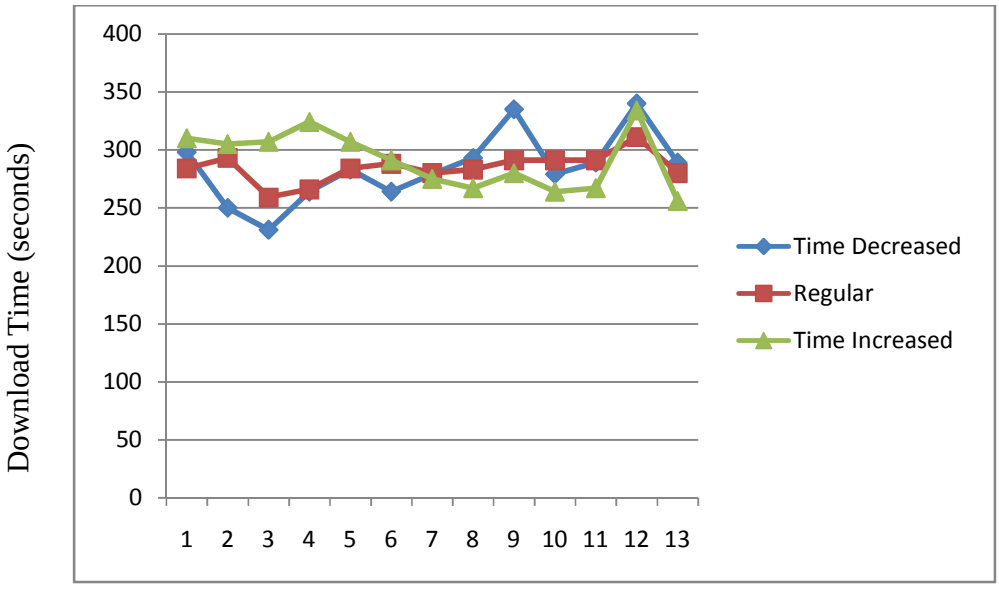

Figure 27: Download Completion time of Nodes in Verification Experiment

6.4 Security

In this paper, we mainly focus on the effects of incentive mechanisms to regular BitTorrent networks. There are many security problems which may ruin the incentive mechanism. First, each node who installs the incentive mechanism will create a log.dat file to store its online time which is vulnerable to malicious attacks. Secondly, the client component of our incentive mechanism receives unencrypted data from server. There is a

possibility that some peers may send fake information back to the client in order to to get better download speeds.

7.0 Conclusion

This paper proposes an incentive mechanism to incentivize seeders upload to others and evaluate the performance of the incentive mechanism by comparing the download completion time in regular BitTorrent networks with three other revised BitTorrent networks. Experimental results demonstrated that the proposed incentive mechanism could help those who contributed more to others get better download speed. Furthermore, if the majority of nodes in a BitTorrent network have incentive mechanisms installed, the download completion time of each node will decrease with the increasing number of seeders. However, in regular BitTorrent, the increasing number of seeders will not result in the decrease of every node's download completion time.

Content in BitTorrent is stored in all nodes in the network. The more nodes a BitTorrent system has, the more available the content in it is. The incentive mechanism which incentivizes seeders to stay longer in the network could potentially increase the available content in the BitTorrent network.

Although the protocol can help those who stay longer in the network get better download speed, it also has weaknesses. In some situations, it will increase the overall download completion time. Most importantly, the incentive mechanism needs to be installed on the majority of nodes in the BitTorrent network in order to work efficiently. Only small portions of nodes in a BitTorrent network have incentive mechanism

installed, each node's download completion time will not be managed effectively by the incentive mechanism.

8.0 Future Work

This work opens up many directions for future research. One major piece is that the incentive mechanism couldn't decrease the overall download completion time of the BitTorrent network.

A better algorithm will be able to detect the size of shared content and the quality of shared content in one node. If the shared content is very rare in the BitTorrent network, it should be rewarded more than those staying online but only have very commonly shared content.

Furthermore, the incentive mechanism needs to run third party scripts and make revisions to the core portion of Deluge. These changes make Deluge very hard to deploy in different platforms. Improvements for the installation of incentive mechanism to be easily deployed on different platforms are necessary.

Appendix A

File: client.py

```
from deluge.ui.client import client
from deluge.log import setupLogger
from twisted.internet import reactor, defer, protocol
from twisted.protocols import basic
from collections import defaultdict
from twisted.protocols.basic import LineReceiver
from twisted.internet.protocol import ClientFactory
setupLogger()


#struct of peer_info {{0: {'torrent_id': 'c3dabbfcad23ce50566eac209e004fe9903e35ea',
#                          'ip': '60.181.114.35:12345',
#                          'time': 1,
#                          'speed': 2},

global peer_info
global counter

#number of conections establised to all associated peers
global connection
#sum of associated peers of all torrent files
global total_peer
#upload bandwidth of local pc
global up_bandwidth
up_bandwidth = 100
port = 59500
total_peer = 0
connection = 1
counter = 0

peer_info = defaultdict( dict )

'''
Component to get peers in local pc, Two functions:
1. get_torrent_id will get all torrent ids in current deluge client
2. get_peer will get each torrent's associated peers
'''
def get_torrent_id():
    d = client.connect()
    def on_get_id_success(result):
        print "Connection in get_torrent_id was successful!"
        def on_get_config(result):
            get_peer(result)
        client.core.get_session_state().addCallback(on_get_config)
    d.addCallback(on_get_id_success)

    def on_get_id_fail(result):
        print "Connection in on_get_id_fail failed!"
        print "result:", result

    # We add the callback (in this case it's an errback, for error)
    d.addErrback(on_get_id_fail)
```

```
#get every torrent's associated peers in local PC
def get_peer(torrent_id):
    d = client.connect()
    def on_connect_success(result):
        print "Connection in get_peer was successful!"
        def on_get_config(result, torrent_number, m):
            #detect whether a torrent has peer or not
            global counter
            counter = counter + 1
            if(result['peers'] == ()):
                print "torrent id", torrent_id[torrent_number], "doesn't has associated peers"
            else:
                for i in range(0, len(result['peers'])):
                    global peer_info, total_peer
                    peer_info[total_peer]['ip'] = result['peers'][i]["ip"]
                    peer_info[total_peer]['torrent_id'] = torrent_id[torrent_number]
                    total_peer = total_peer + 1
                    #Get out of the loop and give the dict to connect function
                    if((counter == m) & (i == len(result['peers']) - 1)):
                        clientconnection()
        for i in range(0, len(torrent_id)):
            client.core.get_torrent_status(torrent_id[i], ["peers"], True).addCallback(on_get_config, i,
len(torrent_id))
    d.addCallback(on_connect_success)

    def on_connect_fail(result):
        print "Connection failed!"
        print "result:", result
    d.addErrback(on_connect_fail)

'''
Component to connect peers and get time
1. client connection. Initialize the connection to each peer
2. class GetTime. Define the behavior if connectMade is success or line received is success
   if lineReceived is success, then computer ip value sent back from server with ip value stored in peer_info
   if these two values are the same, then set the time send back from server to the corresponding ip node
3. clas GetTimefactory. Define the behavior if the connection is lost of failed
4. In this component, if the number of connection is equal to the number of peers, then we go to the
setemptyvalue() funciton
   in the setemptyvalue function, it will set empty time value to 1 which means that specific server doesn't
install
   the incentvie mechanism
'''
#connect to all associated peers in local PC
def clientconnection():
    global peer_info
    factory = GetTimeFactory()
    for i in range(0, len(peer_info)):
        reactor.connectTCP(peer_info[i]['ip'].split(":")[0], port, factory)

class GetTime(LineReceiver):
    def connectionMade(self):
        self.sendLine("online time")
    def lineReceived(self, line):
        global connection
```

```
        for i in range(0, len(peer_info)):
            #line.split(":")[0] is the ip address of peer
            #line.split(":")[1] is the online time of peer
            if (peer_info[i]['ip'] == line.split(":")[0]):
                peer_info[i]['time'] = line.split(":")[1]
        print "ipcorrect", "connection:", connection, "peer_info", len(peer_info)
        if(connection == len(peer_info)):
            setemptyvalue()
        connection = connection + 1
        self.transport.loseConnection()

#if the number of connections is equal the number of total peers in local PC.
#total peers means the sum of peers of all torrents
#for example: torrent one has 10 peers. torrent two has 20 peers. Then, the total number of peers is 30
class GetTimeFactory(ClientFactory):
    protocol = GetTime
    def clientConnectionFailed(self, connector, reason):
        global connection
        print 'connection failed:', "connection", connection, "peer_info",
len(peer_info),reason.getErrorMessage()
        if(connection == len(peer_info)):
            setemptyvalue()
        connection = connection + 1
    def clientConnectionLost(self, connector, reason):
        global connection
        print "connectionlost", "connection", connection, "peer_info",
len(peer_info),reason.getErrorMessage()
        if(connection == len(peer_info)):
            setemptyvalue()
        connection = connection + 1


def setemptyvalue():
    global peer_info
    for i in range(0, len(peer_info)):
        try:
            peer_info[i]['time']
        except KeyError:
            peer_info[i]['time'] = 1.0
    speedmanager()

'''
Manage speed has three function:
1. compute. Compute each peer's upload speed bandwidth
2. speedmanager. process the whole peer_info dictionary and use compute function to calculate
3. speedset. set the key"speed" of peer_info dictionary as each peer's upload limit
'''
def compute(min, max):
    sum = 0
    for i in range(min, max):
        sum = sum + int(peer_info[i]['time'])
    for i in range(min, max):
        peer_info[i]['speed'] = float(peer_info[i]['time']) * up_bandwidth  / sum
```

```
def speedmanager():
    global peer_info
    sum = 0
    j = 0
    #in the for loop, if j is not equal to i, we compute from j to i
    #after we are out of the for loop, the last torrent is not computed, so we use compute function out side the
for
    #to process the last part
    for i in range(0, len(peer_info)):
        if (peer_info[j]['torrent_id']) != peer_info[i]['torrent_id']:
            compute(j, i)
            j = i
    compute(j, len(peer_info))
    speedset()


def speedset():
    global peer_info
    speedcontrol = client.connect()
    #ip = ('150.254.186.34', 80)
    # We create a callback function to be called upon a successful connection
    def speedcontrol_success(result):
        def on_get_config(result, torrent_id, ip, value):
            print "speed set for ",torrent_id, ip ,"value is", value
        for i in range(0, len(peer_info)):
            ip_port = (peer_info[i]['ip'].split(':')[0], int(peer_info[i]['ip'].split(':')[1]))
            client.core.set_peer_max_upload_speed(peer_info[i]['torrent_id'],                    ip_port,
peer_info[i]['speed']).addCallback(on_get_config, peer_info[i]['torrent_id'], ip_port, peer_info[i]['speed'])

    def speedcontrol_fail(result):
        print "Connection to control speed failed!"
    speedcontrol.addCallback(speedcontrol_success)
    speedcontrol.addErrback(speedcontrol_fail)

def main():
    #peer_connection()
    get_torrent_id()
    reactor.run( )

if __name__ == "__main__":
    main()
```

File: server.py

```
'''
Created on Mar 7, 2011
@author: zack
'''
from deluge.log import setupLogger
from twisted.internet import reactor, defer, protocol
from twisted.protocols import basic
import math
import socket

setupLogger()
```

```
#location of the files store online time
global filename
#user's online time
global time
#if the pc doesn't install the incentive mechanism, its default is 1 instead of 0
#because we have a fractional function to compute value
time = 1

port = 59500
filename = "/home/zack/Documents/log3.dat"
#####################################
#Get local machine's online time
#structure of data send back to client
#(ip:time) for example:
#(192.168.1.1:1.0034)
#####################################
def fileinput(path):
    try:
        global time
        file = open(path,"r")
        inline = file.readline()
        value = inline.split(':')[1]
        time = float(value)
    except IOError:
        pass
    #86400 is the number of seconds of a day, we want the value of time increase by 0.24 per day.
    #we could find the prototype of the funcion in PPT stored in win7 CS298 file
    time = (float(time) / 86400)*0.24
    T = math.pow(1+1/float(time),float(time))
    #get local PC's ip address
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(("gmail.com",80))
    finalstr = ''
    finalstr = s.getsockname()[0] + ":" + str(T)
    return finalstr
#####################################
#Get usernames and time from each peer
#####################################
'"server wait part"'
class EchoProtocol(basic.LineReceiver):
    def lineReceived(self, line):
        if line == 'online time':
            value = fileinput(filename)
            self.sendLine(str(value))
        else:
            self.sendLine("You said: " + line)

class EchoServerFactory(protocol.ServerFactory):
    protocol  = EchoProtocol

if __name__ == "__main__":
    reactor.listenTCP(port, EchoServerFactory( ))
    reactor.run( )
```

File: counter.py

```
from deluge.ui.client import client
from twisted.internet import reactor
from twisted.internet import task
from deluge.log import setupLogger
setupLogger()

#each user's online time
global time

#number of files seeding in local machine
global seed_number

#value should add for time per second
global adder

#count number of threads in local PC
global counter

#location and name of the file we want to store user's online time
global filename

#initialize value of global variable and file location
counter = 0
seed_number = 0
time = 0
filename = "/home/zack/Documents/log3.dat"
#store user's online time

def fileinput(filename):
    try:
        global time
        file = open(filename,"r")
        inline = file.readline()
        value = inline.split(':')[1]
        time = float(value)
    except IOError:
        pass

def get_torrent_id():
    d = client.connect()
    def on_get_id_success(result):
        print "Connection in get_torrent_id was successful!"
        def on_get_config(result):
            get_progress(result)

        client.core.get_session_state().addCallback(on_get_config)
    d.addCallback(on_get_id_success)

    def on_get_id_fail(result):
        print "Connection in on_get_id_fail failed!"
        print "result:", result

    # We add the callback (in this case it's an errback, for error)
    d.addErrback(on_get_id_fail)
```

```
#Get the number of files seeding in local machine
def get_progress(torrent_id):
    d = client.connect()
    global seed_number
    global adder
    global counter
    seed_number = 0
    adder = 0
    counter = 0
    def on_connect_success(result):
        print "Connection in get_progress was successful!"
        def on_get_config(result):
            global seed_number
            global adder
            global counter
            global time
            counter = counter + 1
            if result["progress"] == 100.0:
                seed_number = seed_number + 1
                adder = (seed_number - 1)*0.1 + 1
                time = adder + time
                print "online time is :", time
                FILE = open(filename,"w")
                FILE.write("online time:" + str(time))
                #the if condition loop will return the final adder value
            if counter == len(torrent_id):
                print "seed number:", seed_number, "adder value is:", adder

        for i in range(0, len(torrent_id)):
            client.core.get_torrent_status(torrent_id[i], ["progress"], True).addCallback(on_get_config)

    d.addCallback(on_connect_success)

    def on_connect_fail(result):
        print "Error, get torrent progress"
        print "result:", result
    d.addErrback(on_connect_fail)


l = task.LoopingCall(get_torrent_id)
l.start(1.0) # call every second
fileinput(filename)

reactor.run()
```

```
#File: Install
#Install necessary libraries for deluge
su -c 'yum groupinstall "Development Tools"'
su -c 'yum install wget python python-devel twisted pyOpenSSL gettext pyxdg boost boost-devel openssl openssl-devel zlib zlib-devel libnotify pygame xdg-utils python-mako python-chardet pygtk2 pygtk2-devel python-setuptools python-setuptools-devel librsvg2-devel'
mkdir ~/deluge && cd ~/deluge && wget http://download.deluge-torrent.org/source/deluge-1.3.0.tar.bz2 && wget http://libtorrent.googlecode.com/files/libtorrent-rasterbar-0.14.11.tar.gz
gunzip libtorrent-rasterbar-0.14.11.tar.gz
```

```
tar -xvjf deluge-1.3.0.tar.bz2 && tar -xvf libtorrent-rasterbar-0.14.11.tar
mv libtorrent-rasterbar-0.14.11 libtorrent
mv libtorrent deluge-1.3.0
cd ~/deluge/deluge-1.3.0 && python setup.py clean -a && python setup.py build && su -c 'python setup.py install'

#File: CopyLibrary
#Copy simplejson, chardet libraries to nodes of Planetlab.
#!/bin/sh
# Rotate procmail log files
node="ds-pl1.technion.ac.il"
cd /home/zack/Downloads
scp -i ~/Documents/id_rsa simplejson.tar.gz sjsu_jsu_p2p_streaming@$node:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa chardet-1.0.1.tar.gz sjsu_jsu_p2p_streaming@$node:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa charjson sjsu_jsu_p2p_streaming@$node:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa installfile sjsu_jsu_p2p_streaming@$node:/home/sjsu_jsu_p2p_streaming
#File: UpAllScript

#Upload counter.py, client.py, server.py to 20 nodes of PlanetLab
#!/bin/sh
# Rotate procmail log files
node1="planetlab1.cesnet.cz"
node2="planetlab1.cs.stevens-tech.edu"
node3="planetlab-2.imperial.ac.uk"
node4="planetlab1.ucsd.edu"
node5="planetx.scs.cs.nyu.edu"
node6="planetlab2.hiit.fi"
node7="planetlab2.cs.pitt.edu"
node8="planetlab1.cs.pitt.edu"
node9="planetlab2.ucsd.edu"
node10="ricepl-1.cs.rice.edu"
node11="pl1.eecs.utk.edu"
node12="planetlab1.dtc.umn.edu"
node13="planetlab2.cs.stevens-tech.edu"
node14="planetlab-01.bu.edu"
node15="lefthand.eecs.harvard.edu"
node16="agni.iitd.ernet.in"
node17="planetlab-2.cs.uh.edu"
node18="planetlab-2.ssvl.kth.se"
node19="planetlab-2.cse.ohio-state.edu"
node20="righthand.eecs.harvard.edu"
cd /home/zack/workspace/CS298/src
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node1:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node2:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node3:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node4:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node5:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py sjsu_jsu_p2p_streaming@$node6:/home/sjsu_jsu_p2p_streaming
```

```
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node7:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node8:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node9:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node10:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node11:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node12:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node13:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node14:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node15:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node16:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node17:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node18:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node19:/home/sjsu_jsu_p2p_streaming
scp -i ~/Documents/id_rsa client.py counter.py server.py
sjsu_jsu_p2p_streaming@$node20:/home/sjsu_jsu_p2p_streaming
```

File: torrent.py

```
# torrent.py
#

"""Internal Torrent class"""

import os
import time
from urllib import unquote
from urlparse import urlparse

from deluge._libtorrent import lt

import deluge.common
import deluge.component as component
from deluge.configmanager import ConfigManager, get_config_dir
from deluge.log import LOG as log
from deluge.event import *

from time import gmtime, strftime
TORRENT_STATE = deluge.common.TORRENT_STATE

    def set_max_connections(self, max_connections):
        self.options["max_connections"] = int(max_connections)
        self.handle.set_max_connections(max_connections)
```

```
def set_max_upload_slots(self, max_slots):
    self.options["max_upload_slots"] = int(max_slots)
    self.handle.set_max_uploads(max_slots)

def set_max_upload_speed(self, m_up_speed):
    self.options["max_upload_speed"] = m_up_speed
    if m_up_speed < 0:
        v = -1
    else:
        v = int(m_up_speed * 1024)

    self.handle.set_upload_limit(v)

def set_max_download_speed(self, m_down_speed):
    self.options["max_download_speed"] = m_down_speed
    if m_down_speed < 0:
        v = -1
    else:
        v = int(m_down_speed * 1024)
    self.handle.set_download_limit(v)

#add this new function to limit speed of peer
def set_peer_download_speed(self, ip, value):
    if value < 0:
        v =  -1
    else:
        v = int(value * 1024)
    filename = "/home/zack/Documents/log.dat"
    FILE = open(filename,"a")
    FILE.write("pymodules download: " + str(strftime("%a, %d %b %Y %H:%M:%S", gmtime())) + "\n")
    FILE.close()
    self.handle.set_peer_download_limit(ip, v)

def set_peer_upload_speed(self, ip, value):
    if value < 0:
        v =  -1
    else:
        v = int(value * 1024)
    filename = "/home/zack/Documents/log.dat"
    FILE = open(filename,"a")
    FILE.write("pymodules upload:" + str(strftime("%a, %d %b %Y %H:%M:%S", gmtime())) + "\n")
    FILE.close()
    self.handle.set_peer_upload_limit(ip, v)

def set_stop_ratio(self, stop_ratio):
    self.options["stop_ratio"] = stop_ratio

def set_stop_at_ratio(self, stop_at_ratio):
    self.options["stop_at_ratio"] = stop_at_ratio

def set_remove_at_ratio(self, remove_at_ratio):
    self.options["remove_at_ratio"] = remove_at_ratio

def set_trackers(self, trackers):
    """Sets trackers"""
```

```
        if trackers == None:
            trackers = []
            for value in self.handle.trackers():
                tracker = {}
                tracker["url"] = value.url
                tracker["tier"] = value.tier
                trackers.append(tracker)
            self.trackers = trackers
            self.tracker_host = None
            return

        log.debug("Setting trackers for %s: %s", self.torrent_id, trackers)
        tracker_list = []

        for tracker in trackers:
            new_entry = lt.announce_entry(tracker["url"])
            new_entry.tier = tracker["tier"]
            tracker_list.append(new_entry)
        self.handle.replace_trackers(tracker_list)

        # Print out the trackers
        #for t in self.handle.trackers():
        #    log.debug("tier: %s tracker: %s", t["tier"], t["url"])
        # Set the tracker list in the torrent object
        self.trackers = trackers
        if len(trackers) > 0:
            # Force a reannounce if there is at least 1 tracker
            self.force_reannounce()

    def get_eta(self):
        """Returns the ETA in seconds for this torrent"""
        if self.status == None:
            status = self.handle.status()
        else:
            status = self.status

        if self.is_finished and self.options["stop_at_ratio"]:
            # We're a seed, so calculate the time to the 'stop_share_ratio'
            if not status.upload_payload_rate:
                return 0
            stop_ratio = self.options["stop_ratio"]
            return ((status.all_time_download * stop_ratio) - status.all_time_upload) /
status.upload_payload_rate

        left = status.total_wanted - status.total_done

        if left <= 0 or status.download_payload_rate == 0:
            return 0

        try:
            eta = left / status.download_payload_rate
        except ZeroDivisionError:
            eta = 0

        return eta
```

```
def get_ratio(self):
    """Returns the ratio for this torrent"""
    if self.status == None:
        status = self.handle.status()
    else:
        status = self.status

    if status.total_done > 0:
        # We use 'total_done' if the downloaded value is 0
        downloaded = status.total_done
    else:
        # Return -1.0 to signify infinity
        return -1.0

    return float(status.all_time_upload) / float(downloaded)


def get_peers(self):
    """Returns a list of peers and various information about them"""
    ret = []
    peers = self.handle.get_peer_info()

    for peer in peers:
        # We do not want to report peers that are half-connected
        if peer.flags & peer.connecting or peer.flags & peer.handshake:
            continue
        try:
            client = str(peer.client).decode("utf-8")
        except UnicodeDecodeError:
            client = str(peer.client).decode("latin-1")

        # Make country a proper string
        country = str()
        for c in peer.country:
            if not c.isalpha():
                country += " "
            else:
                country += c

        ret.append({
            "client": client,
            "country": country,
            "down_speed": peer.down_speed,
            "ip": "%s:%s" % (peer.ip[0], peer.ip[1]),
            "progress": peer.progress,
            "seed": peer.flags & peer.seed,
            "up_speed": peer.up_speed,
        })

    return ret

def get_file_progress(self):
    """Returns the file progress as a list of floats.. 0.0 -> 1.0"""
    if not self.handle.has_metadata():
        return 0.0
```

```
    file_progress = self.handle.file_progress()
    ret = []
    for i,f in enumerate(self.get_files()):
        try:
            ret.append(float(file_progress[i]) / float(f["size"]))
        except ZeroDivisionError:
            ret.append(0.0)

    return ret


def pause(self):
    """Pause this torrent"""
    # Turn off auto-management so the torrent will not be unpaused by lt queueing
    self.handle.auto_managed(False)
    if self.handle.is_paused():
        # This torrent was probably paused due to being auto managed by lt
        # Since we turned auto_managed off, we should update the state which should
        # show it as 'Paused'.  We need to emit a torrent_paused signal because
        # the torrent_paused alert from libtorrent will not be generated.
        self.update_state()
        component.get("EventManager").emit(TorrentStateChangedEvent(self.torrent_id, "Paused"))
    else:
        try:
            self.handle.pause()
        except Exception, e:
            log.debug("Unable to pause torrent: %s", e)
            return False

    return True

def resume(self):
    """Resumes this torrent"""

    if self.handle.is_paused() and self.handle.is_auto_managed():
        log.debug("Torrent is being auto-managed, cannot resume!")
        return
    else:
        # Reset the status message just in case of resuming an Error'd torrent
        self.set_status_message("OK")

        if self.handle.is_finished():
            # If the torrent has already reached it's 'stop_seed_ratio' then do not do anything
            if self.options["stop_at_ratio"]:
                if self.get_ratio() >= self.options["stop_ratio"]:
                    #XXX: This should just be returned in the RPC Response, no event
                    #self.signals.emit_event("torrent_resume_at_stop_ratio")
                    return

        if self.options["auto_managed"]:
            # This torrent is to be auto-managed by lt queueing
            self.handle.auto_managed(True)

        try:
            self.handle.resume()
        except:
```

```
                pass

            return True

    def connect_peer(self, ip, port):
        """adds manual peer"""
        try:
            self.handle.connect_peer((ip, int(port)), 0)
        except Exception, e:
            log.debug("Unable to connect to peer: %s", e)
            return False
        return True

    def move_storage(self, dest):
        """Move a torrent's storage location"""
        if not os.path.exists(dest):
            try:
                # Try to make the destination path if it doesn't exist
                os.makedirs(dest)
            except IOError, e:
                log.exception(e)
                log.error("Could not move storage for torrent %s since %s does not exist and could not create the
directory.", self.torrent_id, dest)
                return False
        try:
            self.handle.move_storage(dest.encode("utf8"))
        except:
            return False

        return True

    def save_resume_data(self):
        """Signals libtorrent to build resume data for this torrent, it gets
        returned in a libtorrent alert"""
        self.handle.save_resume_data()
        self.waiting_on_resume_data = True

    def write_torrentfile(self):
        """Writes the torrent file"""
        path = "%s/%s.torrent" % (
            os.path.join(get_config_dir(), "state"),
            self.torrent_id)
        log.debug("Writing torrent file: %s", path)
        try:
            self.torrent_info = self.handle.get_torrent_info()
            # Regenerate the file priorities
            self.set_file_priorities([])
            md = lt.bdecode(self.torrent_info.metadata())
            torrent_file = {}
            torrent_file["info"] = md
            open(path, "wb").write(lt.bencode(torrent_file))
        except Exception, e:
            log.warning("Unable to save torrent file: %s", e)

    def delete_torrentfile(self):
        """Deletes the .torrent file in the state"""
```

```
        path = "%s/%s.torrent" % (
            os.path.join(get_config_dir(), "state"),
            self.torrent_id)
        log.debug("Deleting torrent file: %s", path)
        try:
            os.remove(path)
        except Exception, e:
            log.warning("Unable to delete the torrent file: %s", e)

    def force_reannounce(self):
        """Force a tracker reannounce"""
        try:
            self.handle.force_reannounce()
        except Exception, e:
            log.debug("Unable to force reannounce: %s", e)
            return False

        return True

    def scrape_tracker(self):
        """Scrape the tracker"""
        try:
            self.handle.scrape_tracker()
        except Exception, e:
            log.debug("Unable to scrape tracker: %s", e)
            return False

        return True

    def force_recheck(self):
        """Forces a recheck of the torrents pieces"""
        try:
            self.handle.force_recheck()
            self.handle.resume()
        except Exception, e:
            log.debug("Unable to force recheck: %s", e)
            return False
        return True


    def cleanup_prev_status(self):
        """
        This method gets called to check the validity of the keys in the prev_status
        dict.  If the key is no longer valid, the dict will be deleted.

        """
        for key in self.prev_status.keys():
            if not self.rpcserver.is_session_valid(key):
                del self.prev_status[key]
```

File: Core.py

```
#
# core.py
#
from deluge._libtorrent import lt
```

```
import os
import glob
import base64
import shutil
import threading
import pkg_resources
import warnings
import tempfile


from twisted.internet import reactor, defer
from twisted.internet.task import LoopingCall
import twisted.web.client

from deluge.httpdownloader import download_file
from deluge.log import LOG as log


import deluge.configmanager
import deluge.common
import deluge.component as component
from deluge.event import *
from deluge.error import *
from deluge.core.torrentmanager import TorrentManager
from deluge.core.pluginmanager import PluginManager
from deluge.core.alertmanager import AlertManager
from deluge.core.filtermanager import FilterManager
from deluge.core.preferencesmanager import PreferencesManager
from deluge.core.autoadd import AutoAdd
from deluge.core.authmanager import AuthManager
from deluge.core.eventmanager import EventManager
from deluge.core.rpcserver import export

class Core(component.Component):
    def __init__(self, listen_interface=None):
        log.debug("Core init..")
        component.Component.__init__(self, "Core")

        # Start the libtorrent session
        log.info("Starting libtorrent %s session..", lt.version)

        # Create the client fingerprint
        version = [int(value.split("-")[0]) for value in deluge.common.get_version().split(".")]
        while len(version) < 4:
            version.append(0)

        self.session = lt.session(lt.fingerprint("DE", *version), flags=0)

        # Load the session state if available
        self.__load_session_state()

        # Set the user agent
        self.settings = lt.session_settings()
        self.settings.user_agent = "Deluge %s" % deluge.common.get_version()

        # Set session settings
```

```
        self.settings.send_redundant_have = True
        self.session.set_settings(self.settings)

        # Load metadata extension
        self.session.add_extension(lt.create_metadata_plugin)
        self.session.add_extension(lt.create_ut_metadata_plugin)
        self.session.add_extension(lt.create_smart_ban_plugin)

        # Create the components
        self.eventmanager = EventManager()
        self.preferencesmanager = PreferencesManager()
        self.alertmanager = AlertManager()
        self.pluginmanager = PluginManager(self)
        self.torrentmanager = TorrentManager()
        self.filtermanager = FilterManager(self)
        self.autoadd = AutoAdd()
        self.authmanager = AuthManager()

        # New release check information
        self.new_release = None

        # Get the core config
        self.config = deluge.configmanager.ConfigManager("core.conf")

        # If there was an interface value from the command line, use it, but
        # store the one in the config so we can restore it on shutdown
        self.__old_interface = None
        if listen_interface:
            self.__old_interface = self.config["listen_interface"]
            self.config["listen_interface"] = listen_interface

    def start(self):
        """Starts the core"""
        # New release check information
        self.__new_release = None

    def stop(self):
        # Save the DHT state if necessary
        if self.config["dht"]:
            self.save_dht_state()
        # Save the libtorrent session state
        self.__save_session_state()

        # We stored a copy of the old interface value
        if self.__old_interface:
            self.config["listen_interface"] = self.__old_interface

        # Make sure the config file has been saved
        self.config.save()

    def shutdown(self):
        pass

    def save_dht_state(self):
        """Saves the dht state to a file"""
        try:
```

```
            dht_data = open(deluge.configmanager.get_config_dir("dht.state"), "wb")
            dht_data.write(lt.bencode(self.session.dht_state()))
            dht_data.close()
        except Exception, e:
            log.warning("Failed to save dht state: %s", e)

    def get_new_release(self):
        log.debug("get_new_release")
        from urllib2 import urlopen
        try:
            self.new_release = urlopen(
                "http://download.deluge-torrent.org/version-1.0").read().strip()
        except Exception, e:
            log.debug("Unable to get release info from website: %s", e)
            return
        self.check_new_release()

    def check_new_release(self):
        if self.new_release:
            log.debug("new_release: %s", self.new_release)
            if deluge.common.VersionSplit(self.new_release) >
deluge.common.VersionSplit(deluge.common.get_version()):
                component.get("EventManager").emit(NewVersionAvailableEvent(self.new_release))
                return self.new_release
        return False

        @export
    def pause_torrent(self, torrent_ids):
        log.debug("Pausing: %s", torrent_ids)
        for torrent_id in torrent_ids:
            if not self.torrentmanager[torrent_id].pause():
                log.warning("Error pausing torrent %s", torrent_id)

    @export
    def connect_peer(self, torrent_id, ip, port):
        log.debug("adding peer %s to %s", ip, torrent_id)
        if not self.torrentmanager[torrent_id].connect_peer(ip, port):
            log.warning("Error adding peer %s:%s to %s", ip, port, torrent_id)

    @export
    def move_storage(self, torrent_ids, dest):
        log.debug("Moving storage %s to %s", torrent_ids, dest)
        for torrent_id in torrent_ids:
            if not self.torrentmanager[torrent_id].move_storage(dest):
                log.warning("Error moving torrent %s to %s", torrent_id, dest)

    @export
    def pause_all_torrents(self):
        """Pause all torrents in the session"""
        for torrent in self.torrentmanager.torrents.values():
            torrent.pause()

    @export
    def get_torrent_status(self, torrent_id, keys, diff=False):
        # Build the status dictionary
        status = self.torrentmanager[torrent_id].get_status(keys, diff)
```

```
    # Get the leftover fields and ask the plugin manager to fill them
    leftover_fields = list(set(keys) - set(status.keys()))
    if len(leftover_fields) > 0:
        status.update(self.pluginmanager.get_status(torrent_id, leftover_fields))
    return status

@export
def get_torrents_status(self, filter_dict, keys, diff=False):
    """
    returns all torrents , optionally filtered by filter_dict.
    """
    torrent_ids = self.filtermanager.filter_torrent_ids(filter_dict)
    status_dict = {}.fromkeys(torrent_ids)

    # Get the torrent status for each torrent_id
    for torrent_id in torrent_ids:
        status_dict[torrent_id] = self.get_torrent_status(torrent_id, keys, diff)

    return status_dict

@export
def get_session_state(self):
    """Returns a list of torrent_ids in the session."""
    # Get the torrent list from the TorrentManager
    return self.torrentmanager.get_torrent_list()

@export
def get_config(self):
    """Get all the preferences as a dictionary"""
    return self.config.config

@export
def get_config_value(self, key):
    """Get the config value for key"""
    try:
        value = self.config[key]
    except KeyError:
        return None

    return value

@export
def get_config_values(self, keys):
    """Get the config values for the entered keys"""
    config = {}
    for key in keys:
        try:
            config[key] = self.config[key]
        except KeyError:
            pass
    return config
@export
def set_torrent_max_connections(self, torrent_id, value):
    """Sets a torrents max number of connections"""
    return self.torrentmanager[torrent_id].set_max_connections(value)
```

```
@export
def set_torrent_max_upload_slots(self, torrent_id, value):
    """Sets a torrents max number of upload slots"""
    return self.torrentmanager[torrent_id].set_max_upload_slots(value)

@export
def set_torrent_max_upload_speed(self, torrent_id, value):
    """Sets a torrents max upload speed"""
    return self.torrentmanager[torrent_id].set_max_upload_speed(value)

@export
def set_torrent_max_download_speed(self, torrent_id, value):
    """Sets a torrents max download speed"""
    return self.torrentmanager[torrent_id].set_max_download_speed(value)


#add the speed limit function for peer
@export
def set_peer_max_download_speed(self, torrent_id, ip, value):
    return self.torrentmanager[torrent_id].set_peer_download_speed(ip, value)

@export
def set_peer_max_upload_speed(self, torrent_id, ip, value):
    return self.torrentmanager[torrent_id].set_peer_upload_speed(ip, value)
```

References

1. BitTorrent Specification Wiki.
http://wiki.theory.org/BitTorrentSpecification/.

2. B. Cohen. Incentives Build Robustness in BitTorrent. In Proc. Of the Workshop on Economics of Peer-to-Peer Systems(P2PEcon'03), Berkeley, CA, June 2003.

3. PlanetLab platform. http://www.planet-lab.org.

4. N.Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on Cooperation in BitTorrent Communities. In Proc. Of the Workshop on Economics of Peer-to-Peer Systems(P2PEcon' 05), Philadelphia, PA, August 2005.

5. A.R. Bharambe, C. Herley, and V.N.Padmanabhan. Analyzing and Improving a BitTorrent Network's Performance Mechanisms. In Proc. Of Infocom'06, Barcelona, Spain, April 2006.

6. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like System. In Proc. Of IMC'05, Berkeley, CA, October 2005.

7. M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Monthes in a Torrent's Lifetime. In Proc. of PAM's 04. Antibes Juan-les-Pins, France, April 2004.

8. S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding. In Proc. of the Workshop on Economics of Peer-to-Peer Systems(P2PEcon'05), Philadelphia, PA, August 2005.

9. T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In Proc. of HotNets-V, Irvine, CA, November 2006.

10. J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent P2P file-sharing system: Measurements and Analysis. In Proc. of IPTPS' 05, Ithaca, NY, February 2005.

11. D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In Proc. of SIGCOMM'04, Portland, OR, August 30-September 3, 2004.

12. J. Shneidman, D. Parkes, and L. Massoulie. Faithfulness in Internet Algorithms. In Proc. of the Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems(PINS' 04), Portland, OR, September 2004.

13. Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, February 2005.

13. Y. Tian, D. Wu, and K. W. Ng. Modeling, Analysis and Improvements for BitTorrent-Like File Sharing Networks. In Proc. of Infocom'06, Barcelona, Spain, April 2006.

14. M. Izal, G. Urvory-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Monthes in a Torrent's Lifetime.

15. Libtorrent. Retrieved from http://www.rasterbar.com/products/libtorrent/ on March 17, 2011.

16. Deluge. Retrieved from http://www.deluge.com. on March 16, 2011.

7. A. Fettig. Twisted Network Programming Essentials. O'Reilly, October 2005.

8. Feldman, M., Lai, K. Stoica, Ion. Chuang, John. Robust Incentive Techniques for Peer-to-Peer Networks. EC'04, May 17-20, 2004, New York, USA.

9. The Internet Engineering Task Force Request for Comments 5694. Retrieved from http://tools.ietf.org/search/rfc5694 on February 23, 2011.

12. Isdal, Tomas. Using BitTorrent for Measuring End-To-End Internet Path Characters. Trita-CSC-E 2006:148, ISSN-1653-5715, 2006.

13. Jiajun Wang, Chuohao Yeo, Vinod Prabhakaran, and Kanna Ramchandran. "On the role of helpers in peer-to-peer file download systems: design, analysis and simulation." In IPTPS, 2007.

14. M. Sirivianos, J. H. P. X. Yang, and S. Jarecki. "Dandelio: Cooperative Content Distribution with Robust Incentives." In USENIX, 2007.

15. Jeffrey Shneidman and David C. Parkes. "Rationality and Self-Interest in Peer to Peer Networks. " In IPTPS, 2003.

17. L.Cherkasova and J.Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks (USITS 2003)", In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, March 2003.

18. M. Sirivianos, J. H. Park, X. Yang and S. Jarecki. "Dandelion: Cooperative Content Distribution with Robust Incentive. ". In USENIX, 2007.

19. Kaune, S.; Tyson, G.; Pussep, K.; Mauthe, A.; Steinmetz, R.; “The Seeder Promotion Problem: Measurements, Analysis and Solution Space”. ICCCN, 2010 Proceedings of 19th Internaiontla on Digital Object Identifier.

20. Deluge Software. http://www.deluge.com.