An Efficient Algorithm for Resource Sharing in Peer-to-Peer Networks

Wei-Cherng Liao, Fragkiskos Papadopoulos, and Konstantinos Psounis

University of Southern California, Los Angeles, CA 90089
{weicherl, fpapadop, kpsounis}@usc.edu

Abstract. The performance of peer-to-peer systems depends on the level of cooperation of the system's participants. While most existing peer-to-peer architectures have assumed that users are generally cooperative, there is great evidence from widely deployed systems suggesting the opposite. To date, many schemes have been proposed to alleviate this problem. However, the majority of these schemes are either too complex to use in practice, or do not provide strong enough incentives for cooperation.

In this work we propose a scheme based on the general idea that offering uploads brings revenue to a node, and performing downloads has a cost. We also introduce a theoretical model that predicts the performance of the system and computes the values of the scheme's parameters that achieve a desired performance. Our scheme is quite simple and very easy to implement. At the same time, it provides very strong incentives for cooperation and improves the performance of P2P networks significantly. In particular, theory and realistic simulations show that it reduces the query response times and file download delays by one order of magnitude, and doubles the system's throughput.

Keywords: P2P networks, user cooperation, theoretical analysis, realistic simulations.

1 Introduction

Peer-to-peer (P2P) systems provide a powerful infrastructure for large-scale distributed applications, such as file sharing. While cooperation among the system's participants is a key element to achieve good performance, there has been growing evidence from widely deployed systems that peers are usually not cooperative. For example, a well known study of the Gnutella file sharing system in 2000 reveals that almost 70% of all peers only consume resources (download files), without providing any files to the system at all [1]. This phenomenon is called "free-riding".

Despite the fact that this phenomenon was identified several years ago, recent studies of P2P systems show that the percentage of free-riders has significantly increased [2]. This is not because industry and academia have ignored the problem. There is a large body of work on incentive mechanisms for P2P networks, varying from centralized and decentralized credit-based mechanisms, *e.g.*

[3, 4, 5, 6], to game-theoretic approaches and utility-based schemes, *e.g.* [7, 8], to schemes that attempt to identify and/or penalize free-riders, *e.g.* [9, 10, 11], the last two being proposed by the popular KaZaA and eMule systems. The problem of free-riders is hard to tackle because the solution has to satisfy conflicting requirements: minimal overhead, ease of use, and at the same time good amount of fairness and resilience to hacking.

In this paper we propose and study the performance of an efficient algorithm that is very easy to use, it enforces users to be fair, and it can be implemented in a number of ways that tradeoff overhead and resilience to malicious users. According to the algorithm, users use tokens as a means to trade bytes within the system. A user earns K_{up} tokens for each byte he/she uploads to the system and spends K_{down} tokens for each byte he/she downloads from the system. The user also gains K_{on} tokens for each second his/her machine is on the system (*i.e.* it is online). A user can initiate a download only if the number of tokens that he/she has is large enough to download the complete file.

The proposed algorithm relies on the general idea that users should be awarded for offering uploads and staying online, and pay for performing downloads. While others have proposed solutions that use the same general idea in the past, *e.g.* [6, 8], there are a number of questions that either have not been addressed at all or have been studied via simulations only: (i) How should one tune the parameters that dictate the gain from uploads and the cost of downloads? Specifically to our scheme, what is the right value for the parameters K_{on}, K_{up}, K_{down}? (ii) What is the exact effect of such an algorithm on overall system performance over a wide range of conditions? (iii) Would a *small* number of malicious users, that manage to subvert the scheme, degrade overall performance noticeably? (iv) Is it possible to trade off one performance metric for another by varying the parameters of the algorithm, e.g. trade off download delay for total system capacity? Our theoretical analysis of the performance of the resulting system, coupled with extensive realistic simulations, gives concrete answers to all these questions. Interestingly enough, it shows that the query response times and file download delays can be reduced by one order of magnitude while being able to sustain higher user download demands.

An important aspect of any solution to the free-riding problem is if the information about the users' behavior is determined and maintained locally and without any interaction with the other peers of the system (localized solutions), or it is determined and maintained by either a centralized authority (non-localized centralized solutions), or by the continuous exchange of information among the system's participants (non-localized distributed solutions). Localized solutions are simple and impose very little overhead but they are easy to subvert. Non-localized solutions are hard to subvert but are complex to use in practice. (KaZaA, eMule, and BitTorrent all use localized approaches.) In any case, our proposed scheme can be easily implemented in any way, and we describe later in the paper how to implement it efficiently with each one of the approaches.

The rest of the paper is organized as follows: In Section 2 we briefly discuss prior work on providing incentives for P2P systems. In Section 3 we provide a detailed description of the proposed scheme. In Section 4 we provide a mathematical model that predicts the performance of the system and gives guidelines on how to set the scheme's parameters. In Section 5 we present realistic experiments of P2P systems on top of TCP networks. In Section 6 we briefly discuss some implementation thoughts and finally conclude in Section 7.

2 Related Work

There has been a large body of work on incentive mechanisms for P2P networks. Three of the most popular localized schemes are the ones implemented by the eMule [11], the KaZaA [12], and the BitTorrent [13] systems. eMule rewards users that provide files to the system by reducing their waiting time when they upload files using a scoring function (called *QueueRank*). Similarly, in KaZaA, each peer computes locally its *Participation Level* as a function of download and upload volumes, and peers with high participation levels have higher priority [10]. A disadvantage of both of these schemes is that they provide relatively weak incentives for cooperation since peers that have not contributed to the system at all can still benefit from it, if they are patient enough to wait in the upload queues. Other problems include that they favor users with high access bandwidth, which may result in frustration or a feeling of unfairness [14], and that they are vulnerable to the creation and distribution of hacked daemons that do not conform to the desired behavior [15]. BitTorrent uses a different scheme that is specific to its architecture. Each peer periodically stops offering uploads to its neighbors that haven't been offering uploads to him/her recently. This scheme is hard to subvert. However, it suffers from some unfairness issues and it only works with "BitTorrent-style" systems, that is, in systems where files are broken into pieces, and downloading a file involves being connected to almost all of ones neighbors in order to collect and reassemble all the pieces of the file.

Non-localized proposals are primarily concerned with creating systems that cannot be subverted. Some of them make use of credit/cash-based systems. They achieve protection from hackers by either using central trusted servers to issue payments (centralized approach), *e.g.* [3, 4], or by distributing the responsibility of transactions to a large number of peers (distributed approach), *e.g.* [6]. Other distributed approaches use lighter-weight exchanged-based mechanisms, *e.g.* [16], or reputation management schemes, *e.g.* [5]. These mechanisms are indeed hard to subvert but they are also quite complex to use in practice [16].

In this paper we decouple the issue of how to design an algorithm to prevent free-riding from the issue of how to implement this algorithm in a P2P system. We first propose an efficient scheme that provides very strong incentives for cooperation. We show this via both theory and simulations. Then, we show that the scheme is generally applicable to any P2P system and comment on how to implement it using either a localized, or a non-localized approach. Another important contribution is the theoretical analysis of the performance of a P2P system with and without the proposed scheme. The analysis yields a set of equations that are used to predict the system's performance under a wide range of conditions, and to tune the parameters of the scheme.

3 A Simple and Effective Algorithm

As mentioned earlier, the algorithm uses tokens as a means to trade bytes within the system. Each user is given an initial number of tokens M when he/she first joins the network. This allows new users to start downloading a small number of files as soon as they join the system. When a user rejoins the system he/she uses the amount of tokens he/she previously had.

Users spend K_{down} tokens for each byte they download from the system and earn K_{up} tokens for each byte they upload to the system. This forces users to offer files for upload proportionally to the number of files they want to download. Further, users gain K_{on} tokens/sec while being online. This mechanism of accumulating tokens serves two purposes. First, it allows users who are not contacted frequently for uploads to gain tokens by just being online, which is more fair towards users with low access bandwidth [14]. Second, it provides an incentive for users to keep their machines on the system even when they are not downloading a file, which helps to prevent the so-called problem of "low availability" [17]. Note that the value of K_{on} should be relatively small, in order to prevent users from gaining many tokens by just keeping their machines on without providing any uploads. Finally, a user can initiate a download only if the number of tokens he/she currently possesses is greater or equal to the number of tokens required to download the requested file.

This scheme provides strong incentives for cooperation. Free-riders are "forced" to provide some uploads to the system in order to gain tokens fast enough to sustain their desirable download demands. Some free-riders may decide to share their files as soon as they are out of tokens. Others may adopt a more dynamic behavior and decide to adjust the number of uploads they provide to the system as a function of the number of tokens they currently have. In any case, the change in the free-rider's behavior increases the amount of available system resources tremendously, which, in turn, significantly improves the system's performance, as we shall see in Section 5.

4 A Mathematical Model for the Proposed Scheme

In this section we derive a mathematical model that predicts the system's performance and can be used to tune the parameters of the scheme. Predicting the performance of the system from the model is beneficial because the alternative is P2P simulations/experiments, and those either involve a significantly smaller number of peers than in reality, or are prohibitively expensive. Tuning the parameters of the scheme is important because an arbitrary setting of their parameters may lead to several undesired situations. For example, giving a large value to K_{on} may provide tokens to the free-riders fast enough, so that there won't be any reason for them to start sharing their files with the system. As another example, giving relatively small values to both K_{on} and K_{up} may reduce the token accumulation rate of cooperative users so much such that they can't sustain their download demands.

4.1 System Dynamics

We assume a system that implements the proposed scheme which we call "system with the tokens". Recall that K_{down} and K_{up} are expressed in tokens/byte and K_{on} in tokens/sec. Now, let C_{down} and C_{up} denote the file download and upload speeds of a user (access line bandwidth), both expressed in bytes/sec. The user spends $K_{down}C_{down}dt$ tokens if he/she is downloading files from other peers during time $(t, t+dt)$. Also, he/she earns $K_{on}dt$ tokens if he/she is online during time $(t, t+dt)$ and $K_{up}C_{up}dt$ tokens if other users are uploading files from the user under study during time $(t, t+dt)$. Let $T(t)$ denote the number of the user's tokens at time t, with $T(0) \geq 0$. We can then write the following differential equation:

$$\frac{dT(t)}{dt} = K_{on}I_{on}(t) + K_{up}C_{up}I_{up}(t) - K_{down}C_{down}I_{down}(t), \tag{1}$$

where

$$I_{on}(t) = \begin{cases} 1 & \text{if the user is online in (t, t+dt)} \\ 0 & \text{otherwise} \end{cases},$$

$$I_{up}(t) = \begin{cases} 1 & \text{if the user provides uploads in (t, t+dt)} \\ 0 & \text{otherwise} \end{cases},$$

$$I_{down}(t) = \begin{cases} 1 & \text{if the user performs downloads in (t, t+dt)} \\ 0 & \text{otherwise} \end{cases}.$$

Taking expectations on both sides of Equation (1), and interchanging the expectation with the derivative on the left hand side[1], we get:

$$\frac{dE[T(t)]}{dt} = K_{on}P_{on}(t) + K_{up}C_{up}P_{up}(t) - K_{down}C_{down}P_{down}(t), \tag{2}$$

where $P_{on}(t)$ is the probability the user is online at time t, $P_{up}(t)$ is the probability that the user provides uploads to the system at time t, and $P_{down}(t)$ is the probability that the user performs downloads from the system at time t. Note that Equation (2) can be regarded as a fluid model describing the token dynamics.

$P_{on}(t)$, $P_{up}(t)$, and $P_{down}(t)$ depend on how the user behaves given the number of tokens that he/she has at some point in time, and on his/her download

[1] Taking into account that $T(t)$ is bounded in practice, we can use the bounded convergence theorem [18] to justify the interchange.

demands. Along these lines, one can define user profiles and solve the differential equation. Due to limitations of space we will not proceed with this task here. (The interested reader is referred to [19]). Instead, we will only study the steady state by setting $\frac{dE[T(t)]}{dt} = 0$, and dropping the time dependence from the probabilities in Equation (2). Note that the existence of a steady state can be easily justified for a free-rider, by taking into consideration that in the long-run he/she will spend as many tokens as he/she gains. [2]

Without loss of generality assume $P_{on} = 1$. [3] Let R_{up} be the long-run average rate of file upload requests per second that the user handles, which we refer to as the *upload rate.* Also, let R_{down} be the long-run average rate of file download requests per second that the user initiates, which we refer to as the *download rate.* Last, let S denote the average file size in the system in bytes. Then, it is easy to see that $P_{up} = \frac{R_{up}S}{C_{up}}$ and $P_{down} = \frac{R_{down}S}{C_{down}}$. [4] Equation (2) in steady state yields:

$$K_{on} + K_{up}R_{up}S - K_{down}R_{down}S = 0.$$

Taking the average over all free-riders yields:

$$K_{up} = K_{down}\left(\frac{E[R_{down}|\text{FR}]}{E[R_{up}|\text{FR}]}\right) - \frac{K_{on}}{E[R_{up}|\text{FR}]S}. \tag{3}$$

Equation (3) relates the parameters of the scheme, K_{on}, K_{up}, and K_{down}, with the average download and upload activity of free-riders. We will later use it to select the parameter values that yield a target performance. But first, we need to compute the average download and upload rates, which is the next topic.

4.2 User Download Rate (R_{down})

Let N be the number of peers in the system and let a proportion α of them be free-riders. Assume that free-riders are uniformly distributed over the system. Also, assume that both cooperative users and free-riders have the same download demands. In particular, they have the same query request rate, denoted by R_q queries/sec, and the same preference over files, that is, each query is for file i with some probability $Q_f(i)$ irrespectively of the query issuer. Finally, assume that free-riders respond to a query only if the amount of tokens they currently have is less than the amount required to download the file they currently desire. (Recall that cooperative users always respond to query requests.)

Let $P_{ans}(i)$ be the probability that a query request for file i is successfully answered. Now, recall that in the system with the tokens a user can initiate a download only if the amount of tokens he/she has is larger than the amount

[2] Considering the existence of a steady state for a non-freerider is a bit more involved. As we will shortly see he/she may or may not have a steady state. Nevertheless, this will not be important for the system dynamics.

[3] A similar analysis holds for $P_{on} < 1$.

[4] Assuming a stable system, the exact values of C_{up} and C_{down} are not important for our analysis.

required to download the file. Let P_{tkn}^{FR} and P_{tkn}^{NF} denote respectively the probability that a free-rider and a non-freerider have enough tokens to initiate a download. Then, we can express the average download rate of free-riders and non-freeriders as follows:

$$E[R_{down}|\text{FR}] = \sum_i R_q \cdot Q_f(i) \cdot P_{ans}(i) \cdot P_{tkn}^{FR}, \tag{4}$$

$$E[R_{down}|\text{NF}] = \sum_i R_q \cdot Q_f(i) \cdot P_{ans}(i) \cdot P_{tkn}^{NF}, \tag{5}$$

where the summation is taken over all files i. Clearly, the average download rate over all users in the system is:

$$E[R_{down}] = E[R_{down}|\text{FR}] \cdot \alpha + E[R_{down}|\text{NF}] \cdot (1 - \alpha). \tag{6}$$

To complete the calculation of the download rates, note that R_q, $Q_f(i)$, and α are given quantities. (There exist a large body of work in measurement studies of P2P systems, e.g. [20, 21], from which one can deduce typical values for these quantities.) Hence, what remains is to compute P_{ans}, P_{tkn}^{FR}, and P_{tkn}^{NF}. We start by deriving a relation between P_{tkn}^{FR} and P_{tkn}^{NF}. First, note that in steady state the token earning rate equals the token spending rate for each free-rider. A free-rider responds to a query request only when he/she doesn't have enough tokens, i.e. with probability $1 - P_{tkn}^{FR}$. Since a non-freerider always responds to a query request, it is easy to see that the token earning rate of free-riders over that of non-freeriders equals $1 - P_{tkn}^{FR}$. Now, the token spending rate is proportional to the download rate, and Equations (4) and (5) imply that the token spending rate of free-riders over that of non-freeriders equals $\frac{P_{tkn}^{FR}}{P_{tkn}^{NF}}$. Assuming that non-freeriders are also in steady state (in which case Equation (3) also holds if the average is taken with respect to non-freeriders only), we can equate the two ratios and write $P_{tkn}^{NF} = \frac{P_{tkn}^{FR}}{1-P_{tkn}^{FR}}$. Clearly, this equality is valid for $P_{tkn}^{FR} \leq 0.5$. In particular when $P_{tkn}^{FR} = 0.5$, $P_{tkn}^{NF} = 1$, which implies that non-freeriders always have enough tokens to initiate downloads. For $P_{tkn}^{FR} > 0.5$, the last equality no longer holds. In this case the token earning rate of non-freeriders will be larger than their token spending rate, which implies that their amount of tokens will continuously increase. However, this still suggests that $P_{tkn}^{NF} = 1$. We can now write:

$$P_{tkn}^{NF} = \min\left(1, \frac{P_{tkn}^{FR}}{1 - P_{tkn}^{FR}}\right). \tag{7}$$

Now, lets find a relation for $P_{ans}(i)$. First, assume that due to congestion at the overlay layer [22], each message (either a query request or a query response) has a probability p of being dropped at some peer. [5] Then, if L is the average

[5] This assumption is introduced to make the model more general. A well designed system usually has $p \approx 0$, which is accomplished by setting the buffer size of the TCP socket sufficiently large.

number of overlay hops until a query is answered, $P_{drop} = 1 - (1-p)^L$ is the probability that the query response is lost. Next, observe that if $K \leq N$ is the average number of peers that a query request can reach, then the request can be answered by an average of $K \cdot ((1 - P_{tkn}^{FR}) \cdot \alpha + 1 \cdot (1-\alpha))$ peers. Finally, let $P_f(i)$ be the probability that a peer has file i. We can then write:

$$P_{ans}(i) = 1 - (1 - P_f(i) \cdot (1 - P_{drop}))^{K \cdot ((1-P_{tkn}^{FR}) \cdot \alpha + 1 - \alpha)}. \qquad (8)$$

4.3 User Upload Rate (R_{up})

The total number of downloads equals the total number of uploads, and thus the expected download and upload rates over *all* nodes are also equal. This does not mean that all peers provide uploads. For example, in a system that does not implement the proposed scheme $E[R_{down}] = E[R_{up}]$ but we know that only non-freeriders provide uploads, i.e. $E[R_{up}|\text{FR}] = 0$, and hence $E[R_{up}|\text{NF}] = \frac{E[R_{down}]}{(1-\alpha)}$. On the other hand, in the system with the tokens each free-rider answers to a query request with probability $1 - P_{tkn}^{FR}$. As a result, this system behaves as if there are $N \cdot ((1-\alpha) + \alpha \cdot (1 - P_{tkn}^{FR}))$ non-freeriders. It is easy to see that the expected upload rate of each non-freerider is now given by:

$$E[R_{up}|\text{NF}] = \frac{E[R_{down}]}{(1-\alpha) + \alpha \cdot (1 - P_{tkn}^{FR})}. \qquad (9)$$

And, since $E[R_{up}] = E[R_{down}]$, the expected upload rate of each free-rider equals:

$$E[R_{up}|\text{FR}] = \frac{(1 - P_{tkn}^{FR}) \cdot E[R_{down}]}{(1-\alpha) + \alpha \cdot (1 - P_{tkn}^{FR})}. \qquad (10)$$

4.4 Choosing the Right Values for K_{on}, K_{up}, and K_{down}

We use P_{tkn}^{FR} as the design parameter of our system since it dictates how often free-riders offer uploads, which, in turn, specifies the average amount of available resources in the system. We are given the query- and file-popularity probability functions $Q_f(i)$, $P_f(i)$, the query request rate R_q, and information about the overlay network. (For example, information about the overlay network includes the percentile of free-riders α, the socket buffer sizes that dictate the drop probability p, and the structure of the overlay graph as well as the search algorithm that dictate the number of peers that a query reaches K and the average path length between a query issuer and a query responder L.) We want to find a set of values for K_{on}, K_{up} and K_{down} that will satisfy a target P_{tkn}^{FR}, and, in turn, a target performance.

First, observe from Equation (3) that it is the *relative* values of K_{on}, K_{up}, and K_{down} that are important for the proper operation of the system. Recall also that K_{on} should be sufficiently smaller than the token spending rate of free-riders. This is to prevent them from accumulating enough tokens by just staying online without offering any uploads. Thus, we should have $K_{on} \ll K_{down}E[R_{down}|\text{FR}]S$.

With the above observations in mind we proceed as follows in order to satisfy the target P_{tkn}^{FR}:

(i) Fix K_{down} to some arbitrary value,
(ii) use Equation (7) to compute P_{tkn}^{NF}, (To guarantee that cooperative users will not be penalized, P_{tkn}^{NF} should be close to 1.)
(iii) use Equations (4) and (8) to compute the value of $E[R_{down}|\text{FR}]$, and Equations (10), (6) and (5) to compute $E[R_{up}|\text{FR}]$,
(iv) assign a value to K_{on} which is one order of magnitude smaller than $K_{down}E[R_{down}|\text{FR}]S$, (The specific value turns out not to affect the performance sizeably.) and
(v) use Equation (3) to find the right value for K_{up}.

Conversely, if we are given the values of K_{on}, K_{up}, and K_{down} we can use our equations to predict quantities like $E[R_{down}|\text{FR}]$, $E[R_{down}|\text{NF}]$, $E[R_{up}|\text{FR}]$ and so on. [6]

In the next Section we verify the accuracy of our analysis via experiments on top of TCP networks, and show the impact of the proposed scheme on system's performance.

5 Experiments

5.1 Simulation Setup

For our experiments we use GnutellaSim [23], a packet-level peer-to-peer simulator build on top of ns-2 [24], which runs as a Gnutella system. We implement the file downloading operation using the HTTP utilities of ns-2.

We use a 100-node transit-stub network topology as the backbone, generated with GT-ITM [25]. We attach a leaf node to each stub node. Each leaf node represents a peer. The propagation delays assigned to the links of the topology are proportional to their length and are in the order of ms. We assign capacities to the network such that the congestion levels are moderate. The capacity assigned to a peer's access link is 1.5Mbps.

In order to test the algorithm on a general gnutella-like unstructured P2P network we use Gnutella v0.4, which uses pure flooding as the search algorithm and does not distinguish between peers. The TTL for a query request message is set to 7 (the default value used in Gnutella).

All peers join the system initially and never go offline. For simulation purposes we implement the following user behavior: each user initiates query requests at the constant rate of 1 query every 20sec. Once a timeout for a query request occurs, the corresponding query is retransmitted. The maximum number of retransmissions is set to 5, and the timeout to 60sec.

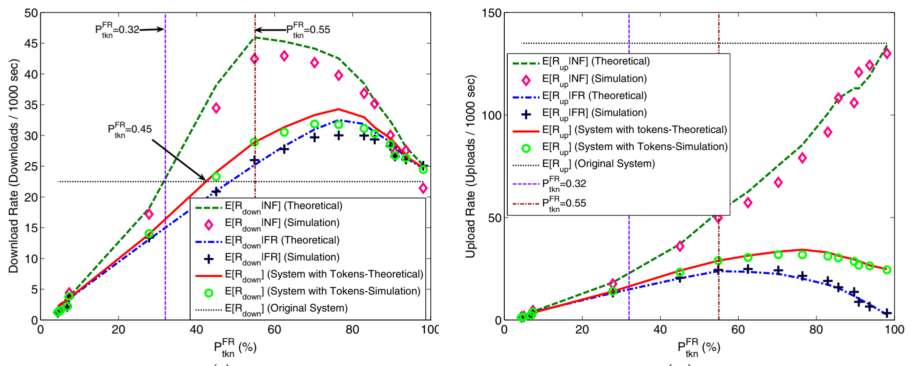
[6] Note that we can also use Equations (4)...(10) to compute upload/download rates in a system that does not implement the scheme, by setting $P_{tkn}^{FR} = 1$.

There are 1000 distinct files in the system, $i = 1...1000$. A query request is for file i with probability proportional to $\frac{1}{i}$ (Zipf distribution). The number of replicas of a certain file is also described by a Zipf distribution with a scaling parameter equal to 1, and the replicas of a certain file are uniformly distributed among all peers. These settings are in accordance with measurement studies from real P2P networks [20, 21]. We distinguish two systems: (i) the original system which does not implement the proposed algorithm, and (ii) the system with the tokens. In both systems, 85% of peers are free-riders in accordance to the percentage reported in [2]. Finally, the file size is set to 1MB.

5.2 Simulation Results

Download and Upload Rates. For various values of the design parameter P_{tkn}^{FR} we compute the corresponding values of K_{on}, K_{up} and K_{down} according to the procedure described in the previous Section. We then assign these values to all users of the system and compare the theoretical download and upload rates with the experimental results. Figures 1(i) and 1(ii) show respectively the expected download and upload rate over all non-freeriders, over all free-riders, and over all users of the system, as a function of P_{tkn}^{FR}. The horizontal line in Figure 1(i) represents the expected download rate of a user in the original system. (Clearly, in the original system $E[R_{down}] = E[R_{down}|FR] = E[R_{down}|NF]$.) The horizontal line in Figure 1(ii) represents the expected upload rate of a non-freerider in the original system. (Recall that in this system $E[R_{up}|FR] = 0$.)

It is clear from the plots that analytical and simulation results match. Further, we can make several interesting observations. First, notice that as P_{tkn}^{FR} increases, the download rate for both classes of users first increases and then starts decreasing until it reaches the value of the original system. Second, observe that while the upload rate of free-riders behaves in a similar manner, the upload rate of non-freeriders continuously increases until it reaches its original value. Based on these observations we divide the plots into three regions. The



(i) (ii)

Fig. 1. (i) User's expected download rate, and (ii) user's expected upload rate first region corresponds to $P_{tkn}^{FR} < 0.32$. In this region, both classes of peers are constrained to a lower download rate compared to the original system, since the probability of having tokens to initiate a new download after a successful query is pretty low. Notice that for $P_{tkn}^{FR} = 0.32$, and hence for $P_{tkn}^{NF} = 0.47 < 1$, cooperative users can at least sustain the same download rate they had in the original system. The second region corresponds to $0.32 \leq P_{tkn}^{FR} \leq 0.55$. In this region, users accumulate tokens at a higher rate than before. Since there are more responses than in the original network, users can use the extra tokens to initiate more downloads. Notice that cooperative users earn tokens faster than free-riders since they always respond to query requests. At $P_{tkn}^{FR} = 0.55$, non-freeriders achieve their maximum download rate, which is approximately twice the one they had in the original system. Finally, the third region corresponds to $0.55 < P_{tkn}^{FR} \leq 1$. In this region free-riders accumulate tokens faster than before and reduce their query response rate since they do not need to provide as many uploads as before. This causes cooperative users to handle more uploads. Futher, since the query response rate regulates the download rate, the latter also decreases. At $P_{tkn}^{FR} = 1$, the two systems have approximately the same performance, as expected.

Impact on Delays. Figures 2(i) and 2(ii) show respectively the average query response time (that includes retransmissions) and the average download delay as a function of P_{tkn}^{FR}. The plots can be divided in the same three regions as before. For $P_{tkn}^{FR} < 0.32$, the low user download rate imposes a low load into the network. This yields the low delays. For $0.32 \leq P_{tkn}^{FR} \leq 0.55$, as the user download rate increases, the load in the network and hence the delays also increase. Note that the query and download delays are still significantly smaller than in the original system, despite that the download rate, and hence the load, is higher. This is because a significant portion of the load is now handled by the free-riders. For $0.55 < P_{tkn}^{FR} \leq 1$ the delays continue to increase even though the download rate decreases. This is because free-riders provide fewer and fewer uploads. As P_{tkn}^{FR}

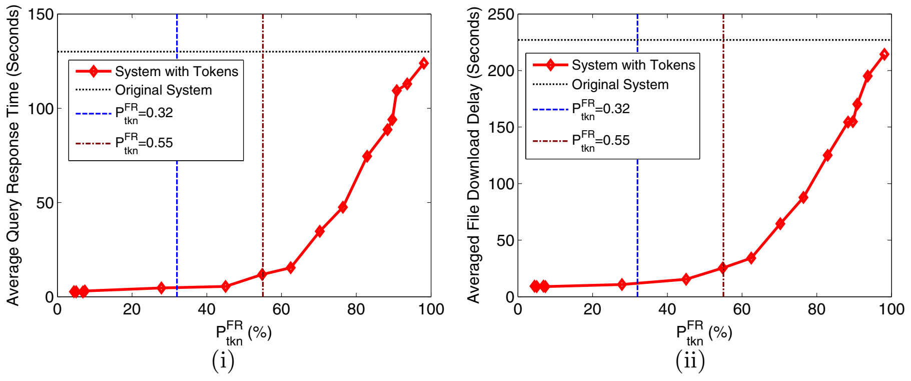


Fig. 2. (i) Average query response time, and (ii) average file download delay

approaches 1, the performance of the two systems is approximately the same. To fairly compare the delays between the two systems, we should consider the case where the load is the same, i.e. where $E[R_{down}] = 22$ downloads/1000sec. This value corresponds to $P_{tkn}^{FR} = 0.45$, and as we can see from the plots this corresponds to approximately one order of magnitude lower query and file download delays. This is a gigantic amount of improvement on the system's performance.

As a final note, the best operating region is the second, where $0.32 \leq P_{tkn}^{FR} \leq 0.55$. In this region, we can either choose to operate the system at $P_{tkn}^{FR} = 0.32$, where cooperative users can sustain the same download demands as in the original system, or sacrifice a bit from the performance improvement with respect to reduced delays to support higher user demands.

6 Implementing the Scheme

As mentioned before, this scheme can be implemented either locally or non-locally. Implementing this scheme locally is quite simple. The local P2P client takes care of bookkeeping by increasing the user's tokens for each acknowledged byte he/she uploads and for being online, and by decreasing the tokens for each byte the user downloads. However as we have already mentioned, this approach is quite vulnerable to hacked clients.

There are several directions for making the hacking of localized solutions hard. One can utilize encryption techniques *e.g.* [26] that make unauthorized modifications to data (such as the scheme's parameters) hard. In addition, one can also use technologies like DRM (Digital Rights Management) in order to protect the entire client's code from being altered *e.g.* [27], and re-distribute new clients frequently in order to minimize the number of hacked clients that can be connected to the network. Further, one could also employ techniques such as tamper-proofing and self-checking in order to verify the client's code integrity during the join process and/or on every download request *e.g.* [28, 29, 30, 31]. Of course, the only way to guarantee that all P2P clients are original is to have a trusted platform where both the hardware and the operating system can be trusted [32]. This is clearly not an option in practice. However, interestingly enough, both theory and simulations dictate that our scheme is quite resilient to a *small* number of hacked clients. In particular, the system performance is virtually unchanged when the hackers comprise less than 10% [19]. Hence, all one needs to do is to make it hard for users to use hacked clients.

The scheme can be also implemented in a secure non-localized centralized manner, where peers exchange messages with a centralized trusted authority that updates and maintains their amount of tokens. Peers would communicate with the centralized authority once they finish downloading to report the source node and the file size, periodically while being online, and to get permission to initiate a new download. This is similar to the main idea that many centralized "cash-based" systems, *e.g* [3, 4], follow. Finally, the scheme can be also implemented in a secure non-localized decentralized manner, *e.g.* by utilizing the framework suggested in [6].

7 Conclusion

In this paper we studied a simple algorithm that provides strong incentives for cooperation in file sharing P2P networks. We derived a mathematical model that describes the system's dynamics and which can be used for parameter tuning and performance prediction. We demonstrated the effectiveness of the algorithm via experiments with TCP networks. Future work consists of performing larger scale experiments, implementing the scheme in an operational P2P network, and extending our analytical methodology to compute other important performance metrics, e.g. the improvement on the expected download delays and response times.

References

1. E. Adar and B. Huberman, "Free riding on gnutella," `http://www.firstmonday.dk/issues/issue5_10/adar`, October 2000 (accessed Aug. 2005).
2. D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: the Bell Tolls?," *IEEE Distributed Systems Online Journal*, vol. 6, no. 6, June 2005.
3. "Mojonnation," `http://www.mojonation.net/Mojonation.html` (accessed Aug. 2005).
4. J. Ioannidis, S. Ioannidis, A. Keromytis, and V. Prevelakis, "Fileteller. paying and getting paid for file storage," in *Proc. of 6th International Conference on Financial Cryptography*, March 2002.
5. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "EigenRep: Reputation management in P2P networks," in *Proc. of 12th International World Wide Web Conference (WWW 2003)*, May 2003.
6. V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "KARMA: A secure economic framework for P2P resource sharing," in *1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.
7. C. Buragohain, D. Agrawal, and S. Suri, "A game-theoretic framework for incentives in P2P systems," in *Proc. of International Conference on Peer-to-Peer Computing*, Sep 2003.
8. L. Ramaswanmy and L. Liu, "Free-riding: A new challenge to peer-to-peer file sharing systems," in *Proc. of the 36th Hawaii international conference on system sciences*, 2003.
9. M. Feldman, C. Papadimitriou, I. Stoica, and J. Chuang., "Free-riding and whitewashing in Peer-toPeer systems," in *SIGCOMM Workshop*, 2004.
10. "KaZaA participation level," `http://www.kazaa.com/us/help/glossary/participation_ratio.htm` (accessed Aug. 2005).
11. "The emule project," `http://www.emule-project.net/` (accessed Aug. 2005).
12. "KaZaA media desktop," `http://www.kazaa.com/` (accessed Aug. 2005).
13. "Bittorrent," `http://www.bittorrent.com/protocol.html` (accessed Aug. 2005).
14. H. Bretzke and J. Vassileva, "Motivating cooperation in peer to peer networks," in *Proc. of User Modeling UM03*, June 2003.
15. "Hack KaZaA participation level," `http://www.davesplanet.net/kazaa/` (accessed Aug. 2005).
16. K. Anagnostakis and M. Greenwald, "Exchanged-based incentive mechanisms for peer-to-peer file sharing," in *Proc. of 24th International Conference on Distributed Computing Systems*, 2004.

17. R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability," in *Proc. of 2nd IPTPS*, 2003.
18. R. Durrett, *Probability: Theory and Examples*, Duxbury Press, 2nd edition, 1996.
19. W.-C. Liao, F. Papadopoulos, and K. Psounis, "An efficient algorithm for resource sharing in peer-to-peer networks," Tech. Rep. CENG-2005-15, University of Southern California, 2005.
20. S. Saroiu, K. P. Gummadi, R. J. Dunn, S.D. Gribble, and H. M. Levy, "An analysis of internet content delivery systems," in *Proc. of the Fifth Symposium on Operating System Design and Implementation (OSDI)*, December 2002.
21. J. Chu, K. Labonte, and B. N. Levine, "Availability and locality measurements of peer-to-peer file sharing systems," in *Proc. of SPIEITCom: Scalability and Traffic Control in IP Networks*, July 2002.
22. Mostafa Amar Qi He, "Congestion control and massage loss in gnutella networks," in *Proc. of Multimedia Computing and Networking*, 2004.
23. "Packet-level Peer-to-Peer Simulation Framework and GnutellaSim," `http://www.cc.gatech.edu/computing/compass/gnutella/` (accessed Oct. 2005).
24. "Network simulator," `http://www.isi.edu/nsnam/ns` (accessed Sep. 2005).
25. K. Calvert, M. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, 1997.
26. "Data encryption standard," `http://www.itl.nist.gov/fipspubs/fip46-2.htm` (accessed Oct. 2005).
27. T. Sander, *Security and Privacy in Digital Rights Management*, Springer, 1st Edition, 2002.
28. D. Aucsmith, "Tamper resistant software: An implementation," in *Proc. 1st International Information Hiding Workshop*, May 1996.
29. H. Chang and M. Atallah, "Protecting software code by guards," in *Proc. of 1st ACM Workshop on Digital Rights Management*, May 2002.
30. Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. Jakubowski, "Oblivious hashing: A stealthy software integrity verification primitive," in *Proc. of 5th International Information Hiding Workshop*, October 2002.
31. C.S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 6, June 2002.
32. S. W. Smith, *Trusted Computing Platforms: Design and Applications*, Springer, 1st Edition, 2004.