Analysis and Simulation of Incentives to Seed in BitTorrent

Robbie Hott
Sean Krems

Abstract

Peer-to-peer (P2P) networking has become the preferred method of file sharing among users. BitTorrent, one such network, has emerged as the leader in peer-to-peer file sharing. BitTorrent has two classes of peers: seeders, those who share a file, and leechers, those who download a file. Seeders are the backbone of the BitTorrent network. There, is, however, little or no incentive to encourage peers to seed. We have developed a simulation of a P2P network based on the BitTorrent model. By implementing a variety of different incentive mechanisms within the framework of our simulation, we have analyzed the effectiveness of these incentives to seed. In addition, we have proposed a new incentive mechanism which has also been implemented within our simulation.

1 Introduction

Peer-to-peer (P2P) networking has emerged as the preferred method of file transfers. It is estimated that nearly 35% of all Internet traffic is P2P [10]. A P2P computer network relies primarily on the computing power and bandwidth of the participants in the network rather than placing the burden on a relatively low number of servers. This approach differs from the traditional client-server technique in that a peer is both a producer and a consumer of the service. A peer can generate a workload while also providing capacity to process the workload requests of other peers. In addition, a peer's lifetime is not always constant; a peer may be active in the system, contributing to the capacity and requests to the system, or may go off-line and be removed entirely from the system. In the client-server model, a server can always provide service as long as it operating. However, P2P only works if other peers are available; if there are no peers, there is no network. The topology of each model is shown in Figure 1.

BitTorrent is currently one of the most popular peer-to-peer networks. BitTorrent was designed to facilitate fast downloads of popular files. In order to achieve this goal, large files are divided into pieces of size 256 KB each. The set of peers attempting to download the file do so by connecting to several other peers simultaneously and downloading different chunks of the file from different peers. BitTorrent uses a centralized software called a tracker which is contacted first when a file is requested. The tracker then returns a list of peers that have the file. The peer determines which file fragments it needs and begins to download. While the peer is downloading, the user also becomes an uploader or seeder. In addition, when the download is complete, the downloader automatically becomes a seeder. In this way, files can easily and quickly propagate through the system. To share a file or group of files through BitTorrent, clients first must create a torrent. A torrent is a small file which contains metadata about the files to be shared and about the host computer providing

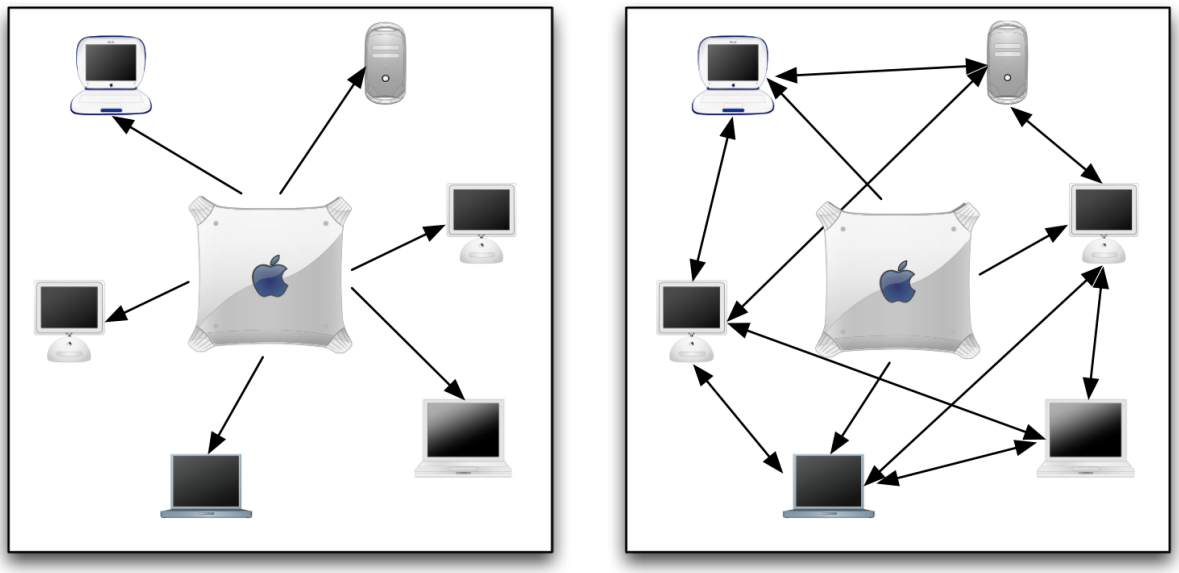

Figure 1: Client-Server (left) vs. Peer-To-Peer (right)

the file. The exact information contained in the torrent file depends on the version of the BitTorrent protocol. The files, however, do share some common characteristics. Each torrent file contains an *announce* section, which specifies the URL of the tracker, an *info* section which contains suggested names for the files and the length of the files, and a SHA-1 hash code to verify the integrity of the data they receive. In addition, all torrent files have the suffix `.torrent`. The torrent files are usually made available on the Internet.

Despite the relatively recent surge of research into peer-to-peer networks, BitTorrent still has two main issues: BitTorrent does not provide anonymity to peers and it does not provide effective incentives for users to seed a file. Peers are faced with the conflict between their eagerness to download files and their unwillingness to upload. Oftentimes, some peers are unwilling to upload at all. These peers are referred to as freeloaders. Freeloaders download files but do not upload files in exchange. In [6], it was shown that freeloaders finish downloads in about the same time as peers who contribute substantial amounts to the network. Clearly, freeloaders are not penalized properly and seeders are not appropriately rewarded. This lack of proper reward and punishment can cause the system to become inefficient and unreliable.

The desire of a peer to maximize its utility leads to three central questions of this paper. First, what is the best strategy that maximizes the utility of an individual peer? Second, what is the best incentive mechanism to encourage peers to seed? Third, what can be done to deter freeloaders? The backbone of BitTorrent are the peers who choose to seed files; without them, the network does not exist. Therefore, this paper will focus on incentives to seed.

The remainder of this paper is organized as follows. Section 2 presents some related work. Section 3 analyzes some of the currently proposed incentive mechanisms, and Section 4 introduces our proposed mechanism. In Section5, we explain the simulation developed

to study incentives to seed in BitTorrent, and the results are presented in Section 6. The future work of this research is discussed in Section 7. Finally, the conclusions are discussed in Section 8.

2 Related Work

Several research studies have analyzed existing BitTorrent deployments or used analytical modeling [12] to characterize the properties of the BitTorrent protocol and to improve its performance. In [5], tracker logs and an instrumented client are used to analyze the behavior of 180,000 peers in a torrent distributing Linux RH9 over five months. The download and upload rates of their instrumented peer are positively correlated, illustrating the effectiveness of the incentive mechanism. A very popular BitTorrent site is monitored and reported on its observed characteristics including data availability and integrity, flashcrowd handling, and download performance in [11]. The authors of [12] use a simple fluid model to study steady state performance for a BitTorrent-like system. They prove that, under certain conditions, a Nash equilibrium exists, and derive analytical upper bounds for freeloading.

Much research has also been devoted to providing incentives to seeders without degrading the performance of other users. The approaches and methods of provide incentives to seeders vary greatly. Some websites provide incentives to seeders. For example, the software from `Yabtuc.org` consists of a website with an integrated tracker which the behavior of peers related to seeding. If peers do not upload a sufficient amount, their access is temporarily denied. This system, however, is very centralized due to the integration of the website and tracker at a single location.

In [1], several cooperation enhancements used by BitTorrent communities are discussed. These enhancements are sharing-ratio enforcement and broadcatching. In sharing-ratio enforcement, peers register with the community and a metric called the sharing-ratio is maintained. The sharing-ratio is simply the total amount uploaded divided by the total amount downloaded. If the sharing-ratio of a peer falls below a minimum value, then the peer is restricted in downloading. Another site-level cooperation enhancement is a technique called broadcatching [1]. In order to advertise, some websites use RSS feeds listing new files available for download. In fact, some of these websites allow peers to automatically download a certain file as it becomes available. For example, a peer might configure BitTorrent to automatically download past episodes of a favorite television show. The logic behind broadcatching is that since files are downloaded automatically, peers might leave their computers for an extended length of time. While the peer is away, the completed file is seeded to other peers. In [1], it is shown from log traces that these site-level enhancements do improve cooperation within BitTorrent.

It is suggested in [14] that there can be no incentives to seed in BitTorrent. It is instead suggested that inter-torrent cooperation among peers is a better alternative. This policy follows the idea that peers only seed if there is some benefit to be gained from it. Basically, peers download small pieces of a file, even if the file is not desired, and then upload to peers in its system. In exchange, all other peers in the system do the same. The idea also involves matching peers in the same network and sharing bandwidth among peers. There is also a somewhat trivial approach of allowing peers with high uploading rates to have

correspondingly high download rates as well.

3 Incentive Mechanisms

In this section, we introduce the incentive mechanisms that are explored in this paper. In some cases, we have implemented the incentive mechanism within our simulation to determine its effectiveness.

3.1 Points Mechanism

One of the most basic incentive mechanisms deployed in P2P networks is the points incentive mechanism. The Maze P2P file sharing operates using a point system. Using this mechanism, peers consume points by downloading files and earn points by uploading files. Download requests are queued according to their points total. The formula to determine queue placement is $RT - 3\log_{10} P$ where RT is the peer's request time and P is the peer's point total [8].

There are, however, several known issues regarding the points incentive mechanism in Maze. The first issue is to determine the proper amount of points to earn or lose for uploading and downloading, respectively. The simplest solution is to make the point values equal. That is, when one peer loses points another peer gains an equal number of points. Enforcement of this policy, however, creates a disadvantage for some peers. If a peer has a slow upload speed or unpopular files, then that peer will not accumulate as many points as other peers. As a result, that peer will effectively be prevented from downloading any files. To correct this problem, more upload points are given than download points. In this way, uploading is encouraged and all cooperating peers have a chance to download files. Peers, however, can abuse this method by two techniques referred to as pairwise collusion and spam account collusion. In pairwise collusion, two peers exchange large amounts of data to increase the point totals of each other. In spam account collusion, a peer registers a number of dummy accounts and then exchanges data between the main account and dummy accounts to increase their point total. These techniques, however, can be countered by an algorithm called EigenTrust which can be explored more in [7]. The issue of trust in P2P networks and BitTorrent is beyond the scope of this paper. Both equal point incentives and unequal point incentives have been implemented within the framework of our simulation.

Another issue with the points incentive mechanism is bootstrapping for new users. Peers initially must have enough points to download files so that they can be shared in the future. In the current implementation of Maze, peers can download 3 GB of data before its download speed is throttled which is discussed in the next section. To avoid this problem, our simulation assigns all peers the same number of points at the start.

3.2 Throttling

Throttling a peer's bandwidth is another proposed incentive mechanism. This idea has been adapted from the policies of several Internet Service Providers (ISP) in order to limit the amount of P2P traffic on the network. Throttling, however, is not really an incentive

mechanism in the sense that it provides little reward to those who seed. This mechanism instead is a policy to punish those who do not cooperate. The download speed of a peer is decreased proportionally to its amount of uploading. That is, the more a peer uploads files the faster it will be able to download files in return. This mechanism is effective at decreasing the throughput and efficiency of freeloading, but it does not prevent it entirely. Patient freeloading peers can complete as a many downloads as they wish. The process takes much longer than those who upload sufficient amounts but all downloads eventually finish. In the Maze P2P file sharing network, new peers are allotted to download at least 3 GB. After the limit has been reached, Maze throttles the download speed to 300 kbps. The throttling policy implemented within our simulation is formula-based. The formula is that if a user shares files, they will get $\left(1 - \frac{1}{\text{uploads}-\text{downloads}+2}\right) * 100\%$ of their potential bandwidth, and if they freeload, they will only get 1/10 of their potential bandwidth.

3.3 Tit-for-Tat

In [6], a game-theoretic framework is applied to the development of an incentive mechanism referred to as tit-for-tat. This mechanism is closely related to the prisoner's dilemma and the iterated prisoner's dilemma from game theory. In the prisoner's dilemma, suppose two people want to exchange their items. The exchange of their items is atomic; one person must decide whether to send their items or not without any knowledge of what the other person will do. If they both send their items, then they both benefit from the exchange. If the decision differs, then one person will have both items and the other will have nothing. Any rational person should choose to not send their items and have both items. Using this argument, it is apparent that cooperation should not be possible. The rationality, however, changes when the two people are concerned about future transactions. The repeated exchange is referred to as the iterated prisoner's dilemma. In [6], the iterated prisoner's dilemma is related to BitTorrent. In [4], computers tournaments were conducted to test the rationality of different players. In all tournaments, the winning player used a tit-for-tat strategy. The tit-for-tat strategy always cooperates in the first exchange. After the first exchange, it does what the other player did in the previous move.

An incentive mechanism based on the results obtained in [4] is proposed in [6]. The mechanism is based on the tit-for-tat strategy. Peers maintain the upload amount u and the download amount d for each peer. The deficit of each link to a peer is defined as $u - d$. A peer ensures that the deficit of every link is restricted to a bound such that $u - d \leq f * c$ where c is the size of a fragment and f is referred to as the nice factor. Using the tit-for-tat strategy, a peer can upload an even amount to all other peers. The nice factor basically represents a value that represents the amount that a peer is willing to risk in order to establish cooperation. Other peers can possibly take advantage of peers with large nice factors, but they will benefit more if they cooperate. A comparison of this strategy to the current BitTorrent strategy can be found in [6].

3.4 Optimistic Unchoking

BitTorrent employs two different techniques: choking (not uploading) and unchoking (uploading). The current choking algorithm implemented in BitTorrent always unchokes a fixed number of peers; the default number to unchoke is four. It is, however, important to choose the right peers to unchoke so that the built-in congestion control of TCP can utilize the full upload capacity [2]. The process of choosing which peers to unchoke is based soley on the the peer's current download rate. However, this method suffers from having no method of discovering if other connections are better than the current connections being used. As a result, after the peers with the highest download rate are unchoked, all other peers are choked except for one that ia allowed by an incentive mechanism called optimistic unchoking. The purpose of optimistic unchoking is simply to find a link with a better download rate. Peers are able to recalculate and find a link for a sufficiently long period so that the link can be placed on the unchoking list of the other peer. If the peer downloads from this link at a higher rate than some of the previously unchoked links then it replaces the old link. Otherwise, a round-robin fashion is used to choose another link for the next iteration of optimistic unchoking. In [6], it is suggested that continuous 30 second intervals are enough time to perform optimistic unchoking.

However, many believe this incentive mechanism does not foster cooperation within BitTorrent. Four common properties of a cooperative peer are defined in [6].

1. Peers are nice. They cooperate as long as other peers do as well.

2. Peers are retaliatory. They stop cooperating immediately if other peers stop. This prevents peers from being exploited.

3. Peers are forgiving. They cooperate if other peers want to cooperate again. This helps reestablish trust within the network.

4. The behavior of each peer is clear and predictable. Peers signal to other peers that they act reciprocally. This can help foster cooperation among other peers.

The optimistic unchokimg method does not promote these properties because it abandons a peer upon finding one with a better download rate. Therefore, we conclude that this incentive mechanism is not effective within BitTorrent.

3.5 Credit/Debt Method

The concept employed by credit card companies has been adapted as an incentive mechanism for BitTorrent in [9]. A peer maintains data about all other peers it connects. Each peer keeps track of the number of fragments sent to another peer and the number of fragments received from another peer. The difference of these two numbers expresses a monetary relationship among the peers. If the difference between the two numbers is negative, then a peer owes a debt to another peer. If the difference between the two numbers is positive, then a peer has a credit from another peer. These values are pairwise. That is, if peer A has a credit from peer B, then peer B owes a debt to peer A. So if a peer is owed a debt by another peer, then that peer can download any file utilizing all available upload bandwidth until the

debt is repaid. The total number of fragments that a peer has received from another peer is a measure of the confidence that it has in its peer [9]. One possible issue not discussed in [9] is how to deal with situations in which multiple peers are requesting files because of owed debts. In this case, how will the upload bandwidth be distributed among the requesting peers? We suggest that the upload bandwidth allotted to a peer is proportional to the debt owed to that peer and to the size of the file requested. In this way, peers possesing the most debt are satisfied first along with peers who request smaller files which presumably finish first.

The debt and confidence values are used by all peers to discriminate freeloaders from other cooperating peers. In this way, peers are able to refuse service to peers who attempt to freeload. A value known as the debt threshold is set to determine how freeloading is defined. As a result, requests for a file are agreed upin unless the debt value exceeds the debt threshold value. In [9], the debt threshold is increased dynamically as a function of the confidence value. This allows more leniency to peers that have performed well over time. This technique is referred to as pairwise trade.

One issue, however, arises from pairwise trading. If a peer A wants a fragment from peer B and peer B does not owe a debt to peer B, then peer A cannot download the fragment. This problem is solved by a technique introduced in [9] referred to as transitive trading.

Transitive trading allows a peer to take advantage of its earned credit to obtain fragments from peers which might otherwise refuse to serve it. This technique works by identifying a debt-based path from itself to another peer that has the desired fragment. Each peer in the debt-based path has credit with the next peer and is below the debt threshold of the next peer. In this way, a peer can obtain a file by indirectly calling upon its owed debts. In [9], it is suggested that locating a debt-based path can be achieved efficiently in a structured overlay network like Pastry [13]. This can be performed by looking up the desired fragment using debt as the proximity metric.

Once a debt-based path is identified, the following simple protocol can be used to guarantee a secure exchange of data and credit along the trading chain. We assume that peer B has a file that peer A wants to download. The file is first broken into a number of fixed-size blocks. Peer A sends a message through the trading chain to peer Z to request a specific block. After the block is sent, peer A then retransmits requests along the same trading chain. Nodes along the path can incrementally adjust their debt and credit tables when they forward these requests.

4 Proposed Incentive: Three Strikes Throttling

Through analysis and simulation of existing BitTorrent incentive mechanisms, we propose a new mechanism referred to as Three Strikes. The idea of Three Strikes Throttling is based on a model of the U.S. criminal justice policies. According to [15], the three strikes laws are statutes enacted by state governments in the United States which require the state courts to hand down a mandatory and extended period of incarceration to persons who have been convicted of a serious criminal offense on three or more separate occasions. We apply this concept to freeloading. This mechanism can actually be thought of as a hybrid incentive mechanism between the throttling mechanism and the three strikes laws. If a

peer is freeloading, it is initially ignored. If a peer freeloads again, it is ignored once again. However, if a peer freeloads three or more times, then its download speed is immediately throttled to a very small value. We initially thought that the freeloading peer should be restricted from downloading entirely. However, we believe this policy would substantially destabilize the network. The choice of throttling instead of blocking allows the network to be more reliable and improves performance of all peers.

Unlike the U.S. criminal courts, the Three Strikes Throttling policy is, however, extremely forgiving to those who attempt to freeload. Prior freeloading attempts are forgiven and maximum bandwidth capabilities are returned when the freeloading peer begins to upload. By implementing this mechanism, we believe that freeloading can effectively be deterred without degrading the performance of all peers.

The Three Strikes Throttling incentive mechanism was implemented within the framework of our simulation to analyze its performance and effectiveness at punishing freeloading. It is based on the Throttling method described earlier, and all of the formulas remain the same. The main difference is that for Three Strikes Throttling, a freeloading peer gets three chances to cooperate before their bandwidth is severely restricted.

3 strikes you are out

combined methiod, proportional amount of points no freeloading allowed

5 Simulation of BitTorrent-based P2P Network

We have implemented a next-event simulation, in C, based on the BitTorrent File Distribution Network. Before beginning our simulation, we make the following assumptions:

1. Peers are randomly distributed,
2. Peers can utilize all of their available bandwidth for file sharing,
3. Peer download bandwidth will range from 28.8 Kbps dial-up modems to 5Mbps cable connections,
4. Peer upload bandwidth will range from 13.3 Kbps dial-up modems to 2.5Mbps cable modems,
5. Peers want to attain all the files in our network.

Our simulation allows us to define some constants before we start. We can define the total number of peers in our system, the range of file sizes, the number of files in the world, and the time the simulation will run, and the maximum number of downloads a user can have at the start of the simulation. To ease the visibility and readability of the graphs and results, we have chosen a total of 10 peers, file sizes between 1 and 100 MB, 50 total files with peers starting with under 10 initial downloads, and the simulation running for 100 seconds.

The simulation starts with some activity already happening, so we get a more realistic result, rather than having all users start into the system at time $t = 0$ with no files and no downloads. We first assign a random number of downloads to each peer, between 0 and the value set for maximum initial downloads. Then we give each peer a random number of

previous uploads, between 0 and their number of downloads times the number of peers, since they could have seeded it to multiple other users.

After prior setup, more setup needs to be done. We set the peers' download speeds randomly between 28.8 and 5000 Kbps and their upload speed randomly between 13.3 Kbps and half their download speed. We also fix some peers to certain values to test our incentives. Peer 7 is our new user, with 0 downloads, 0 uploads, and 0 files attained. Peer 8 is our freeloader, having a random number of downloads, but who has not uploaded anything. Lastly, peer 9 contains every file in the system, and acts as our server in case a peer cannot find the file they are looking for.

For the actual running of the system, at each timestep, a peer will be chosen at random to look for a file that it does not currently have. Then it will search the network for a peer that has that file, first randomly, then linearly so that in the worst case scenario it will download the file from the last peer, peer 9. On the non-incentive driven case of the simulation, the downloader will achieve the minimum of their available download bandwidth and the uploaders available upload bandwidth. The download time will be set based on the file size divided by the download speed. When a download starts, the download is set up, and when it finishes, the downloaders number of downloads is incremented and the seeders upload count is increased. The system continues until time $t = MAX\_TIME$ is reached, when the simulation will stop.

5.1 Incentive Mechanisms

To implement the incentive mechanisms into this simulation setup, the code must be changed slightly. For the point, throttling, and three-strike throttling incentives, the same system can be implemented into the simulation.

Peers now keep track of an integer incentive that is their ability to download. At startup, the peers' incentive is set equal to their initial number of uploads. When a file download is initiated, the downloader's incentive is decreased. When a file download is complete, the seeder's incentive is increased.

For the point system, it works just as this, and if a peer's incentive is zero, they are not allowed to download anymore files, and therefore their incentive integer is never decreased to less than zero. For throttling, the incentive is allowed to be any value of integer, and if the peer's incentive is zero or greater, then they get $(1 - \frac{1}{incentive+2}) * 100\%$ of their available bandwidth. If the peer's incentive is less than zero, they may only attain 1/10 of their bandwidth. For three-strike throttling, the incentives work just as in the throttling method, however, if a peer's incentive integer is between -3 and -1, they are allowed full bandwidth to download a file. When a file is uploaded from that peer, their incentive integer is automatically reset to zero, resetting the number of times they may freeload before being penalized.

6 Results

Once complete, our simulation gave some surprising results for the different incentive mechanisms. They will be discussed in detail separately, then compared. We will discuss in detail

the results when no incentives are used, when the point system is used, when throttling is used, and when the three-strike incentive is used. At the beginning of our simulation, we can see the initial values for our peers' uploads and downloads in Figure 2. This is the number of uploads and downloads they have contributed at time $t = 0$. It can easily be seen that most peers have uploaded files, except for the fixed peers and one random peer. For our fixed peers, peer 7 the newcomer, peer 8 the freeloader, and peer 9 the server, they have uploaded no files. For peer 4, a non-fixed peer, the simulation randomly set her uploads to zero, but since her downloads start at 1 file, we can easily consider her a newcomer as well.

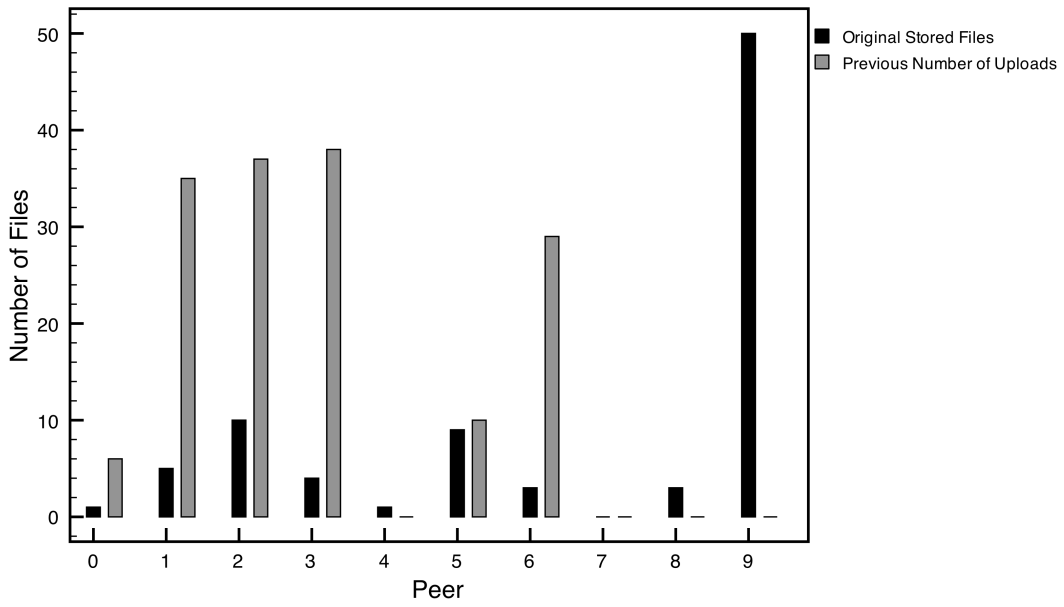

Figure 2: Number of original uploads and the original number of downloads for each peer. Peer 8 is our token freeloader, peer 7 is our newcomer, and peer 4 is a randomly generated newcomer since she starts with only 1 previous file.

Subsections 6.1 through 6.4 reference graphs that are slightly confusing to read. The black bar for each peer indicates the number of files that peer started with. The dark gray bar above the black one indicates how many files the peer has downloaded after 100 seconds. Similarly the light gray portion indicates the number of files the peer has attained after 1,000 seconds and the white bar is the total after 10,000 seconds. Since the number of files attained after a certain number of seconds running is cumulative, we show them on the same histogram bar. The number of files attained between two time intervals is the amount of that time's color showing for a certain peer.

6.1 No Incentives

Without incentives, we should expect the system to not favor seeders, freeloaders, or new peers, and from Figure 3 we can see that this holds true. After 100 seconds, most of the nodes

have downloaded 30 or more files, except for nodes 4 and 8, with 8 our freeloader. After 1,000 seconds, almost all the nodes have almost 50 files, except for peer 8, our freeloader. But after 10,000 seconds, all nodes have all the files. We can see that not having shared any files previously does not deter any peer from eventually and very quickly, relative to using incentives. Therefore, after 10,000 seconds, for the system to be as stable as a system with no incentives, we should expect most seeders to have almost all of the files.

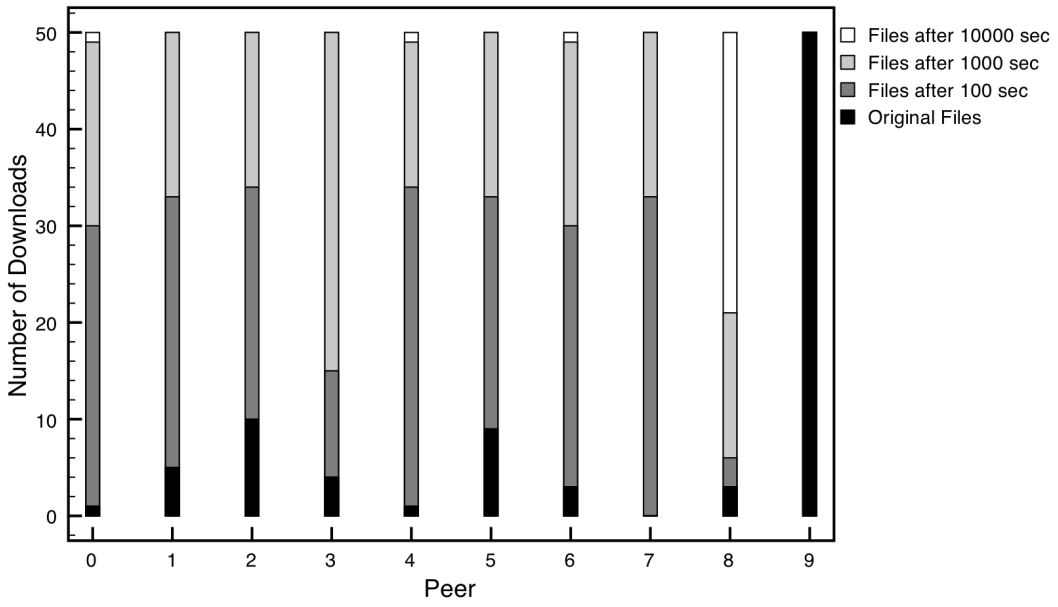

Figure 3: Files attained by peers 0-9 after 100 seconds, 1000 seconds, and 10000 seconds without an incentive mechanism.

6.2 Point System

With the point system, our simulation makes it very clear that it cuts the amount of files freeloaders have the ability to download. As we can see from Figure 4, after 10,000 seconds, freeloader peer 8 was only able to download about 15 files. However, this method truly hurts the newcomers, peers 4 and 7. Peer 7 is only able to download one file in the allotted time. Peer 4 gets more of a break because she starts with one original file and can therefore gain one extra upload point more easily.

The point system is also unstable for long amounts of time as peers who have more files attain a higher incentive, letting no one else download files due to lack of incentive points. After long periods of time, this would cause the system to shut down and not allow any more downloading among peers. This can be seen in the graph since very little downloading is done in the last 9000 seconds of the simulation compared to the other incentive mechanisms observed.

In our simulation, peer 9 gained more incentive points than any other node, and caused

the other nodes to stop downloading. However, peer 9 has no interest in downloading any files, which would allow other peers to download again.

After determining the ineffectiveness of this point system, we changed the values of the point system to give more points for uploading, allowing peers to download more files per upload. By giving two points per upload, the system did not perform significantly different than what was seen by giving one point per upload. Therefore the graph is not shown for two points per upload, however we saw a difference when peers were given four points per upload. Those results are shown in Figure 5.

Through our simulation, we have determined that multiple points per upload performs better than using an equivalent point reward/punishment mechanism, such as our initial point system. However, it still has the same drawbacks as the original version, except that peers who seed much more often will get all the files they want eventually. This incentive scheme still hurts both newcomers and freeloaders. Further, it discourages newcomers much more than freeloaders, since peers 4 and 7 do not get more than 2 files. Therefore, we can still argue that the point system is ineffective, even for multiple points per upload.

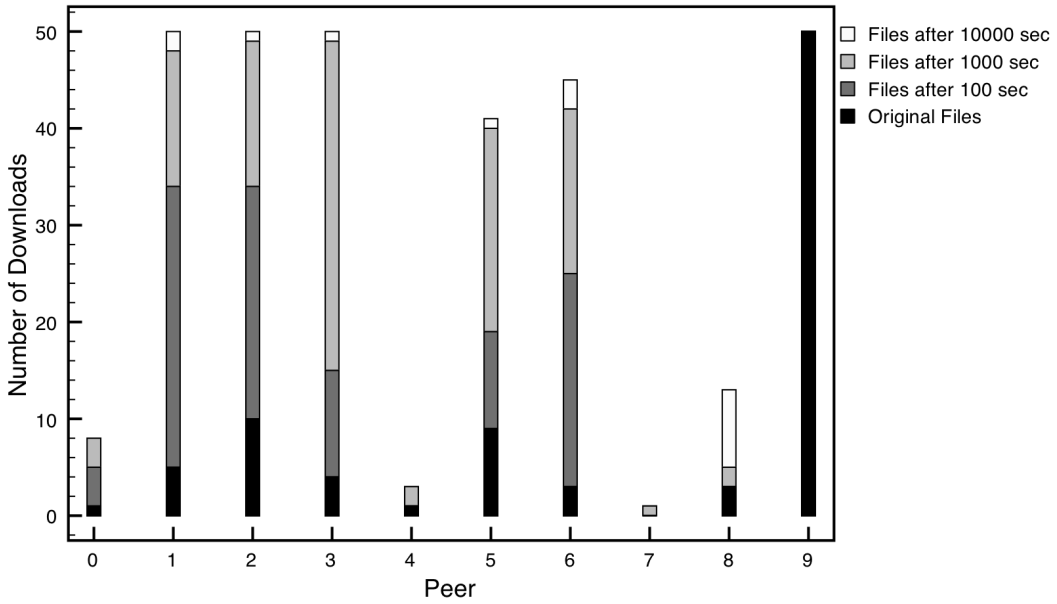

Figure 4: Files attained by peers 0-9 after 100 seconds, 1000 seconds, and 10000 seconds using the point system incentive with 1 point per upload and -1 point for download.

6.3 Throttling

Throttling, as we can expect, performs better than the point incentive scheme because it always allows a peer to download a file, no matter whether they have been freeloading or not. However, if a peer has been freeloading it will be penalized by a reduced file-download bandwidth. As seen in Figure 6, most of the peers attain almost all of the files in 10,000

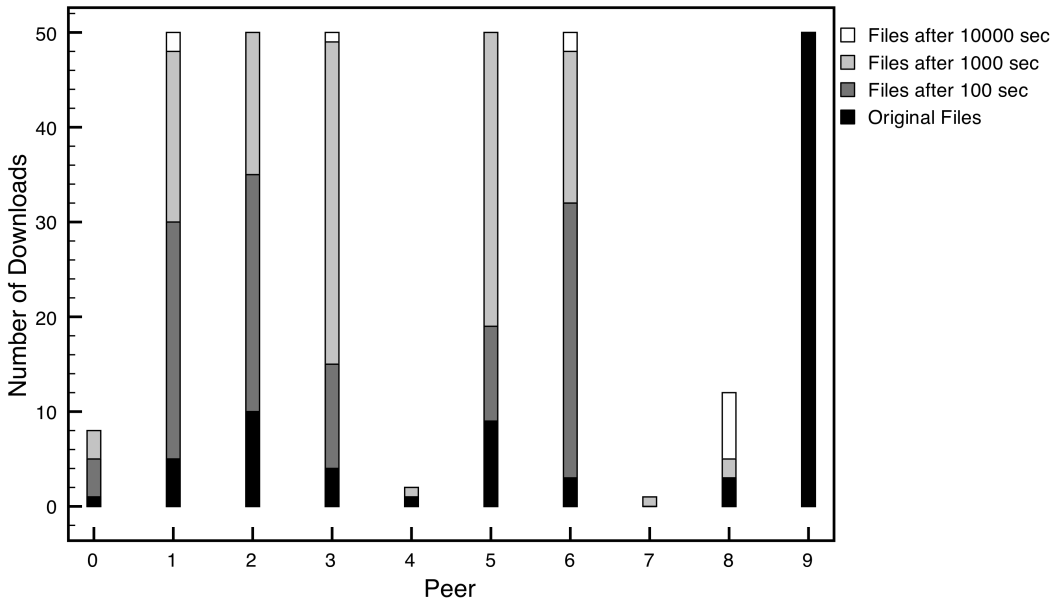

Figure 5: Files attained by peers 0-9 after 100 seconds, 1000 seconds, and 10000 seconds using the point system incentive with 4 points per upload and -1 point per download.

seconds, and even the new peers, peers 4 and 7, were able to download over 30 files. From this we can see that new users would not be penalized so much that they will not be able to download files. Also, our freeloader, peer 8, was still able to download the same number of files as it did in the point system. This scheme is much more stable than the point system because all of the peers in the point system that could only download 8 files or less can now download over 30 files each. Also, this system will never halt due to inability of peers to download in the same manner that the point system will.

6.4 Three Strike Throttling

Three Strike Throttling is a minor change to throttling which gives freeloaders and new peers generosity by allowing them download 3 files at full bandwidth before cutting their bandwidth to 1/10 of their usual bandwidth. It performs almost identical to the throttling incentive, as can be seen in Figure 7. It does, however, let peers download a few extra files versus regular throttling.

This method of incentive also maintains system stability in the same way as the initial throttling approach.

6.5 Comparison

As we have seen from these different methods for imposing incentives against freeloading, it is apparent that a few of them surpass others in stability and ability of a peer to download files. Clearly, the point system is the worst incentive scheme simulated because it bounds the

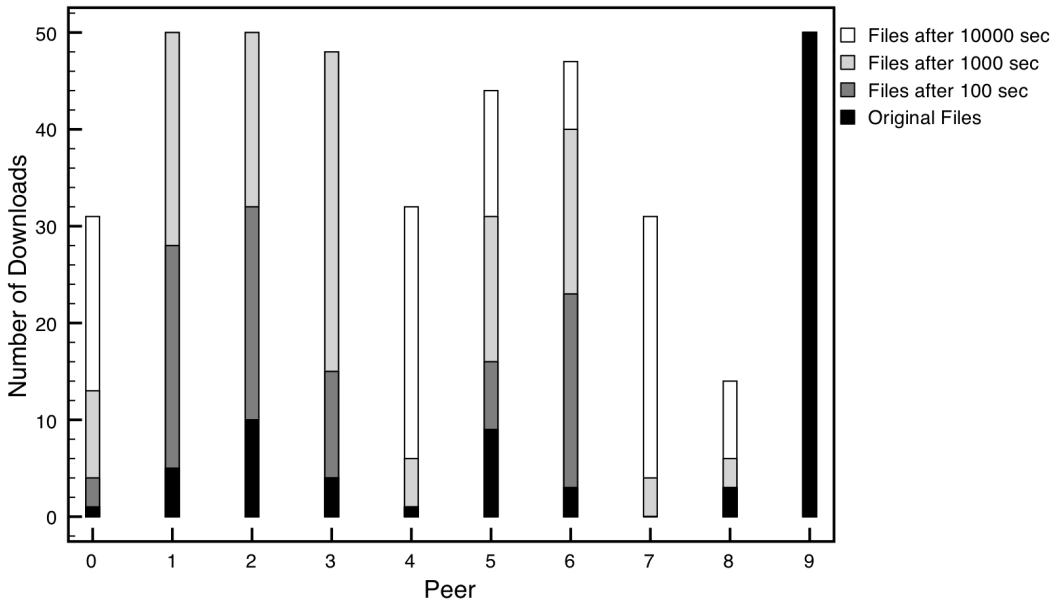

Figure 6: Files attained by peers 0-9 after 100 seconds, 1000 seconds, and 10000 seconds using the Throttling incentive.

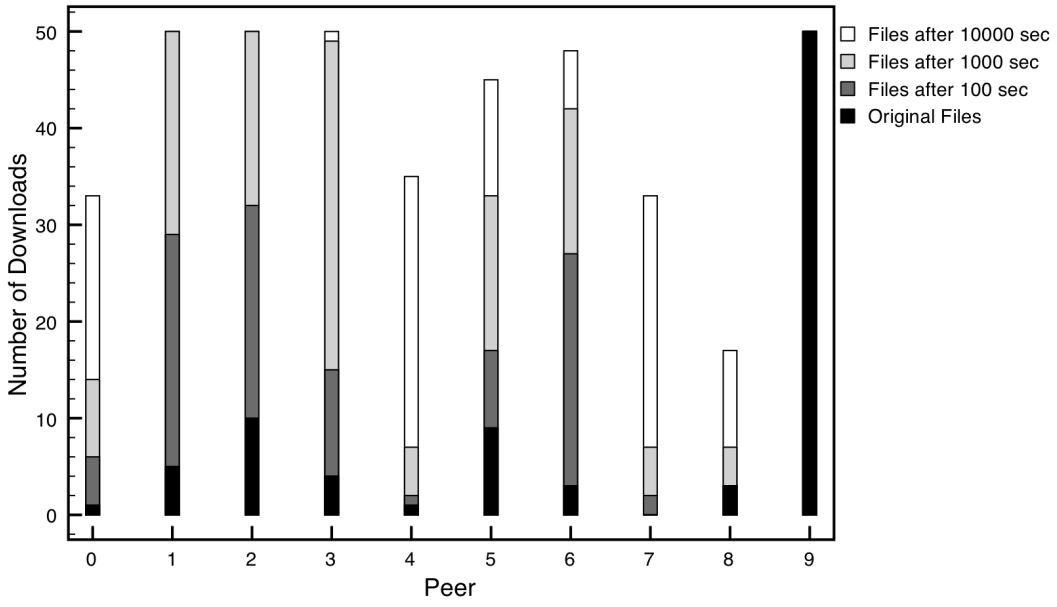

Figure 7: Files attained by peers 0-9 after 100 seconds, 1000 seconds, and 10000 seconds using the Three Strike Throttling incentive.

amount of files shared in the system. Throttling appears to be the best incentive mechanism we have looked at, having a similar overall appearance of the original simulation without incentives implemented, except that peer 8, our freeloader, is punished while others are not punished as much. Those that upload more are successfully allowed to download almost as if there were no incentive mechanism, but those who download less are penalized with the ability to download fewer files, without the entire peer-to-peer system crashing. Three-strike throttling enhances this by giving freeloaders more benefits, and therefore enhances the stability of the system even more.

7 Future Work

The future of this research is best described in several steps. The first would be to improve the capabilities of our simulation while studying and analyzing many more incentive mechanisms. The implementation of new incentive mechanisms within the framework of our simulation can help us better understand how it works. Secondly, the best way to show how well these incentive mechanisms work would be to show them in practice. We would want to find implemenations of these incentive mechanisms in real-world BitTorrent clients or implement them into a real-world BitTorrent client. Then, a better and more comprehensive comparison on how the incentive mechanisms work in practice could be performed. Lastly, it would also be necessary to provide an analytical model of our proposed mechanism as well as other incentive mechanisms to obtain a mathematical comparison.

The progression of the Three Strikes Throttling incentive mechanism is as follows: we must first show in simulated environment that our new incentive mechanism works. This has already been completed in this paper. We next must show mathematically that it is effective. Eventually, we would like to implement the Three Strikes Throttling incentive mechanism in a real-world BitTorrent client. This incentive mechanism seems less restricting than the other mechanisms we compared, but a study into how it could be made better would be warranted in the mathematical model or tweaked into the real-world implementation. The precise percentage of bandwidth a peer attains after freeloading and the formula for unthrottling of bandwidth to seeders could be tested using multiple different values to enhance Three Strikes Throttling to be even better on the network and peers.

8 Conclusions

We have studied several BitTorrent incentive mechanisms including optimistic unchoking, tit-for-tat, points to download, and a technique based on credit and debt. A simulation of a P2P network based on BitTorrent has been implemented. In order to study the effectiveness of cooperation in BitTorrent, several incentive mechanisms have been implemented within the framework of our simulation. Many of the mechanism adequately reward peers that seed but do not properly punish peers that freeload.

We have also proposed a new mechanism based on the three strikes laws called Three Strikes Throttling. Using this technique, freeloading is allowed to a certain threshold. When that threshold is reached, the freeloading peer is throttled to a very low download speed. However, the freeloading peer is completely forgiven when they upload a file. We feel the

Three Strikes Throttling incentive mechanism properly rewards seeders and effectively punishing freeloaders enough while keeping the P2P system stable allowing peers to continue downloading files without system halt. Further analysis must be performed, however, to confirm our results.

References

[1] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner and Matei Ripeanu. Influences on Cooperation in BitTorrent Communities. SIGCOMM05 Workshops, August 2226, 2005, Philadelphia, PA, USA.

[2] Cohen, B. (2003): Incentives Build Robustness in BitTorrent. Proceedings of the Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA.

[3] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling Peer-Peer File Sharing Systems. INFOCOM, 2003.

[4] Douglas R. Hofstadter. The prisoner's dilemma computer tournaments and the evolution of cooperation. Scientific American, May 1983.

[5] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Months in a Torrents Lifetime. In Proc. Passive and Active Measurements (PAM2004), Antibes Juan-les-Pins, France, April 2004.

[6] Seung Jun and Mustaque Ahamad. Incentives in BitTorrent Induce Free Riding. In SIGCOMM, 2005.

[7] Sepandar D. Kamvar, Mario T. Schlosser and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the Twelfth International World Wide Web Conference, 2003.

[8] Qiao Lianz, Yu Pengx, Mao Yangx, Zheng Zhangy, Yafei Daix, and Xiaoming Lix. Robust Incentives via Multilevel Tit-for-tat.

[9] Tsuen-Wan (Johnny) Ngan, Animesh Nandi, Atul Singh, Dan S. Wallach and Peter Druschel. Designing incentives-compatible peer-to-peer systems.

[10] Pasick, Adam (November 4, 2004). LIVEWIRE - File-sharing network thrives beneath the radar. Yahoo! News.

[11] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In 4th International Workshop on Peer-to-Peer Systems (IPTPS05), Feb. 2005.

[12] Qiu, D., Srikant, R.: Modeling and performance analysis of Bit Torrent-like peer-to-peer networks. In: Proceedings of ACM SIGCOMM 2004, Portland, OR, USA (2004)

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems. In Proc. IFIP/ACM Intel Conf. on Distributed Systems Platforms.

[14] Ye tian, Di Wu and Kam-Wing Ng. Analyzing Multiple File Downloading in BitTorrent. Proceedings of the 2006 International Conference on Parallel Processing.

[15] Three strikes law found at `http://en.wikipedia.org/wiki/Three_strikes_law`

Appendix

8.1 Simulation

Below is contained the C source code for our simulation, without any incentives implemented.

```
#include "rngs.h"
#include "rvgs.h"

#include<stdio.h>

#define N 10
#define INITIAL_DOWNLOADS 10
#define MAX_FILES 50
#define MAX_TIME 100.0
#define START_DOWNLOAD 1
#define END_DOWNLOAD 2

struct nodes{
    int downloads;
    int uploads;
    int incentive;
    double files[MAX_FILES];
    float dl_speed;
    float ul_speed;
    double next_download;
};

struct times{
    int downloadee;
    int downloader;
    int file;
    double endtime;
};

int GetNextDownload(double* time, struct nodes* peer, double* download_time) {
    int i;
    int min_i = N-1;
    for (i=0; i<N; i++) {
        if (peer[i].next_download < peer[min_i].next_download) {
            min_i = i;
        }
    }
    *time = peer[min_i].next_download;
    *download_time = peer[min_i].next_download;
    SelectStream(5);
    peer[min_i].next_download = *time + Exponential(1.0);

    return min_i;
}


int GetNextDownloadee(int downloader, struct nodes* peer, int* file) {
    int downloadee;
    int i;
    int j;
    SelectStream(9);
    for(i=0; i<MAX_FILES;i++) {
        if (peer[downloader].files[i] == 0)
            break;
    }
    if (i == MAX_FILES) return -1;
    *file = i;
    SelectStream(6);
    downloadee = Equilikely(0,N-1);
    while (downloadee == downloader || peer[downloadee].files[i] == 0 || peer[downloadee].files[i] == -1) {
```

```
        downloadee = Equilikely(0,N-1);
    }

    return downloadee;
}

double doublemin(double a, double b) {
    if (a <= b)
        return a;
    return b;
}

int nextEvent(double *time, struct nodes* peer, double *download_time, int *file,
                    int *downloadee, int *downloader, struct times* ends, double files[]) {
    int i;
    double min = MAX_TIME+1;
    int mini;
    *downloader = GetNextDownload(time, peer, download_time);
    *downloadee = GetNextDownloadee(*downloader, peer, file);

    for (i=0; i<1000*N; i++) {
        if (ends[i].endtime < min) {
            min = ends[i].endtime;
            mini = i;
        }
    }

    if (min <= *download_time) {
        *downloader = ends[mini].downloader;
        *downloadee = ends[mini].downloadee;
        *time = min;
        *file = ends[mini].file;
        ends[mini].endtime = MAX_TIME +1;
        return END_DOWNLOAD;
    } else {
        for (i=0; i<1000*N; i++) {
            if (ends[i].endtime == MAX_TIME + 1) {
                break;
            }
        }
        ends[i].downloader = *downloader;
        ends[i].downloadee = *downloadee;
        ends[i].file = *file;
        ends[i].endtime = files[*file] / (doublemin(peer[*downloadee].ul_speed, peer[*downloader].dl_speed));
        return START_DOWNLOAD;
    }
}

int main (void) {
    struct nodes peer[N];
    struct times endtimes[1000*N];
    int i;
    int j;
    int k;
    int downloader;
    int downloadee;
    int file;
    int type;
    double download_time;
    double time = 0.0;
    double end_time = MAX_TIME;
    double files[MAX_FILES];
    int original[N];

    PlantSeeds(6768);
    SelectStream(0);

    for (i=0; i<MAX_FILES; i++) {
```

```
    SelectStream(7);
    files[i] = Uniform(1000, 100000);  // This will change how many downloads are finished.
}

for (i=0; i<1000*N; i++) {
    endtimes[i].downloader = -1;
    endtimes[i].downloadee = -1;
    endtimes[i].file = -1;
    endtimes[i].endtime = MAX_TIME + 1;
}

for (i=0; i< N; i++) {
    SelectStream(0);
    peer[i].downloads = Equilikely(0, INITIAL_DOWNLOADS);
    SelectStream(1);
    peer[i].uploads = Equilikely(0, N*peer[i].downloads); // should be some factor of N * downloads, like Equilikely(0, N*
    SelectStream(2);
    peer[i].dl_speed = Uniform(28.8, 5000);
    SelectStream(3);
    peer[i].ul_speed = Uniform(13.3, peer[i].dl_speed/2);
    SelectStream(4);
    peer[i].next_download = Exponential(1.0);


    peer[7].downloads = 0;
    peer[7].uploads = 0;


    for (j=0; j<MAX_FILES; j++)
        peer[i].files[j] = 0;

    for(j=0; j< peer[i].downloads; j++) {
        SelectStream(8);
        k = Equilikely(0, MAX_FILES-1);
        while(peer[i].files[k] != 0)
            k = Equilikely(0, MAX_FILES-1);
        peer[i].files[k] = files[k];
    }

    //set incentive
    peer[i].incentive = peer[i].downloads + peer[i].uploads;
    if (peer[i].incentive == 0)
        peer[i].incentive = 1;
}
//newbee
peer[7].downloads = 0;
peer[7].uploads = 0;
for (i=0; i<MAX_FILES; i++)
    peer[7].files[i] = 0;

//freeloader
peer[8].uploads = 0;

// has it all
peer[N-1].downloads = MAX_FILES;
peer[N-1].uploads = 0;
for (i=0; i<MAX_FILES; i++)
    peer[N-1].files[i] = files[i];

peer[N-1].incentive = MAX_FILES;

for( j=0; j<N; j++) {
    original[j] = peer[j].uploads;
}


while(time < end_time) {
    type = nextEvent(&time, peer, &download_time, &file, &downloadee, &downloader, endtimes, files);
```

```
        if (type == START_DOWNLOAD) {
            if (downloadee != -1) {
                peer[downloader].incentive--;
                peer[downloader].files[file] = -1;

            }
        } else if (type == END_DOWNLOAD) {
            peer[downloadee].incentive++;
            peer[downloader].downloads++;
            peer[downloadee].uploads++;
            peer[downloader].files[file] = peer[downloadee].files[file];

        }



    }

    for( j=0; j<N; j++) {
        printf("%d %d\n", j, peer[j].downloads);
    }

    return 0;
} // end main
```

Other versions of our simulation can be attained by email or web to save paper.