Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems

Lakshmish Ramaswamy
College of Computing
Georgia Institute of Technology
801 Atlantic Drive Atlanta GA 30332
laks@cc.gatech.edu

Ling Liu
College of Computing
Georgia Institute of Technology
801 Atlantic Drive Atlanta GA 30332
lingliu@cc.gatech.edu

Abstract

Most of the research in the field of peer-to-peer file sharing systems has concentrated on performance issues such as efficient file lookup, replicating files to improve file download speeds etc. However there is a new challenge that questions the very existence and usefulness of such systems in the form of "Free Riding". This paper studies the seriousness of the negative impact that free riding can have in a P2P file sharing system. We introduce the concept of utility function to measure the usefulness of peers to the system as a whole, and describe a scheme based on this concept to control free riding. A simple utility function is described to illustrate the scheme. We design and develop a simulation model to study various patterns of sharing behaviors among the peers in a file sharing community and their impact on the system. A set of experimental results is reported. The experiments indicate that the utility based free riding control can increase the lifetime of the system by 10 times, even with a simple utility function.

1 Introduction

The advent of peer-to-peer (P2P) file sharing systems heralds a new era in the field of Internet technology [2] [5] [7] [9]. While these systems alleviate the scalability problem that has dogged the client-server model, they present new data management problems.

It is widely believed that the success of P2P file sharing systems depends upon the quality of service offered by such systems. Accordingly most of the present research in P2P systems has been concentrated on issues such as efficient data placement, fast file lookup, data replication etc.

We argue that, in addition to the quality of service, there is another key aspect that impacts the success and continued sustenance of P2P systems. It is the quality of the data present in the system. For a file sharing system, no matter how excellent the lookup capabilities of a system are, or what file download speeds it offers, if the system does not have a large and growing number of interesting files, it will eventually fail to attract or retain users. Unfortunately, research on developing mechanisms to maintain or enhance the quality of data has not yet received much attention from the P2P research community.

This problem is exemplified by the phenomenon of free riding in many P2P file sharing systems. A recent study [1] on Gnutella file sharing system shows that as many as 70% of its users don't share any files at all. This means that these users use the system for free. This behavior of an individual user who uses the system resources without contributing anything to the system is the first form of the *Free Riding* problem. Such users are referred to as free riders. The study further indicates that not all file sharers share popular and desirable files. It shows that as many as 63% of the peers, who shared some files, never answered a single query. This implies that these file sharers did not share any desirable files. This is a second form of the *Free Riding* problem, wherein users share some files that are not useful.

The free riding problem affects the system in two significant ways. First, the number of files in the system becomes limited or grows very slowly. The number of popular files may become even smaller as the time goes by. This adversely affects user's interest in the system and they eventually pull out of the system. When users who share popular files pull out of the system, the system becomes poorer in terms of the amount of files shared. This is a vicious cycle and it may eventually lead to the collapse of the system. Second, if only a few peers share popular files, all the downloading requests are directed towards those peers. This causes those peers to become hot spots, overloading their machines and causing congestion on their network. Peers frequently experiencing CPU overloads or network congestion due to the P2P system may exit the system if it affects their other routine activities.

Though freeriding problem appears to be a new phenomenon in the field of information sciences, it has existed in community-sharing based fields of human endeavor for centuries. Economists have done comprehensive studies on the twin problems of *free riding* and *The Tragedy of Commons* [4]. Over-fishing in deep oceans, pollution in cities, and over use of pesticides are all recent illustrations of this age old problem.

In order to maintain the value and ensure the healthiness of a P2P file sharing system, there is a need for mechanisms that can help securing cooperation from its users in the form of sharing popular files. Surprisingly, none of the existing P2P file sharing systems, to our knowledge, has offered or incorporated mechanisms that encourage their users to share files with other users in the system.

In this paper we propose a utility function based scheme to control free riding in a P2P file sharing system. The main idea of the utility-based schemes is to create incentives to inspirit users to share interesting files. We identify three important utility factors for building fair incentives in the file-sharing context: the total number of files shared, the total size of data shared, and the popularity of the data shared. A unique feature of our scheme is the utility function that utilizes these three utility factors with a built-in reward/penalty mechanism. We report our initial experimental results, showing the benefits of our utility-based scheme compared with present systems, which have no control on free riding.

2 Scenario Based Analysis

Before introducing our utility based scheme, we take a closer look at the free riding problem in the context of a P2P file sharing system. We choose to use a typical free riding scenario and an extreme scenario when the peers share equal number of files to gain a general understanding of the user behavior and the free riding problem in the P2P file sharing systems. Our analysis shows that the health and the lifetime of a P2P file sharing system is significantly influenced by the behavior of its users.

Consider a community of P2P file sharing system users. Let the community have N peers. Now let us consider two different scenarios.

In the first scenario, let majority of the peers be free riders. These peers don't share any files at all. All the files available in the system are shared by a small number of users. For any file sharing system, the attraction for any user to stay in the system is the amount of new files he can download from the system. Suppose in a P2P system there are 2000 files, out of which a single peer P_1 has contributed 1000 files. In this situation, the utility of the system to peer P_1 is the 1000 files that are being contributed by the other users. After he downloads some of these 1000 files that are of interest to him, if there are no new files being added, he would feel the lack of files that are interesting to him and is likely to withdraw from the system. His withdrawal would result in the system loosing all the files owned by him. This causes the system to become suddenly poorer in terms of the number of files by a large amount. Major file sharers would all be in this state and would eventually leave the system, taking the files they shared with them. Hence it is only a matter of time before which the system would be devoid of popular files causing the expiry of the community.

There is another dimension to this problem that may go unnoticed at first sight. To explain this dimension of the problem, let us assume that the system has N peers, and out of the N peers N_f peers have uploaded some files for sharing. For simplicity of argument let us assume that all of these N_f peers have uploaded equal number of files and all these files are equally popular. On an average let there be R downloading requests per unit time in the system. It is easily seen that each of these N_f file sharing peers would have to serve $\frac{R}{N_f}$ requests per unit time. If N_f is small then each of these N_f peers would have to serve a large number of download requests. This may cause network congestion and CPU overloads on these N_f peers, which affects the overall performance of peer machines. Frequent network congestion and CPU overloads on the peer machine may prompt the user to withdraw from the system in order to shield their machines from CPU overload and network congestion. This accelerates the process of depleting the user community causing an early expiry of the system.

Consider the second scenario, where all users share almost equal number of files. In this scenario $N_f = N$ and therefore each peer will serve only a small number of requests at any point of time. Hence there will be much less concerns on network congestion and CPU overload problems. Further, if there are M files in such an equal sharing community, each peer would have $(M - \frac{M}{N})$ files not owned by itself, which the peer might be interested in downloading. In this scenario, each node would have almost equal number of un-owned files. Contrasting this with the previous scenario indicates that this is a healthy trend that sustains interest of the user community for a longer time period, thereby extending the lifetime of the system.

Therefore it can be concluded that free riding has a negative impact on the lifetime and evolution of P2P file sharing system. A natural question would be whether P2P system developers can do something at the data management level to discourage free riding? If so, what are the policy and implementation issues involved in exercising such a scheme? We address these questions in the next section.

3 Solutions to Free Riding Problem

We have illustrated how free riding is an unhealthy trend that has the potential to curtail the lifetime of a community of P2P users. In this section we describe some technical solutions to this problem.

3.1 Replication Based Scheme

One of the possible solutions that has been adopted by some P2P systems like Kazaa [7] is to replicate files at every peer that downloaded them. This scheme addresses one aspect of the free riding problem. It utilizes the disk space and the network bandwidth resources of the peer downloading a particular file to make an extra replica of the file available for future downloading requests. If there are l replicas of a file, the next download request is equally likely to be directed to any of the l peers holding the copy. This would reduce network congestion and CPU overloading problems that the small number of peers, who were the original owners of the shared files, would have otherwise experienced. However, this replication enforcement scheme doesn't address the more serious problem of the system not getting new files and becoming stagnant. Therefore, the system still has the danger of users becoming uninterested due to the lack of new files, leading to its eventual expiry.

3.2 Utility Function Based Schemes

In this section we discuss three utility-based schemes for controlling free riding in P2P systems. In designing these scheme we aim at encouraging users to share some of their own files with others in the community, which we believe is the key to the long life of any such cooperative community. The underlying idea in these schemes is modeling and measuring the usefulness of every user in the system to the community as a whole. We term this measure as the *Utility* of the user to the system. The aim of such a scheme is to encourage users to improve their own utility value. We believe that this encouragement can be in some form of reward to those users with high utility value and some form of penalization of users with little utility value.

We identify three important factors that reflect an individual user's usefulness to the system. They are (1) the number of files shared by a user, (2) the total size of the data a user has shared, and (3) the popularity of the files shared by the user. Accordingly utility functions can be designed incorporating one or more of these factors. We promote a general utility-based scheme that takes into account all three factors as the most appropriate utility-based scheme for controlling free riding. The first utility function only takes into account the total number of files. The second utility function is based on the sum total of the sizes of all the files contributed by individual users. The third utility function accounts for the number of files, the size of the files, and also the popularity of the files contributed by each user. We explain all the three schemes in detail and analyze their advantages and drawbacks.

Utility Function 1

The first utility function that we consider is based on number of files shared by each user. This would seem appropriate if the system aims to increase the number of files available to users. In concrete terms, any utility function based on the number of files would be similar to the one below.

Let $UList(P_i, T)$ represent the set of files shared by the peer P_i at time instant T and $DList(P_i, T)$ represent the set of files that have been downloaded by peer P_i till time T. The utility of peer P_i at time T represented by $U(P_i, T)$ is given by:

$$U(P_i, T) = \alpha \times |UList(P_i, T)| \quad (1)$$

Where α is a normalizing constant. It can be easily seen from the equation that the peers sharing a larger number would have a higher utility value than those peers sharing smaller number of files.

The free riding control mechanism for such a utility function would be to limit the maximum number of files a peer can download within one unit time to the utility value. The utility value of all peers should be re calculated and restored at the end of the unit time frame. This mechanism ensures that users who share larger number of files would be entitled to download larger number of files.

Utility Function 2

A variant of Utility Function 1 would be to take into account the file size along with the number of files shared. This would ensure that users who share a few larger files would be treated on par with the users who share large number of smaller files. The logic behind this argument is that the cost incurred (in terms of disk space and network bandwidth resources the P2P system uses) to two users, one of whom, say user A, shares 1 file of 10MB and another, say user B, who shares 2 files of 5 MB each is approximately the same. But the utility value allocated by the equation 1 to the user sharing two files is double that of the utility value allocated to the user sharing a single large file. This utility function aims to remedy the above said anomaly. Concretely, let $UList(P_i, T)$ and $DList(P_i, T)$ be defined as in the utility function 1. Let *size*(F) be the size of the file F in bytes. The utility of peer P_i at time T, denoted by $U(P_i, T)$ would now be:

$$U(P_i, T) = \beta \times \sum_{(\forall F_j \in UList(P_i, T))} size(F_j) \quad (2)$$

It is easily seen that this utility function measures the usefulness of the user P_i in accordance with the number of bytes she has shared with the community. The scheme limits the maximum number of bytes a peer can download per unit time to the current value of the utility function.

At the first glance both these utility functions seem to be reasonable. If the utility of the system is measured in terms of the total number of files or the total number of bytes available for download, then these two utility functions would be in direct correspondence with the usefulness measure of the system to its users. Unfortunately, it would be inadequate to measure the usefulness of a P2P file sharing system by sheer number of files or even the number of bytes available in the system. In order to be really useful to the community of users the system should not only contain lots of data but also contain lots of interesting, useful and popular data. For example if a music file sharing system has 200,000 music files most of which are very old and from artists who are not so popular then the system can hardly be deemed as successful. In comparison a system that has less number of files but a relatively larger number of popular files may actually be more useful to the community.

Utility Scheme 3

The discussion in the previous section illustrated that it is necessary to design utility functions that reflect the popularity of the files shared by each individual user. The free riding control mechanism would then encourage users not to just share more files but share more files that are popular with the user community.

The question that crops up immediately is how to measure popularity of files shared in the system? We observed that the popularity of a file should not be an attribute that can be pre-assigned when a file enters the system. Rather it should be measured by how many times a file was actually downloaded within some finite time period. It should also be appreciated that a file, no matter how popular it was when it entered the system, will gradually loose its popularity as it becomes old. By measuring a file's popularity using the number of actual downloads at regular intervals, we can capture this phenomenon too. Hence we can conclude that the actual number of times a file was downloaded not only indicates its relative popularity in the system but also reflects the change in the file's popularity with time.

Accordingly, we have designed a utility function that accounts for the popularity of files as measured by the number of times it was downloaded. The utility function is made up of two components which we call as *Reward* and *Penalization*. *Reward* measures a peers utility to the system and *penalization* measures how much of the system's resource the peer has used.

In concrete mathematical terms, the *Reward* for P_i, denoted by $R(P_i, T)$, is characterized as: as:

$$R(P_i,T) = \gamma \times \sum_{\forall F_j \in UList(P_i,T)} (k + DCount(F_j,T)) \times size(F_j) \tag{3}$$

In equation 3, $UList(P_i, T)$ represents the set of files shared by the peer P_i at time instant T, and $DCount(F_j, T)$ denotes the number of times the file F_j has been downloaded until the current time T. $Size(F_j)$ denotes the size of the file in bytes and γ is a normalizing constant. The term k is a constant bias for all files shared by all users. It can be seen that the reward value of an individual peer is proportional to the number of times his files have been downloaded by other peers. The reader would have observed that in equation 3, the term $DCount(F_j, T)$ is scaled by the size of the file. This is done for the reason of fairness.

The necessity of having a bias constant (represented by k in equation 3) is substantiated in the following argument. When a peer joins the group or shares a new set of files, the files wouldn't be downloaded immediately by any user. There is always a time lag that exists between the time when a file is shared and the time when it is first downloaded. Without this initial bias, a peer sharing very popular files may not have any reward points during the initial period due to this time lag. The free riding control mechanism (which is dealt with in detail in the next paragraph), prevents peers with zero utility points from downloading any files. Therefore even a peer sharing popular files might be restricted from downloading files from the system during the time lag. To rectify this anomaly we included a one time, initial bias to all files shared by any user.

The *Penalization* for a peer P_i in some sense captures how much of the systems resource the peer has utilized in the form of files downloaded by it till time T. In mathematical terms the *Penalization* for a peer P_i at time T , denoted as $PN(P_i, T)$, is given by

$$PN(P_i,T) = \sum_{\forall F_l \in DList(P_i,T)} size(F_l) \tag{4}$$

The utility value for peer P_i is defined as the difference between its reward and penalization, i.e., $U(P_i, T) = R(P_i, T) - PN(P_i, T)$. Therefore the utility of peer P_i at time T represented by $U(P_i, T)$ is given by:

$$\begin{aligned} U(P_i,T) = &\sum_{(\forall F_j \in UList(P_i,T))} \gamma(k + DCount(F_j,T)) \times size(F_j) \\ &- \sum_{\forall F_l \in DList(P_i,T)} size(F_l) \end{aligned} \tag{5}$$

In equation 5, $UList(P_i, T)$ and $DList(P_i, T)$ are defined as in equation 1. $size(F_j)$ and $DCount(F_j, T)$ are defined as in equation 3.

We now explain the free riding control scheme associated with the above utility function. The mechanism is very similar to the mechanisms previously discussed. Whenever a peer attempts to download any file from the system, the mechanism verifies whether its current utility value is higher than the size of the file it is attempting to download. The mechanism denies the request if the peer's utility value is lower than the file size. If the utility value is higher than the file size, the peer is permitted to download the file. However, upon downloading a file, the peer's utility value is reduced by an amount equal to that of the downloaded file size.

In addition, we decided to reward every user with some *Freebie* utility points at regular intervals, irrespective of whether she shares any files or not. We felt that this would be an encouragement for the users to join the P2P community and to continue in the community even at times when they cannot add new files. Some users might want to see whether the system has the kind of files they would like to download. Therefore in order to encourage users to stay on with the system the scheme allocates some freebie utility points to all users at regular intervals. In doing so, we are aware of the fact that allocating freebies permits some amount of free riding. But the amount of free riding can be determined and controlled by the system designers and maintainers. This is in contrast to the present systems where the free riding is uncontrolled and rampant.

This free riding control mechanism is simple to implement. Whenever a new peer joins the group, the system checks the number of files and the sizes of the files the peer brings along with it. Accordingly, the initial value of the utility is set as $U(P_i, T_0) = \gamma \times \sum_{(\forall F_j \in UList(P_i,T_0))} k \times size(F_j) + FreePoints$. Whenever another peer downloads file F_l from peer P_i, the utility value of P_i is incremented by $\gamma \times Size(F_l)$. If the peer P_i adds a new file say F_m, its utility value would go up by $\gamma \times k \times Size(F_m)$. If peer P_i downloads a file say F_n from any other user, the system reduces it utility value by $Size(F_n)$. At the end of regular intervals of time (hence forth referred to as epochs), the system allocates some (small amount of) freebies to all peers whose utility value has fallen below a preset threshold.

4 Experiments and Results

In this section we describe the simulations we performed and the corresponding results. It should be noted that we are dealing with situations which are quite different from the normal computer systems simulations. The most basic difference between traditional computer systems and our simulation is that the former attempts to model a mechanical or a logical object like computer system, network behavior, program behavior (or parts thereof), whereas in the later we attempt at modeling a community of users, the behaviors of users, and the decisions each user would make.

4.1 Simulation Model

As already stated, the goal of our simulation is to model a community of users in a P2P system based on the behavior of individual users. While designing this kind of simulator we have been influenced by similar work in the area of social sciences. Social scientists have been using simulations to study the various social phenomena like evolution and expiry of societies, cooperation models and their effects on society, dynamics of decision making process in organizations, and so forth [3, 10]. We have adopted these ideas to the specific situation we are dealing with. While our simulation model draws ideas from a host of these work, our model has its own unique features.

Each peer in our system is characterized by the following parameters. A peer identifier(PID), a list of files that it owns and shares at any time instant, and a list of files it has already downloaded from the system. We represent the list of files a peer P_i owns at time T by $UList(P_i, T)$ and the list of files downloaded by peer P_i till time T by $DList(P_i, T)$. The set of all files available in the system at time T is represented by $GlList(T)$. It is just the union of $UList$ of all peers present in the system at time T.

$$GlList(T) = \bigcup_{i=1}^{i=N} UList(P_i, T) \tag{6}$$

Each file resource in our model has four parameters. (1) A file identifier(Fid) indicating the system wide unique identifier allotted to the file, (2) an Owner identifier ($Owid$) indicating which peer owns the file, (3) $Size$ parameter indicating the size of the file in bytes, and (4) a popularity metric ($Poplrty$) which indicates the popularity of the file. $Poplrty$ is a random integer between 0 and 100 and is assigned to a file at the time of its creation. We would like to explicitly state that this parameter has no relationship with the same term used in section 3.2. The utility function calculation doesn't depend upon this parameter. This parameter is solely used to generate user file access patterns. As explained later in this subsection, whenever a peer wants to download a file, it selects the file based on this $Poplrty$ parameter.

Modeling user behavior without free riding control
We now explain how we model the behavior of an individual peer in our P2P system. We would like to model the user as close to reality as possible. As in actual P2P file sharing systems, a peer in our model repeatedly attempts to download files from the system. In doing so it executes the following cycle.

Peer P_i prepares a list of all files available in the system that are not owned by itself or that have not been already downloaded by it. We call this list as $ToBeDList$. $ToBeDList(P_i, T)$ denotes all the files in the system that are not owned by P_i and have not been downloaded till time T. This list is obtained through the following equation.

$$ToBeDList(P_i, T) = GlList(T) - (UList(P_i, T) \cup DList(P_i, T)) \tag{7}$$

The peer then sums up the $Poplrty$ of all files in the $ToBeDList$ to obtain $TotalPoplrty(P_i, T)$.

$$TotalPoplrty(P_i, T) = \sum_{F_j \in ToBeDList(P_i,T)} Poplrty(F_j) \tag{8}$$

The peer P_i compares $TotalPoplrty(P_i, T)$ with a preset $PoplThreshold$. If $TotalPoplrty(P_i, T) <$

$PoplThreshold$, then it means that there are not enough file choices for the peer to download. In this case peer P_i increments a counter which we call as $WaitCntr$ and chooses to wait for random time. This models users who on finding out that there are not enough files in the system choose to wait and try again later. If $WaitCntr$ exceeds a preset threshold called the $CntrThreshold$, then the peer P_i exits the system. This models peers who on repeated attempts find no new choices of files in the system, hence getting frustrated exit the system. When peer P_i exits the system all the files owned by it will be removed from the $GlList$ of the system.

If $TotalPoplrty(P_i, T) \geq PoplThreshold$, then there are enough files for the peer to download. Then the peer randomly selects a file from the $ToBeDList$ based on the $Poplrty$ values of individual files in the list. The file selection process itself works as follows. The peer places $Poplrty$ values of all files in the $ToBeDList$ in an array called the $PoplrtyArray$. Then the peer generates a random number (called the $RandPoplrty$) between 0 and $TotalPoplrty$. The peer then searches through the $PoplrtyArray$ to find an index l such that $\sum_{j=0}^{j=l-1} PoplrtyArray[j] < RandPoplrty$ and $\sum_{j=0}^{j=l} PoplrtyArray[j] \geq RandPoplrty$. The file corresponding to the index l in the $PoplrtyArray$ is chosen for downloading. It can be seen that in the above described file selection process, a file with large $Poplrty$ value is likely to be chosen with higher probability than a file with lower $Poplrty$ value.

Once a file is selected for downloading, the model simulates downloading of the file by the peer. To model this effect, we let peer P_i sleeps for $\frac{FileSize}{Bandwidth}$ time period, where $FileSize$ indicates the size of the chosen file in bytes and $Bandwidth$ indicates the average download speeds in bytes per second. At the end of downloading the peer again sleeps for a random time indicating the rest period. We introduced this sleep because in P2P file sharing systems, the peers would not be continuously downloading files. Rather they download, wait for some time and then again look for other files to download. At the end of this random sleeping, the whole cycle is repeated until the peer exits the system as discussed earlier.

The P2P system itself expires when the number of files available in the system becomes very low or when there are no more peers left in the system. We measure the $Lifetime$ of the system as the difference between the system start time and the system expiration time. This $Lifetime$ indicates the sustainability of the system.

Modeling user behavior under the utility scheme
The above model represents a P2P file sharing system without any policies for controlling free riding. This model needs some modification to adopt it to represent the scenario when the system has policies that discourage free riding. We now explain how this model has been adopted to represent a system that has utility function based scheme to control free riding.

The user behavior in the new scenario is essentially the same as the previous scenario, except for the fact that whenever a peer attempts to download a file, its utility value will be verified against the size of the file it is trying to download. If the size of the file is lesser than the utility value, the downloading is permitted and the utility value is reduced by an amount equal to the size of the file.

If the size of the file is greater than the utility points available, file downloading is denied. Now the peer has two choices. Either it can share an extra file, or it can choose to wait and return later if its utility value has increased sufficiently to permit the downloading (remember that the utility value can increase either because some other peer downloaded a file from this peer or because some Freebie utility points were allotted to it at the end of an epoch). In our simulations whenever a peer doesn't have enough utility points to download a file, with a some small predetermined probability (represented as $AddProb$) it chooses to add a file and get additional utility points equal to $\gamma \times size(NewFile)$. With $(1 - AddProb)$ it chooses to wait for a random time and come back again and check whether it has got enough points to download the file it wants. This cycle repeats till the peer accumulates enough points for downloading the desired file.

Whenever a peer P_i downloads a file from another peer P_j, the utility value of peer P_j is increased as indicated in the previous subsection. In order to allocate freebies, we have chosen to have epochs in our model. An epoch is a prespecified time duration. At the end of each epoch, freebies are added to each peer.

4.2 Model Implementation

We have implemented the above described model in Java. Each peer in the system is implemented as a thread. The thread repeatedly executes the operations described in the previous subsection. In addition to these threads there is a main thread, that wakes up at the end of each epoch to allocate freebies. This thread also tabulates the statistics at the end of each epoch. All the peer threads are created at the beginning of the simulation. The simulator, at this point of time doesn't support peers joining the group after the simulation has started. We plan to include this feature in future.

Each thread representing a peer maintains a list of all files it owns($UList$). It also maintains a list of files that it has already downloaded($DList$). The global data structures include a list of all the files available in the system and another list of all the peers present in the system. Each time a peer wants to download a file, it prepares a $ToBeDList$ of files. This list just contains the files that are present in the Global file list but not present in either the owned files list or the downloaded files list of the peer. The peer randomly chooses a file from the list based on the $Poplrty$ value of all the files in the $ToBeDList$ as described in the previous subsection.

Whenever a peer shares an extra file, it is added both to its $UList$ and to the global file list ($GlList$). Whenever a peer quits a system, it removes all the files that it owns (i.e., those in its *UList*) from the global file list of the system ($GlList$). Once a peer quits the system, the thread corresponding to the same would die. When the number of peers in the system falls to zero or the number of files in the system becomes zero, the system expires.

In the next section we explain the actual experiments we performed and the results obtained.

4.3 Experimental Results

In order to make the above model tractable and the results comprehensible, we had to make some some assumptions regarding the parameters of the model. Before we discuss the actual experiments and results, we state the assumptions we have made.

First, we assume that all the files in the system are 5.6 MB in size ($Size = 5.6M$). Second, we assume that all peers in the P2P system have the same bandwidth of 56KB/sec ($Bandwidth = 56KB/sec$) Downloading each file takes approximately 100 secs. We also set the rest period after a peer downloads a particular file and the wait period to be a random number around the file downloading time duration (100sec). Further we set the duration of an epoch to be ten times that of downloading a file, which makes epoch-period to be of 1000 secs. This means that a peer having enough utility points can download up to 5 files in each epoch.

Second we set the γ and k parameters in equation 5 to be 1. Also the freebie points that are allocated to each peer is set to be equal to that of the file size (5.6M). This means that even if a peer doesn't share any files it can download a single file in each epoch, where as a peer having enough points can download as many as 5 files.

Understanding the Effects of Free Riding
In the first experiment we just wanted to test the hypothesis that the lifetime of a system where everyone shares equal number of files is higher than the lifetime of a system where there are just a few peers who share the files and the rest are free riders. To test this hypothesis, we set up a system with no free riding control of any sort and subjected the system to two different scenarios. In both scenarios the number of files was made equal to 15 times the number of nodes.

The first scenario represents a typical P2P file sharing system with large number of free riders. 2% of the total nodes share 250 files each, 4% share 100 files each, 8% share 50 files each and 8% share 25 files each. The rest 78% are freeriders. In the second scenario each node shares

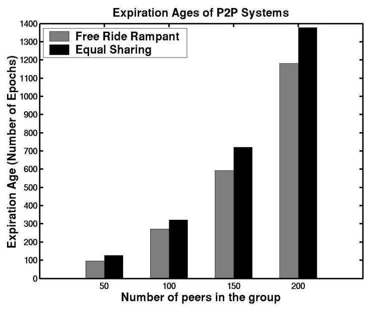

Figure 1. **Lifetime of Systems With and Without Free Riding**

the same number of files. Each peer shares exactly 15 files. The graphs in Figure 1 show the results of the two scenarios when the number of peers was 50, 100, 150 and 200. The Y axis in the graph measures the lifetime of the systems in number of epochs. It is seen from the graphs that the system where each node shares the same amount of files lives for around 15% to 25% longer than the system where freeriding is rampant.

This phenomenon is explained as follows. In first scenario where free riding is rampant, the $ToBeDList$s for the peers that share large number of files is much shorter than those of the free riders. If there are 1500 files in the system and if one peer owns 250 files, the number of files that it can download is limited to 1250, whereas a free rider sharing practically nothing has 1500 files to be downloaded. Therefore the $TotalPoplrty$ of the $ToBeDList$ of a peer sharing large number of files reaches the $PoplThreshold$ (recall that this threshold denotes the condition of a peer that experiences lack of popular files in the system and hence pulls out) much earlier than that of a free rider. Hence it is likely to loose interest much earlier than a free rider. This causes it to exit the system. Once a node with large file exits the system, the system becomes poorer in terms of the total number of available files. This has the cascading effect of other peers having their own $ToBeDList$ list becoming shorter by a large amount, accelerating their exits from the system.

In a system wherein each node shares same number of files, every peer has same number of files to be downloaded from the system. Hence we don't observe the phenomenon of large sharers exiting the system very early, which would have accelerated its expiry.

To explain this phenomenon further, we have plotted the

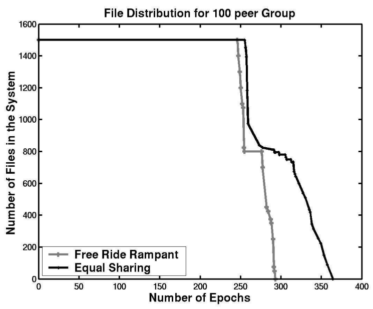

Figure 2. **Number of Files in the System as a Function of Epoch Number**

graph of the number of files available as a function of the epoch number for both the equal sharing system and the free riding rampant system. Figure 2 shows the graph of the number of files in the system against the epoch number for a group of 100 peers. As is evident from the graph, there are sudden large drops in the number of files available for the free ride rampant system. These indicate the points where peers who share large files are withdrawing from the system. When these peers withdraw, suddenly the system becomes poor due to the loss of a large number of files. In contrast, in the curve corresponding to the equal sharing system, the drops are shorter and they also occur later. This phenomenon can be explained as follows. In a system with equal sharing peers, each peer has 1485 files to be downloaded. However a peer sharing 250 files in free ride rampant system, has only 1250 files. This accelerates its exit from the system causing the system to become poor by losing 250 files at once.

As we have already mentioned, another problem with free riding is that a few peers that share files receive all the downloading request. To demonstrate this effect, we measure a quantity which we term as *Average-System-Skewness*($AvgSysSkew$). We define Average-Skewness in terms of another related quantity called *Epoch-Skewness* ($EpSkew$). Average-Epoch-Skewness of epoch i is defined as follows.

$$EpSkew(i) = \frac{\sum_{\{\text{peers alive at epoch i}\}} (\| (UC(p_j,i) - DC(p_j,i)) \|)}{\text{Number of peers alive at epoch i}} \tag{9}$$

In equation 9, $UC(p_j, i)$ denotes the number of times any file owned by peer p_j has been downloaded by other peers during the epoch i. $DC(p_j, i)$ denotes the number of files peer p_j has downloaded from other peers in the epoch i. This quantity indicates the average difference between the number of uploads and number of downloads in each epoch. System-Average-Skewness is defined as the average of the Epoch-Skewness over all epochs in which the system was alive.

$$AvgSysSkew = \frac{\sum_{\{i|\text{System alive at i}\}} (EpSkew(i))}{\text{Total Number of Epochs}} \tag{10}$$

In a healthy system, the value of Average-System-Skewness should be close to zero. High values indicate that there are many peers that are downloading a lot more than other peers are downloading from it or vice versa.

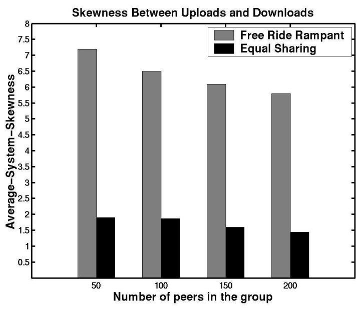

Figure 3. **Average-System-Skewness of System With and Without Free Riding**

Figure 3 indicates this quantity for peer groups of 50, 100, 150 and 200 peers for system with equal sharing and the system with rampant free riding. The value of Average-System-Skewness of free riding rampant system of 50 nodes is 7.2 as against a value of 1.9 for equal sharing system. The result is expected because in the system with rampant free riding, there are nodes that only download but don't share anything for others to download, whereas in an equal sharing system each peer can be expected to download as many files as other peers download from it.

These set of experiments clearly demonstrates that free riding not only limits the lifetime of a system, but also is an unhealthy trend.

System performance under the utility scheme

Having demonstrated the need to tackle the free riding problem in the previous set of experiments, we now turn our attention to the performance of the system with our utility function.

In these set of experiments we considered a peer group of 50, 100 and 150 peers. However, we considered a more realistic distribution of files on peers. The workload we considered consisted of 20% of peers sharing 50 files each, another 20% of peers sharing 20 files each and another 20% share 10 files each. The rest 40% are free riders.

We performed three sets of experiments. The first set was a system with no control on free riding. The second and third set of experiments implemented our utility based free riding control mechanism. In the second set, the $AddProb$ (the probability of adding a file when a peer has very few points) is set to 0.05. In the third set of experiments it was set to 0.1.

Table 1 indicates the lifetime of each of the three systems

Number of Peers	50	100	150
No Utility Function	119	512	736
Utility Function 3 (A P = 0.05)	224	1200	2653
Utility Function 3 (A P = 0.10)	393	5309	7531

Table 1. Lifetime of Systems With and Without Free Riding Control

in number of epochs. It is seen that for a 50 peer group, when the addition probability $AddProb$ (AP for short) was set to 0.1, the lifetime of the system with utility based free riding control was approximately 3.5 times the lifetime of the system with system with no free riding control. Whereas the lifetime of a 150-peer system with utility based free riding control system was 10 times that of the the system with no free riding control.

It should be noted that the percentage improvement in lifetime is much higher for the systems with a larger number of peers when compared the systems with fewer peers. This phenomenon is explained as follows. In systems with large number of peers, there are large number of free riders as well. The utility based free riding control forces these free riders either to share a new file or wait. Even if the peers decide to share new files with small probability, the number of files entering the system would be considerable. This helps to retain the interests of the peers in the system for much longer time. Thus the lifetime of the system itself becomes high.

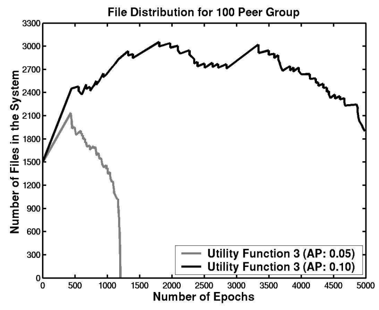

Figure 4. Number of Files at Various Epochs

Figure 4 indicates the number of files in the system as a function of the epoch number for a system with 100 peers. The two curves in the graph correspond to the cases when the *AddProb* was set to 0.05 and 0.1 respectively. In order to capture details and for better clarity, we have plotted the graph for the case of *AddProb*=0.1 only till the number of peers in the system falls to half of the original number of peers. The zig-zag curve in this case clearly demonstrates that the number of files in the system increases even after a few nodes have pulled out of the system. In fact the peak number of files are available in the epoch number 1802, by this epoch 8 peers had already pulled out of the system. After the epoch number 5000, the fall is steep and the whole system expires within 200 more epochs. The same zig-zag effect is observed in the case when the $AddProb$ is 0.05. However, the effect is on a much smaller scale.

We would like to note that our experiments at this stage don't account for new peers entering the system. If we incorporate this feature, then we can expect the system to become stabilized and live for much longer time.

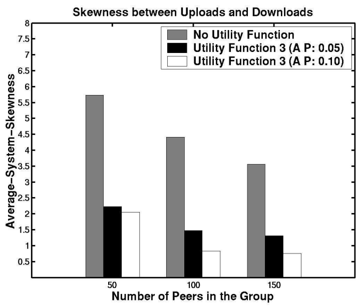

Figure 5. Average-System-Skewness for Utility Based Scheme

Figure 5 shows the Average-System-Skewness for systems with 50, 100 and 150 nodes in the three cases, a system with no utility function, a system with utility function at *AddProb* = 0.05, and a system with utility function at *AddProb* = 0.1. In a system with 50 peers the Average-System-Skewness of the system with no free riding control was 2.5 times that of Average-System-Skewness of the system with free riding control with *AddProb* set to 0.1. Average-System-Skewness was as low as 0.76 for a system with 150 nodes and *AddProb* set to 0.1. This indicates that the difference between the number of uploads and downloads is extremely low, which is a very healthy trend. This demonstrates the effectiveness of the utility scheme in protecting peers being exploited and preventing some peers exploiting others. Due to space limitations we have not been able to provide complete set of our experimental results. Interested reader is referred to the technical report version of the paper [11] for a more elaborate set of results.

5 Discussion

The experiments we discussed in the previous section demonstrates a number of important points. The first set of results illustrated the seriousness of the free riding problem and the need to tackle this growing menace. The second set of results demonstrated the effectiveness of the utility function based schemes in controlling free riding. Even simple utility functions like the one we used in our experiments not only increase the lifetime of the system by leaps and bounds, but also guard against some peers exploiting other peers.

The particular utility function that we have used is a general one. Obviously, specialized utility functions should be designed for specific systems. Our research is a first attempt to show that utility function based schemes are a promising option for controlling free riding.

It should also be noted that our simulation at this point of time, doesn't incorporate new users entering the system. It also doesn't model altruism in actual P2P systems. Altruism is the behavior of a peer wherein it contributes new files to the system without any apparent benefit for itself. The future work would be to incorporate these two features in our simulation model and also design variants of the basic utility functions for various special applications. We also recognise the need for further research on economic models to better represent the user communities and resources in a P2P information sharing environment.

6 Related Work

Research in P2P file sharing systems till now, has predominantly concentrated on file look up mechanisms. While Napster [9] used a centralized directory, Gnutella floods the network with query messages. Chord [14] and Pastry [12] propose novel solutions to this problem by hashing file names to the nodes present in the system. [15] provides a comparative study of different Peer-to-Peer architectures. [13] reports a detailed measurement study comparing Napster and Gnutella with respect to various performance parameters.

The problem of free riding in P2P file sharing systems was first reported in [1]. However, the paper doesn't propose any solution to the problem. Economists and social scientists have studied the problems of free riding, social dilemma, and the effect of them in various communities [4] [10]. Recently, there have been studies on these problems in the context of information and internet technology [8] [6]. But most of these studies stop at recognizing the problem and don't propose any solutions. Our work is unique in the sense that we propose schemes that can effectively counter the free riding problem.

7 Conclusion

While there are several on going research projects on improving the quality of service in P2P file sharing systems, there hasn't been any research to counter the problem of free riding, which is essentially a data quality issue. To address the free riding problem in P2P systems, we have introduced the concept of utility function to measure the usefulness of every user to the system. We have proposed a free riding control scheme based on the general utility function. Further we designed a simulation model to study the user behavior patterns in a peer community and the impact of the policy decisions on the lifetime and health of the system. We expect that this paper would trigger further research in this area of P2P systems.

References

[1] E. Adar and B. Huberman. Free riding on gnutella, September 2000. Available at $http : //www.firstmonday.dk/issues/issue5_10/adar/index.html$.

[2] Freenet home page. http://www.freenet.sourceforge.com.

[3] N. Gilbert. Emergence in social simulations. In *Artificial Societies*, pages 144 – 156. 1995.

[4] N. S. Glance and B. A. Huberman. Dynamics of social dilemmas. *Scientific American*, March 1994.

[5] Gnutella development page. http://gnutella.wego.com.

[6] B. A. Huberman and R. Luckose. Social dilemmas and internet congestion. Science, Volume 277, 1997.

[7] Kazaa home page. http://www.kazaa.com.

[8] D. McFadden. The tragedy of the commons. Available at http://www.forbes.com/asap/2001/0910/061.html.

[9] Napster home page. http://www.napster.com.

[10] D. Parisi. What to do with a surplus. In *Simulating Social Phenomenon*, pages 133 – 151. 1997.

[11] L. Ramaswamy and L. Liu. Free riding: A new challenge to peer-to-peer file sharing systems. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, 2002.

[12] A. Rowstron and P. Dreschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Platforms (Middleware 2001)*, November 2001.

[13] P. K. G. Stefan Saroiu and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical report, Department of Computer Science and Engineering, University of Washington Seattle, 2002.

[14] I. Stocia, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishanan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[15] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proceedings of 21st International Conference on Very Large Databases, VLDB-2001*, September 2001.