An Incentive Mechanism for P2P Networks

Richard T. B. Ma[†] Sam C. M. Lee[†] John C. S. Lui[†] David K. Y. Yau[‡]

***Abstract*— The current peer-to-peer (P2P) information sharing paradigm does not provide incentive and service differentiation for users. Since there is no motivation to share information or resources, this leads to the "free-riding" and the "tragedy of the commons" problems. In this paper, we address how one can incorporate incentive into the P2P information sharing paradigm so as to encourage users to share information and resources. Our mechanism (or protocol) provides service differentiation to users with different contribution values and connection types. The mechanism also has some desirable properties: (1) conservation of cumulative contribution and social utility in the P2P community, (2) maximization of social utility if all requesting clients have the same contribution value, and (3) incentive-based resource distribution. The resource distribution algorithm and the contribution update algorithm are computationally efficient and can be easily implemented. We discuss how one can provide the truth revealing property so that nodes have no motivation to misrepresent their contribution values and connection types. In addition, the mechanism is invulnerable to various malicious actions of users. Experimental results illustrate the efficiency and fairness of our algorithms.**

I. Introduction

The rapid growth of decentralized and structured or unstructured peer-to-peer (P2P) networks [17], [14], [12] holds great potential for efficient information exchange in the Internet. A P2P network may exhibit a power-law topology [13] such that it can propagate queries quickly and, if implemented efficiently [17], it can locate objects in $\log(n)$ time, where n is the number of nodes in the network. However, there are remaining problems in the P2P information sharing paradigm which complicate its deployment. *Free-riding* and the *tragedy of the commons* are two major problems. As reported in [3], nearly 70% of Gnutella users do not share any file with others in a P2P community and nearly 50% of all search responses come from the top 1% of content sharing nodes. Therefore, nodes that share information and resources are prone to congestion, leading to the tragedy of the commons [9]. Another problem is that many users intentionally misrepresent their connection speeds so as to discourage others from going to their nodes for file download. Worse yet, Gnutella-like systems give no service differentiation between users who do not share any information with or make any contribution to the P2P community.

The objective of this paper is to design and analyze a protocol that provides incentives for users to share information and offers preferential service to users who contribute to the P2P community. We address the following issues:

1) *How to utilize transfer bandwidth resources efficiently?*
2) *How to fairly serve different nodes which have different connection types and contributions in a P2P community?*
3) *How to avoid problems of free-riding and the tragedy of the commons?*
4) *How to provide the truth revealing property so that nodes have no motivation to misrepresent their connection types or contributions and, at the same time, avoid various malicious attempts by users to gain better services?*

Many current P2P systems use the first-come-first-served policy in providing file transfer services [2]. This may cause large response time or even starvation for requests queued after other long running requests. Alternatively, round robin scheduling can be used. However, evenly distributing the transfer bandwidth between requesting users may not be suitable. First, this may not be an efficient choice for the P2P network since different nodes may have different connection types (e.g., modem, LAN, ADSL) and speeds, and they may achieve different utilities even if given the same amount of transfer bandwidth resource. Instead, one should consider the problem of distributing the transfer bandwidth resource so as to maximize the aggregate utility. Second, it may not be fair since some requesting nodes may have contributed a lot more than other requesting nodes. These considerations lead us to propose a scheduling policy which is based on the aggregate utility, the connection types and contribution values of individual requesting nodes. Such a policy gives a rational user *incentive* to share information and contribute service to a P2P community.

The balance of the paper is organized as follows. In Section II, we introduce the notations and model of our incentive P2P network. In Section III, we present the algorithms for resource distribution. In Section IV, we present an algorithm for computing the contribution values of all participating nodes. In Section V, we discuss how to avoid various malicious actions by users and enforce the truth revealing property: users have to reveal their true contribution values, connection types, and upload bandwidth. In Section VI, we report experimental results to illustrate the dynamics between the contribution values of participating nodes and their received bandwidth. Related work is discussed in Section VII. Section VIII concludes.

II. Incentive P2P Networks

In this section, we first present a model of the Gnutella protocol – a common P2P network protocol – and some of its inherent problems. Although we compare and contrast our work with Gnutella, the proposed mechanism in this paper can be used to give incentive in more recent P2P protocols, such as [17]. We then present the notations in our design of an incentive-based protocol, and state some of the protocol's desirable properties.

[†]Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, N.T. Hong Kong; {tbma,cmlee,cslui}@cse.cuhk.edu.hk.
[‡]Department of Computer Science, Purdue University, West Lafayette, IN; yau@cs.purdue.edu.

A. The Gnutella protocol and its inherent problems

Gnutella [1], an open P2P protocol for connection management and distributed search, represents a class of decentralized unstructured P2P networks. In a Gnutella-like network, each node (also called a *servent*) plays the role of both a client and a server. The Gnutella protocol specifies rules for sending/answering queries and maintaining the connectivity between different servents. Each servent joins a P2P network by connecting to some existing servents in the network. A servent, say i, performs file searching by sending queries to its neighbors, which can in turn forward the query to their own neighbors. Once the file is located in a set of servents, say $\mathcal{S}$, servent i can request a file transfer from any servent $j \in \mathcal{S}$.

To formally describe the *logical* and *physical* views of the file transfer process, we define the following notations:

$\mathcal{N}$: A set of all servents in a P2P system with $|\mathcal{N}| = N$.
$\Lambda = [\lambda_{i,j}]_{N*N}$, where $\lambda_{i,j}$ represents the average file transfer request rate from servent i to servent j.
$\Psi = (u_1, u_2, \ldots, u_N)$, where u_i, $i \in \mathcal{N}$, represents the *maximal* upload bandwidth (in Mbps) of servent i.
$\mathcal{D} = (d_1, d_2, \ldots, d_N)$, where d_i, $i \in \mathcal{N}$, represents the *maximal* download bandwidth (in Mbps) of servent i.
$\mathcal{R}_k$: The set of servents which may request file download from servent k; i.e., any servent j for which $\lambda_{j,k} > 0$.

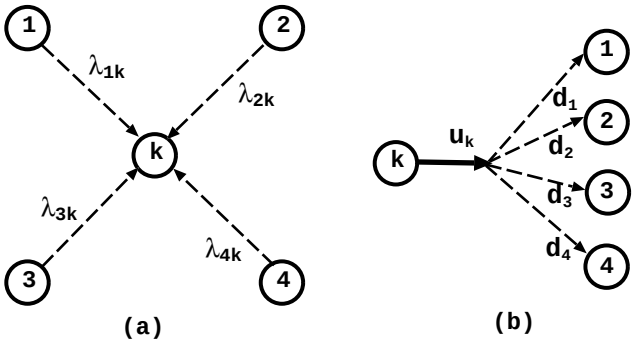

Fig. 1. (a) logical and (b) physical views of file transfers in a P2P network.

Figure 1 illustrates the logical and physical views of the file transfer process. Figure 1(a) depicts the query/search process. Servents $1, 2, 3$, and 4 have found that servent k has the file that they are searching for, and they decide to request the file from servent k. Servent k has a physical download bandwidth of d_k and an upload bandwidth of u_k (both in Mbps). Typically, we have $d_k \geq u_k$. For example, for an ADSL connection, we have $d_k > u_k$. But for other full-duplex network technologies (e.g, Ethernet and ATM), we have $d_k = u_k$. Therefore, to transfer the requested files, servents $1, 2, 3$, and 4 have to *share* the upload bandwidth u_k of servent k. This is illustrated in Figure 1(b).

In this paper, we assume that there are N servents in the P2P network. We define Λ to be an $N \times N$ matrix with $\lambda_{i,j}$ denoting the average file transfer request rate from servent i to servent j. A servent, who shares certain contents in a P2P community, receives different rates of request from other servents. In general, the request rates depend on (1) the popularity of the contents offered by the target servent, and (2) the target servent's upload bandwidth capacity.

The Gnutella protocol specifies that the file transfer is to be carried out over HTTP. Most Gnutella client implementations maintain multiple HTTP connections, but limit the maximum number of such concurrent HTTP transfers at any given time. For example, LimeWare [2] uses *upload slots* to limit the number of HTTP connections. When the upload slots are used up, new file transfer requests are queued and LimeWare uses the FCFS scheduling policy to process the waiting requests. With FCFS, requests may experience a long waiting time. For example, if the active HTTP sessions are being occupied with large file transfers, requests in the queue will have to wait for a long time before receiving service.

To reduce the waiting time, an alternative solution is to use some form of processor-sharing discipline in scheduling the file transfer requests. However, such an approach has its own problems. First, the simple strategy of giving an equal share of the transfer capacity to each requesting node can be inefficient, because the download capacity of some of the requesting nodes may be smaller than their allocated capacity share. Second, even if all the requesting servents can fully utilize their bandwidth share, the equal resource allocations ignore the more relevant issue of end user's satisfaction. For example, different requesting servents may have different utility functions, which quantify the servents' "degree of happiness" when they receive different amounts of the transfer bandwidth resource. Lastly, an equal allocation strategy obviously gives no incentive for servents to contribute to their peers.

B. Incentive protocol: notations & desirable properties

Lack of incentive for sharing leads to the undesirable situation in which a servent behaves like a client most of the time [16]. The design of an incentive protocol for P2P networks is imperative. In such a protocol, the proper allocation of transfer bandwidth to requesting servents should be based on the servents' connection type, utility function, and contribution to the P2P community. Before we present our incentive protocol, we give the necessary notations:

$\Theta = (\theta_1, \theta_2, \ldots, \theta_N)$: A vector which represents the *connection type* (i.e., the upload and download capacities) of all the servents in the P2P network. In particular, $\theta_i \in \Omega$ is the connection type of servent i, which is a function of i's declared upload bandwidth u_i and download bandwidth d_i. The set Ω represents all the possible connection types.
$C_i(t)$ represents the *cumulative contribution* of servent i at time t, where $C_i(t) \in \{I\!R^+ \bigcup 0\}$.
$x_i(t)$ represents the bandwidth allocated to servent i when i requests a file transfer. The bandwidth assignment is based on our incentive protocol.
$U_i(\theta_i, x_i)$: A non-negative function which represents the utility of servent i when it declares its connection type to be θ_i and receives a file transfer service rate of x_i.

Each servent in the system, say i, has a cumulative contribution value $C_i(t)$ at time t. The value of $C_i(t)$ will increase

if servent i provides service to the community (e.g., by transferring files for other requesting servents). It may decrease if servent i requests some service from the community (e.g., requesting file transfers from other servents).

We now state some *desirable properties* of an incentive protocol. In later sections, we will prove that our proposed algorithms achieve these properties.

(1) Conservation of the cumulative contribution and social utility: The aggregate contribution of all servents at any time $t > 0$ is equal to the aggregate cumulative utility of all servents from time 0 up to time t. Formally,

$$\sum_{i=1}^{N} C_i(t) \quad = \quad \sum_{i=1}^{N} \int_0^t U_i(\theta_i, x_i(\tau))d\tau \qquad \forall t > 0. \quad (1)$$

Remark: This property implies that the contribution by any servent in a P2P network via file transfer service is translated into utilities within the P2P community.

(2) Maximization of social welfare through resource allocation: Given a servent k and all the requesting clients in $\mathcal{R}_k$, if $C_i(t) = C_j(t)$ for all $i, j \in \mathcal{R}_k$, an incentive protocol should allocate the transfer bandwidth resource of servent k so as to maximize the *social welfare*. Formally, maximizing the social welfare implies finding a transfer bandwidth vector $\boldsymbol{x}(t) = [x_i(t), \cdots, x_N(t)]$ such that

$$\boldsymbol{x}(t) = \arg\max \left(\sum_{i \in \mathcal{R}_k} U_i(\theta_i, x_i(t)) \right) \quad \text{s.t.} \sum_{i \in \mathcal{R}_k} x_i(t) \leq u_k. \quad (2)$$

Remark: This property implies that the incentive protocol should maximize the aggregate utility (or "happiness") of all the requesting servents in $\mathcal{R}_k$.

(3) Incentive-based resource distribution: The protocol should provide incentive to rational users. Given a servent k and all the requesting clients in $\mathcal{R}_k$, we have two cases:

• **No Congestion:** If the aggregate download bandwidth at time t of all the requesting servents in $\mathcal{R}_k$ is less than or equal to u_k, the upload bandwidth of servent k, then all servents in $\mathcal{R}_k$ will receive a transfer bandwidth equal to their respective maximal download bandwidth such that they will achieve equal utility. Formally, if $\sum_{i \in \mathcal{R}_k} d_i \leq u_k$, then

$$\begin{aligned} x_i(t) &= d_i \\ U_i(\theta_i, x_i(t)) &= U_j(\theta_j, x_j(t)) \qquad \forall i, j \in \mathcal{R}_k. \end{aligned} \quad (3)$$

Remark: This property implies that whenever servent k has sufficient resources, all the requesting servents should receive their maximal download bandwidth such that they are "equally happy".

• **Congestion:** When there is a congestion for servent k (i.e., $\sum_{i \in \mathcal{R}_k} d_i > u_k$), the transfer bandwidth allocation should be a function of the contribution and download bandwidth of all the requesting servents in $\mathcal{R}_k$. Formally, for any two servents $i, j \in \mathcal{R}_k$, if the ratio of contribution to download bandwidth of i is greater than or equal to that of j, servent k will distribute the transfer bandwidth resource such that the utility of servent i is greater than or equal to that of servent j. Formally,

$$\frac{C_i(t)}{d_i} \geq \frac{C_j(t)}{d_j} \Longrightarrow U_i(\theta_i, x_i(t)) \geq U_j(\theta_j, x_j(t)). \quad (4)$$

Remark: This property implies that the incentive protocol should provide higher utilities to servents who have higher contributions per unit data request.

In the following, we present the operational setting of our incentive protocol.

C. Operational setting of incentive P2P protocol

The general setting in which our incentive protocol operates is as follows:

• Each servent has to declare its connection type to the P2P community[1], i.e., servent i has to declare its connection type of θ_i. For our incentive protocol, the connection type of servent i depends only on the upload (u_i) and download bandwidths (d_i).

• To provide fairness and incentive for a P2P community, the utility function, say for servent i, takes on a *concave*, *bounded*, and *normalized* form. The utility function of servent i, which depends on the download bandwidth d_i and the received transfer bandwidth x_i, takes the form:

$$U_i(\theta_i, x_i) = U_i(d_i, x_i) = \begin{cases} \log(\frac{x_i}{d_i} + 1) & \text{if } x_i \leq d_i \\ \log(2) & \text{if } x_i > d_i. \end{cases} \quad (5)$$

Remark: We take this form of utility function based on the following reasons: (a) A $\log$ function is a general form of concave function which can represent a large class of elastic traffic [15] and this fits the file transfer service; (b) The utility function has an upper bound of $\log(2)$, which implies that once a servent receives its maximum download bandwidth, they are *equally satisfied*; (c) The utility function has a value of zero if the received bandwidth $x_i = 0$; (d) The utility function estimates the level of satisfaction given the ratio $\frac{x_i}{d_i}$, the amount of allocated bandwidth to the servent's maximal download bandwidth.

We adopt concepts from *mechanism design* [11]. Under our incentive protocol:

(1) All servents have to declare their connection types. Hence, servent i has a strategy g_i which can declare any connection type $\theta \in \Omega$, where Ω is the set of all connection types in our incentive P2P system. For an honest servent, which can be induced by a protocol having the truth revealing property, the strategy of servent i should be $g_i = \theta_i = (u_i, d_i)$. That is, servent i declares its real connection type.

(2) We interpret $C_i(t)$, the *contribution* of servent i, as the virtual credit that servent i has at time t. The proposed protocol will update the contributions of all the participating servents in any file transfer activity. In particular, the incentive protocol will increase the contribution when a servent offers file transfer service for any requesting servent, and it may reduce the contribution of a servent who requests a file download. Particularly, the initial value of contribution is assigned to be zero which implies this servent has not provided any service to others.

[1] We will address the truth revealing property in a later section.

(3) The outcomes of the proposed protocol are (i) how much transfer bandwidth is to be allocated to each requesting servent, and (ii) the contribution updates for each participating servent.

III. Incentive Protocol for Distributing the Instantaneous Transfer Bandwidth

In this section, we describe the incentive protocol for allocating the instantaneous transfer bandwidth to requesting servents. For ease of discussion, we drop the time dependent notation; i.e., we use x_i instead of $x_i(t)$. Our incentive protocol can achieve *efficiency* for social welfare. Furthermore, it provides *fairness* and *incentive* for sharing resources among all requesting servents. We first illustrate how the incentive protocol can maximize the social welfare. Then we generalize the concept and extend the protocol to include the contribution value of each requesting servent.

A. Protocol to maximize the social welfare

Consider a servent k that is willing to offer its transfer bandwidth resource for use by a set of requesting servents $\mathcal{R}_k$. If all the servents in $\mathcal{R}_k$ are of the same contribution (or if we ignore the contribution factor for the time being), maximizing the social welfare implies finding a transfer bandwidth vector $\boldsymbol{y} = [y_1, \cdots, y_{|\mathcal{R}_k|}]$ such that:

$$SW(u_k, \mathcal{R}_k) := \max \sum_{i \in \mathcal{R}_k} U_i(\theta_i, y_i)$$
$$\text{s.t.} \sum_{i \in \mathcal{R}_k} y_i \le u_k \; ; \; 0 \le y_i \le d_i \quad \forall i \in \mathcal{R}_k. \tag{6}$$

Here, y_i is the allocated transfer bandwidth for servent i in solving the above maximization problem. For our concave and bounded utility functions, we have the following equivalent optimization problem:

$$\max \prod_{i \in \mathcal{R}_k} (y_i + d_i)$$
$$\text{s.t.} \quad \sum_{i \in \mathcal{R}_k} y_i \le u_k \; ; \; 0 \le y_i \le d_i \quad \forall \; i \in \mathcal{R}_k. \tag{7}$$

One way to solve the optimization problem is to try to distribute the resource u_k such that the $(y_i + d_i)$s are as *even* as possible for all $i \in \mathcal{R}_k$. For instance, if we were without constraints, the solution should satisfy:

$$y_i + d_i = y_j + d_j \quad \forall i, j \in \mathcal{R}_k. \tag{8}$$

We use and enhance the progressive filling algorithm [4] to solve the above constrained optimization problem. Our progressive filling algorithm works as follows:

1) Treat a requesting client $i \in \mathcal{R}_k$ as a water bucket with a capacity equals to $2d_i$ and a height equals to $2d_i$.
2) Based on the values of d_i, sort all buckets in ascending order. For bucket i, the initial water level is d_i.
3) In addition, we have u_k amount of "water" (the resource) to distribute to all the buckets. We distribute the water such that the *maxmin fairness* property [4] holds.

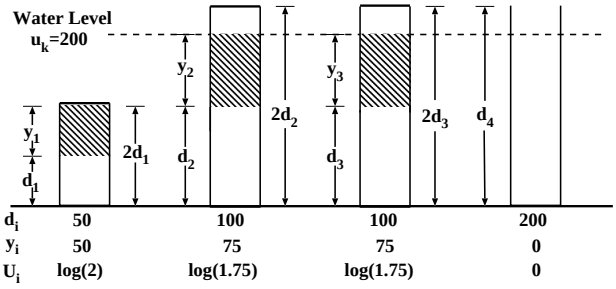

Fig. 2. Example for the Progressive Filling Alg.

We use a simple example to show how the algorithm works. Suppose there are four requesting servents competing for the transfer bandwidth of servent k with $u_k = 200$. The download capacities for the four requesting servents are $[d_1, d_2, d_3, d_4] = [50, 100, 100, 200]$. Figure 2 illustrates the bandwidth allocation result by the progressive filling algorithm, which is $\boldsymbol{y} = [y_1, y_2, y_3, y_4] = [50, 75, 75, 0]$. The solution maximizes the social welfare at a value equal to $\log(2)+\log(1.75)+\log(1.75)$. This example shows that the algorithm biases toward servents with smaller download capacities. The intuition is that given the same unit of bandwidth resource, a servent which has a smaller capacity will reach the maximal utility of $\log(2)$ with a *higher rate* than servent with a larger capacity. Therefore, by assigning the bandwidth to smaller-capacity users, we can achieve a higher social welfare value.

Theorem *1: The progressive filling (PF) algorithm finds a solution to the bandwidth allocation problem which maximizes the social welfare in Eq. (6).*

Proof: Let $f(\boldsymbol{y}) = \sum_{i \in \mathcal{R}_k} U_i(\theta_i, y_i) = \sum_{i \in \mathcal{R}_k} \log(\frac{y_i}{d_i} + 1)$. Suppose the algorithm gives the solution $\boldsymbol{y}^*$. For any bucket i such that $y_i^* > 0$, if $\frac{1}{y_i^* + d_i} < \frac{1}{y_j^* + d_j}$, we know that bucket j has a value $(y_j^* + d_j)$ and it is under the "water" level. In this case, we have $y_j^* = d_j$ and we cannot allocate more resource to bucket j. On the other hand, if $\frac{1}{y_i^* + d_i} > \frac{1}{y_j^* + d_j}$, then bucket j has value $(y_j^* + d_j)$ which is above the "water" level. $y_j^* = 0$ implies that we cannot obtain a better solution by moving any resource from j to other buckets. So the solution of the PF algorithm is guaranteed to be a local maximum. Lastly, since the function f is continuous over a convex constraint set, the local maximum solution is also a global maximum solution. ■

B. Incentive protocol

In the above discussion, we distribute the transfer bandwidth u_k among the requesting servents without considering their contribution values. We now extend the solution to include the contribution value of each requesting servent in order to provide incentive for contributing to the P2P community.

In maximizing the social welfare, Equation (8) implies that for any two requesting servents $i, j \in \mathcal{R}_k$, we distribute the transfer bandwidth resource such that $(y_i + d_i)/(y_j + d_j) = 1$. To provide incentive, we distribute the transfer bandwidth resource u_k of servent k such that the transfer bandwidth vector

$\boldsymbol{x} = [x_1, \cdots, x_{|\mathcal{R}_k|}]$ satisfies

$$\frac{x_i + d_i}{x_j + d_j} = \left(\frac{C_i}{C_j}\right)^r \quad \forall i, j \in \mathcal{R}_k \tag{9}$$

where r is any nonnegative real number. Clearly this is a generalization of the problem of maximizing the social welfare. E.g., if all the requesting clients have the same contribution values (i.e., $C_i = C_j$ for $i, j \in \mathcal{R}_k$), the above formulation is equivalent to Equation (8).

Based on the progressive filling (PF) algorithm given above, we propose an enhanced *contribution dependent* progressive filling (CDPF) algorithm. In essence, the new algorithm tries to satisfy Equation (9) among the requesting servents, if feasible. In doing so, the algorithm also maintains the maxmin fairness property. The CDPF algorithm works as follows:
(1) Treat a requesting client $i \in \mathcal{R}_k$ as a water bucket with a capacity equal to $2d_i$ and a height equal to $2d_i/(C_i)^r$.
(2) Based on the value $d_i/(C_i)^r$, sort all the buckets in ascending order. For bucket i, the initial water level is $d_i/(C_i)^r$.
(3) Distribute u_k amount of water (the resource) to all the buckets. To fill each unit for bucket i, we consume $(C_i)^r$ amount of water.

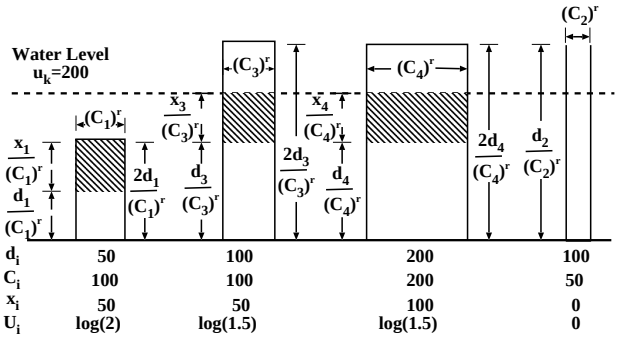

Fig. 3. Example for the Contribution Dependent Progressive Filling Algorithm with $r = 1$.

To illustrate the algorithm, we use the same scenario as in the last example. The contribution values of the four requesting servents in this case are $[C_1, C_2, C_3, C_4] = [100, 50, 100, 200]$, and $r = 1$. Figure 3 illustrates the bandwidth allocation result produced by the CDPF algorithm. Observe that the height of each bucket represents the filling range of each servent and the width represents the unit of allocation for each servent, i.e., C_i for servent i in the problem formulation. The allocated transfer bandwidth vector is $\boldsymbol{x} = [x_1, x_2, x_3, x_4] = [50, 0, 50, 100]$. Note that the solution gives *more* of the resource to a servent which has a higher contribution value (e.g., servent 4 in this case). Also note that the solution does not maximize the social welfare, but provides *fairness* in the form of service differentiation among the servents according to their contributions. It also provides incentive for servents to share their resources. The incentive property can be formally stated as the following theorem.

Theorem *2: For any two requesting servents $i, j \in \mathcal{R}_k$, the CDPF algorithm distributes the transfer bandwidth resource such that:*

$$\frac{(C_i)^r}{d_i} \geq \frac{(C_j)^r}{d_j} \Longrightarrow U_i(\theta_i, x_i) \geq U_j(\theta_j, x_j).$$

Proof: When $\frac{(C_i)^r}{d_i} \geq \frac{(C_j)^r}{d_j}$, the stated condition in Theorem 2 is equivalent to:

$$\frac{d_i}{(C_i)^r} \leq \frac{d_j}{(C_j)^r}. \tag{10}$$

So initially, bucket i has a lower water level than bucket j. Therefore, bucket i will hit its capacity faster than j. In the final bandwidth distribution, we have:

$$\frac{x_i + d_i}{(C_i)^r} \leq \frac{x_j + d_j}{(C_j)^r}. \tag{11}$$

When Equation (11) meets the strictly less than condition, this implies that $x_i = d_i$. In this case, bucket i is fully satisfied and reaches its maximal utility value of $U_i(\theta_i, x_i) = \log 2$. Therefore, $U_i(\theta_i, x_i) \geq U_j(\theta_j, x_j)$. When Eq. (11) meets the equality condition, we divide Equation (10) by $\frac{x_i+d_i}{(C_i)^r} = \frac{x_j+d_j}{(C_j)^r}$, which gives:

$$\begin{aligned} \frac{d_i}{x_i + d_i} \leq \frac{d_j}{x_j + d_j} &\Longrightarrow \frac{x_i + d_i}{d_i} \geq \frac{x_j + d_j}{d_j} \\ &\Longrightarrow \log\left(\frac{x_i + d_i}{d_i}\right) \geq \log\left(\frac{x_j + d_j}{d_j}\right) \\ &\Longrightarrow U_i(\theta_i, x_i) \geq U_j(\theta_j, x_j) \end{aligned}$$

■

Remark: The significance of Theorem 2 is that our incentive protocol possesses the desired properties (2) & (3) given in Section II-B.

The CDPF can be implemented by the following code:

CDPF (r,u_k, all requesting servents in $\mathcal{R}_k$)
1. **if** ($\sum_{i \in \mathcal{R}_k} d_i \leq u_k$) **return x=d**;/*no congestion*/
2. **sort** $\{\frac{d_i}{(C_i)^r}, \frac{2d_i}{(C_i)^r} | i \in \mathcal{R}_k\}$ **in ascending order. Store values and node index in array S and T respectively**;
3. i=1; /*initialize index variable*/
4. level=$d_{T[1]}$;/*initialize feasible water level*/
5. vol=$C[T[1]]^r$;
   /*initialize resource level for all buckets*/
6. **do** {
7. i=i+1;
8. nextLevel=S[i];/*the next testing water level*/
9. **if** ((nextLevel−level)∗vol≥ u_k) {
   /*can't move to next level*/
10. level = level+u_k/vol;
11. $u_k = 0$;
12. }
13. **else**{
   /*move to next feasible level*/
14. u_k=u_k-vol*(nextLevel-level);
15. level=nextLevel;
   /*adjust the unit for filling operation*/
16. **if (S[i] is a lower bound of T[i])** vol=vol+$(C_{T[i]})^r$;

17. **else** vol=vol-$(C_{T[i]})^r$;
18. }
19. } **while** ($u_k > 0$);
20. **for** (**each** $i \in \mathcal{R}_k$)
21. **if** (level> $\frac{d_i}{(C_i)^r}$) x_i=[level-$\frac{d_i}{(C_i)^r}$]*$(C_i)^r$;
22. **return x**;

Our algorithm will assign the bandwidth x_i as equal to d_i, which is the maximal download bandwidth of servent i, for all the requesting servents if the aggregate maximal download bandwidth is less than or equal to the upload bandwidth resource (i.e., $\sum_{i\in\mathcal{R}_k} d_i < u_k$). Otherwise, our algorithm sorts all the lower bounds $\{\frac{d_i}{(C_i)^r}\}$ and upper bounds $\{\frac{2d_i}{(C_i)^r}\}$ in ascending order. Then it tests whether the amount of resource u_k can fill the buckets for reaching these bounds. We initialize the starting water level to be the minimum value of the sorted bounds. The initial marginal amount of water for filling a higher water level is $(C[T[1]])^r$, where $T[1]$ is the index of the servent which has the minimum lower bound. Within each iteration of the do-while loop, we test whether the bound for the next bucket in S can be reached. If it can be reached, we first reduce the remaining resource in u_k, then we adjust $(C[i])^r$ if servent i's bound can be reached, and we assign the transfer bandwidth to all the eligible requesting servents. If the bound cannot be reached, the algorithm terminates and we have found the transfer bandwidth distribution.

Theorem *3: The CDPF algorithm has computational complexity $O(n\log(n))$, where n is the number of requesting servents in $\mathcal{R}_k$.*

Proof: In line 2 of the CDPF algorithm, the sorting operation takes $O(2 * |\mathcal{R}_k|\log(2 * |\mathcal{R}_k|))$ time. The do-while loop between line 6 and 19 takes constant time and, at most, we go through this loop $2 * |\mathcal{R}_k|$ number of times. Therefore, the computational complexity of the CDPF algorithm is $O(2 * |\mathcal{R}_k|\log(2 * |\mathcal{R}_k|))$. Since $|\mathcal{R}_k| = n$, after simplifying, the computational complexity is $O(n\log(n))$. ∎

The proposed algorithm is efficient. A file sharing node only needs to update the resource distribution whenever a new requesting servent initiates a file transfer, or an existing requesting servent finishes the file transfer process. Therefore, our decentralized mechanism is scalable and can handle tens of thousands of nodes joining or leaving a P2P network since not all these nodes are targeting one specific file sharing node for the file transfer at the same time. Finally, although our mechanism needs the information about contribution and connection type of requesting servents, the information needs to be available only when the requesting servent joins the resource competition. Therefore, a file sharing servent does not need to have the information for the whole P2P network.

IV. Contribution Update

After each file transfer activity, we need to update the contribution values of all the participating servents. We give the physical meaning of contribution values and discuss how they should be updated. We will then present our contribution update algorithm and give its complexity analysis. Security issues for the contribution update will be discussed in a later section.

A. Social gain and social payment

In P2P networks, many factors can influence the contribution value of a given servent – e.g., shared storage or shared bandwidth. It is difficult to rank the importance of these factors. Furthermore, it is the shared *contents*, rather than the resources being shared, that attract download requests. Hence, the same amount of bandwidth or storage offered for sharing by two servents does not imply the same amount of contribution by these servents. Rather, we define a servent's contribution as the utility it can provide to the whole P2P community. When a servent, say k, transfers files for other servents, it gives utilities to the community. Therefore, we increase the contribution value of k by the social gain $\mathcal{SG}_k(u_k)$, which is defined as:

$$\mathcal{SG}_k(u_k) = SW(u_k, \mathcal{R}_k) = \max \sum_{i\in\mathcal{R}_k} U_i(\theta_i, y_i). \qquad (12)$$

On the other hand, when a servent $i \in \mathcal{R}_k$ receives transfer bandwidth of x_i based on the CDPF algorithm, the x_i value may not be equal to y_i, which is the solution of the PF algorithm for maximizing the social welfare. Therefore, we define $\mathcal{SP}_i(x_i)$ as the *social payment* for servent i when it receives a transfer bandwidth of x_i :

$$\begin{aligned}\mathcal{SP}_i(x_i) \;=\; & SW(u_k, \mathcal{R}_k) - \\ & [U_i(\theta_i, x_i) + SW(u_k - x_i, \mathcal{R}_k - \{i\})] \qquad (13)\end{aligned}$$

The physical meaning of $\mathcal{SP}_i(x_i)$ is the difference between the maximum aggregate utility under social welfare resource distribution (as solved by the PF algorithm) and the maximum aggregate utility under contribution dependent resource distribution (as solved by the CDPF algorithm). In other words, if x_i bandwidth is assigned to servent i based on the CDPF algorithm, the P2P community will not receive maximum social welfare. Hence, servent i should pay for this difference, and we deduct the payment amount from servent i's contribution value.

B. Instantaneous contribution update

When a servent, say k, provides its transfer bandwidth for use by the P2P community, its contribution value is increased by $SW(u_k, \mathcal{R}_k)$. This increase is equal to the maximum social welfare. On the other hand, the aggregate utility received by the requesting servents in $\mathcal{R}_k$ is not equal to $SW(u_k, \mathcal{R}_k)$. The reason for the difference is that some requesting servents may receive more transfer bandwidth under the CDPF algorithm, as compared with the PF algorithm. Such a servent needs to make a social payment equal to the extra bandwidth received.

In our contribution update mechanism, we compare the resource allocation, x_i, under CDPF with the resource allocation, y_i, under PF for servent i. We choose a servent who obtains the largest amount of extra bandwidth $x_i - y_i$ and reduce that servent's contribution by $\mathcal{SP}_i(x_i)$. The process is repeated until

the solution of the CDPF algorithm equals the solution of the PF algorithm.

Let us illustrate the contribution update process using the example in Figure 3. For the resource owner servent k, C_k is increased by $\mathcal{SG}_k(u_k) = \log(2) + \log(1.75) + \log(1.75)$.

	i=1	i=2	i=3	i=4
(d_i,C_i)	(50,100)	(100,50)	(100,100)	(200,200)
y_i	50	75	75	0
x_i	50	0	50	100
$x_i - y_i$	0	-75	-25	100^*

In the first round shown in the table above, servent 4 gains the most extra resource. Therefore its contribution value is reduced by $\mathcal{SP}_4(100) = SW(200, \{1,2,3,4\}) - [U_4(\theta_4, 100) + SW(100, \{1,2,3\})] = [\log(2) + \log(1.75) + \log(1.75)] - [\log(1.5) + (\log(2) + \log(1.5)]$. There are 100 units of resource remaining for servent k.

	i=1	i=2	i=3
(d_i,C_i)	(50,100)	(100,50)	(100,100)
y_i	50	25	25
x_i	50	0	50
$x_i - y_i$	0	-25	25^*

In the second round shown in the table above, servent 3 gains the most extra resource. Therefore its contribution value is reduced by $\mathcal{SP}_3(50) = SW(100, \{1,2,3\}) - [U_3(\theta_3, x_3) + SW(50, \{1,2\})] = [(\log(2) + \log(1.5)] - [\log(1.5) + \log(2)]$. There are now 50 units of resource remaining for servent k.

	i=1	i=2
(d_i,C_i)	(50,100)	(100,50)
y_i	50	0
x_i	50	0
$x_i - y_i$	0	0

In the last round shown in the table above, CDPF and PF produce the same resource allocation vector. Hence, no more payment is necessary by servents 1 and 2. Lastly, we point out that our contribution update mechanism achieves the desirable property (1) given in Section II-B. This is formally stated in the following theorem.

Theorem *4: The cumulative contribution is conserved to be equal to the total social utility at all time. That is,*

$$\sum_{i=1}^{N} C_i(t) = \sum_{i=1}^{N} \int_0^t U_i(\theta_i, x_i(\tau))d\tau \quad \forall t > 0.$$

Proof: For simplicity, assume $C_i(0) = 0, \forall i \in \mathcal{R}_k$. Let $G_{k,i}(t)$ be the payment of servent i for receiving bandwidth x_i from servent k at time t and $G_k(t)$ be the gain of servent k for providing its bandwidth at time t. Let $\mathcal{R}_k^*(t) \subseteq \mathcal{R}_k(t)$ be the set consisting of all the servents requesting resources from servent k and having positive payments at time t. Consider the time interval $[t, t + \Delta t)$. We have

$$\sum_{i \in \mathcal{R}_k \cup \{k\}} [C_i(t + \Delta t) - C_i(t)] = [G_k(t) - \sum_{i \in \mathcal{R}_k^*} G_{k,i}(t)]\Delta t.$$

Let a_j be the j^{th} servent that is required to pay. According to the payment rules, we have

$$\begin{aligned} G_{k,a_j}(t) &= SW(u_k - \sum_{i=1}^{j-1} x_{a_i}(t), \mathcal{R}_k - \{\bigcup_{i=1}^{j-1} a_i\}) \\ &\quad -[U_{a_j}(\theta_{a_j}, x_{a_j}(t)) \\ &\quad +SW(u_k - \sum_{i=1}^{j} x_{a_i}(t), \mathcal{R}_k - \{\bigcup_{i=1}^{j} a_i\})]. \end{aligned}$$

Summing for all the servents in $\mathcal{R}_k^*$, we have

$$\begin{aligned} \sum_{i \in \mathcal{R}_k^*} G_{k,i}(t) &= SW(u_k, \mathcal{R}_k) - [\sum_{i \in \mathcal{R}_k^*} U_i(\theta_i, x_i(t)) \\ &\quad +SW(u_k - \sum_{i \in \mathcal{R}_k^*} x_i(t), \mathcal{R}_k - \mathcal{R}_k^*)]. \end{aligned}$$

Since $SW(u_k, \mathcal{R}_k) = G_k(t)$, we have

$$\begin{aligned} G_k(t) - \sum_{i \in \mathcal{R}_k^*} G_{k,i}(t) &= \sum_{i \in \mathcal{R}_k^*} U_i(\theta_i, x_i(t)) + \\ &\quad SW(u_k - \sum_{i \in \mathcal{R}_k^*} x_i(t), \mathcal{R}_k - \mathcal{R}_k^*)]. \end{aligned}$$

When the contribution update process finishes, a servent, say i, who does not need to pay receives a transfer bandwidth of x_i under CDPF, which is equal to the bandwidth y_i under the PF algorithm. Therefore, we have

$$SW(u_k - \sum_{i \in \mathcal{R}_k^*} x_i(t), \mathcal{R}_k - \mathcal{R}_k^*) = \sum_{i \in \mathcal{R}_k - \mathcal{R}_k^*} U_i(\theta_i, x_i(t))$$

. Thus,

$$\begin{aligned} G_k(t) - \sum_{i \in \mathcal{R}_k^*} G_{k,i}(t) &= \sum_{i \in \mathcal{R}_k} U_i(\theta_i, x_i(t)) \\ \sum_{i \in \mathcal{R}_k \cup \{k\}} [C_i(t + \Delta t) - C_i(t)] &= \sum_{i \in \mathcal{R}_k} U_i(\theta_i, x_i(t))\Delta t \\ \sum_{i \in \mathcal{R}_k \cup \{k\}} \frac{dC_i(t)}{dt} &= \sum_{i \in \mathcal{R}_k} U_i(\theta_i, x_i(t)) \\ \sum_{k=1}^{N} \sum_{i \in \mathcal{R}_k \cup \{k\}} \frac{dC_i(t)}{dt} &= \sum_{k=1}^{N} \sum_{i \in \mathcal{R}_k} U_i(\theta_i, x_i(t)) \\ \sum_{i=1}^{N} \frac{dC_i(t)}{dt} &= \sum_{i=1}^{N} U_i(\theta_i, x_i(t)) \\ \int_0^t \sum_{i=1}^{N} \frac{dC_i(\tau)}{d\tau} d\tau &= \int_0^t \sum_{i=1}^{N} U_i(\theta_i, x_i(\tau))d\tau \\ \sum_{i=1}^{N} C_i(t) &= \sum_{i=1}^{N} \int_0^t U_i(\theta_i, x_i(\tau))d\tau. \end{aligned}$$ ∎

Our contribution update mechanism is based on a fluid model. In implementation, we divide time into quanta denoted as Δt. At the beginning of each time quantum, we assign the transfer bandwidth using the CDPF algorithm. At the end of

each time quantum, we update the contribution values of servent k and the servents in $\mathcal{R}_k$. The pseudo-code for the contribution update is:

Contribution Update (r,u_k,all requesting servents in $\mathcal{R}_k$)

1. **x** = CDPF(r,u_k,all requesting servents in $\mathcal{R}_k$);
   `/* x is the solution of CDPF algorithm */`
2. **y** = PF(u_k,all requesting servents in $\mathcal{R}_k$);
   `/* y is the solution of PF algorithm */`
3. $C_k = C_k + (\sum_{i \in \mathcal{R}_k} \log(\frac{y_i}{d_i} + 1)) * \Delta t$;
   `/* the resource owner k increases its contribution by the social gain */`
4. **do** {
5. q = arg max $\{x_i - y_i\}$;
   `/* q is the servent who will reduce its contribution in this iteration */`
6. **if** ($x_q - y_q > 0$) {
7. $\mathcal{SP} = \sum_{i \in \mathcal{R}_k} \log(\frac{y_i}{d_i} + 1)$;
8. $\mathcal{R}_k = \mathcal{R}_k - \{q\}$;
9. $u_k = u_k - x_q$;
10. **y** = PF(u_k,all requesting servents in $\mathcal{R}_k$);
11. $\mathcal{SP} = \mathcal{SP} - (\log(\frac{x_i}{d_i} + 1) + \sum_{i \in \mathcal{R}_k} \log(\frac{y_i}{d_i} + 1))$;
    `/* servent` q `reduces its contribution by its social payment */`
12. $C_q = C_q - \mathcal{SP} * \Delta t$;
13. }
14. } **while** ($x_q - y_q > 0$);

In line 3, the algorithm increases the contribution of the resource owner by the social gain $\mathcal{SG}$. The contributions of the requesting servents are reduced in the do-while loop. In line 5, we choose the servent q who gains the most extra resource $x_i - y_i$. From lines 7 to 11, we compute the social payment for servent q and adjust the remaining amount of resource for the remaining requesting servents. In line 12, we decrease the contribution of servent q by its social payment.

Theorem *5: The contribution update algorithm has computational complexity $O(n^2 \log(n))$, where n is the number of requesting servents.*

Proof: From Theorem 3, we know that the CDPF algorithm has complexity $O(n \log(n))$. The PF algorithm is a special case of the CDPF algorithm and also has complexity $O(n \log(n))$. Each iteration within the do-while loop executes the PF algorithm (line 10) once. We execute the do-while loop at most n times in the case that $n - 1$ requesting servents need to make their social payment. Therefore, the do-while loop has complexity $O(n^2 \log(n))$. Since the statements outside the do-while loop have a lower complexity, the contribution update algorithm has time complexity $O(n^2 \log(n))$. ∎

V. Maintaining the Truth Revealing Property

In this section, we discuss how one can provide the truth revealing property. In particular, how to ensure that (1) no servent in the P2P system will misrepresent its contribution value, and (2) each servent will honestly report its true connection type.

Maintaining the True Contribution Values: First, we cannot simply allow a servent to alter its contribution at will. The justifications are

- A malicious servent k may increase its contribution value by *more* than its social gain $\mathcal{SG}_k(u_k)$ after it has provided a file transfer service.
- A malicious servent i may reduce its contribution value by *less* than its social payment $\mathcal{SP}_i(x_i)$ after it has received a file transfer service.

Through these malicious acts, a servent hopes to receive a higher transfer bandwidth when it requests for file download. We propose a light-weight, secure protocol to perform the contribution update process. We adopt the concept of *reputation system* [8]. In our framework, the contribution update process is managed by a *distributed auditing authority* (**DAA**). One can view a **DAA** as a database of servents' contributions distributed among a set of nodes, each of which is an *auditing authority* (**AA**). Servents in a P2P system can contact *any* **AA** in the distributed set for contribution update. The **AA** uses well-studied distributed concurrency control methods [7] to maintain the consistency and integrity of the distributed database. In this paper, we also assume that all the **AA**s are trusted.

Let us describe the contribution update process which involves the **AA**, the servent k that provides a file transfer service, and a set of requesting servents $\mathcal{R}_k$. For ease of presentation, let us consider one particular **AA**. The **AA** maintains a table which records the contribution value of all the servents in a P2P system. Each tuple in the table is of the format {`ID, C_val, timestamp, tran_ID`} where `ID` is a unique identifier of a servent in the incentive P2P system, and `C_val` is the latest contribution value recorded for the servent at time `timestamp` based on the latest transaction ID `tran_ID`. When a new user registers in the incentive network, an entry is created in the **AA**'s table and the servent is assigned an initial contribution value of zero.

Assume that servent i wishes to request a file from servent k. Servent i needs to first request its *contribution certificate*, denoted as $\mathcal{C}_i^*$, from the **AA**. **AA** performs the database lookup and returns $\mathcal{C}_i^*$ back to servent i. In our protocol, $\mathcal{C}_i^*$ contains the `C_val` of servent i, and the certificate is *digitally signed* by the **AA**. Servent i can then provide this certificate $\mathcal{C}_i^*$ and its maximum download bandwidth d_i to servent k for the file download request. The rationale for this step is that servent i *cannot* falsify its contribution value since it is certified by the **AA**. Any tampering of this value can be detected by servent k.

Servent k, upon receiving the file download request from servent i, can verify the validity of $\mathcal{C}_i^*$. If it is valid, k provides the file download service for servent i. At the end of the transfer quantum, servent i will provide a *service receipt* $\mathcal{R}_i^*$ (which is digitally signed by servent i) to servent k. After collecting service receipts from all requesting servents, servent k can submit these receipts to **AA** for the contribution update. The rationale for the service receipt is that servent k *cannot* falsify any contribution activity since **AA** can verify the validity of all service receipts. Based on the contribution update algorithm in Section IV, the **AA** may increase the contribution value of servent k and decrease the contribution value of all requesting clients in $\mathcal{R}_k$.

It is important to point out that after receiving a file transfer service, servent i may refuse to provide its service receipt $\mathcal{R}_i^*$ to servent k in the hope that its contribution value will not be

decreased. In this case, servent k can report this event to the **AA**. When **AA** detects a sufficient number of anomalies against servent i, **AA** can decide to *blacklist* [8] servent i for later activities. For example, the **AA** can refuse to provide the contribution certificate $\mathcal{C}_i^*$ to servent i. Lastly, because the initial value of contribution for a new servent is zero and the contribution value is always non-negative, a malicious servent cannot take advantage of using up its contribution value and then createing a new account in a P2P network to obtain file transfer service without any social contribution.

Truth Revealing of Connection Types: As mentioned, the incentive protocol assumes that all servents honestly report their connection type $\theta_i = (u_i, d_i)$, i.e., their maximum upload and download bandwidth capacities. We now address the truth revealing issue in connection type reporting. We argue that for a *rational* user, there is no incentive to report any connection type different from its true type.

Revealing the download bandwidth d_i: Consider a servent i with a maximum download bandwidth of d_i. There is no incentive for a rational user to report a download bandwidth which is higher than d_i. The reasons are:

- Based on the utility function in Equation (5), receiving a transfer bandwidth x_i higher than d_i will not result in a higher utility than $\log(2)$, which is achieved at having a transfer bandwidth of $x_i = d_i$.
- The proposed CDPF algorithm assigns the transfer bandwidth among requesting servents based on the *maxmin fairness* principle. Therefore, reporting a higher download bandwidth may result in receiving a lower transfer bandwidth allocation x_i.

Also, there is no incentive for a rational user to report a download bandwidth which is less than its true value d_i. The reason is that since servent i does not know about the congestion level at the servent, say k, providing service, declaring a download bandwidth less than d_i may result in a lower transfer bandwidth allocation x_i. This situation occurs when servent k is not congested and the CDPF algorithm assigns the transfer bandwidth x_i based on the declared download bandwidth. Although, one may argue that a requesting servent may probe the file-sharing servent multiple times in order to estimate the congestion level and bandwidth resources so as to obtain a higher bandwidth in the long run. Since the congestion level at the file sharing servent and at the network are time varying, this sort of scheme may not always achieve the result of obtaining a higher bandwidth for the requesting servent.

Revealing the maximum upload bandwidth u_k: Again, consider that servent k is a service providing node. Clearly there is no incentive for servent k to declare an upload bandwidth less than u_k. The reason is that this will discourage other servents from requesting service from servent k, which will in turn result in a *lower* contribution value for servent k.

On the other hand, servent k may declare that it has a higher upload bandwidth than u_k so to attract service requests from other servents. However, the wrong information can be easily detected since nodes requesting service will provide service receipts after the file download process. Therefore, the **AA** can easily verify whether the declared upload bandwidth is equal to the aggregate received bandwidth by all the requesting servents and servent k cannot increase its contribution value this way.

VI. Experimental Results

In here, we present simulation results showing that our mechanism can fairly distribute transfer bandwidth among the requesting servents and can provide higher aggregate utility than other scheduling disciplines like FCFS and processor-sharing.

Experiment A: servents with similar connection type but different contribution values: Four servents make requests to servent k, which has a transfer resource of $u_k = 400$. The contribution values of these requesting servents at time t_0 are $[C_1, C_2, C_3, C_4] = [1, 1.5, 2, 2.5]$. The connection types of all the requesting servents are the same and their maximal download bandwidth are $d_1 = d_2 = d_3 = d_4 = 150$. Each simulation lasts 100 units of time in the interval of $[t_0 \leq t \leq t_0 + 100]$. Figure 4 illustrates the bandwidth assignment $x_i(t)$ and their respective contribution values $C_i(t)$ during the simulation period.

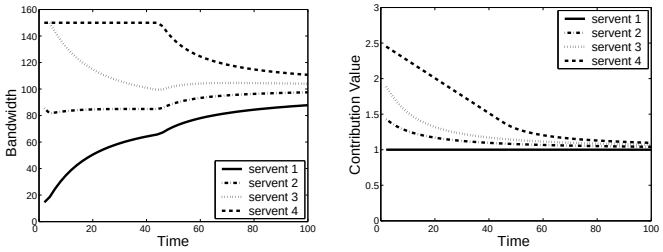

Fig. 4. (a) Instantaneous bandwidth assignment and (b) instantaneous contribution values for competing servents.

In experiment A, we observe that although all the servents have the same connection type, our CDPF algorithm assigns *higher bandwidth* to the servents which have larger contribution values. At the same time, our mechanism decreases the contribution values of these servents (since they are getting more social resources). Eventually, all servents tend to have equal contribution values and equal bandwidths, which maximize the aggregate utility among all the servents.

Experiment B: servents with different connection types and contribution values: The connection types of the requesting servents are different and they are $d_1 = 100, d_2 = 150, d_3 = 200, d_4 = 250$, respectively. All the other settings are the same as in Experiment A. For time $t_0 \leq t \leq t_0 + 100$, we illustrate the bandwidth assignment $x_i(t)$ and contribution $C_i(t)$ in Figure 5. In experiment B, we observe that our instantaneous

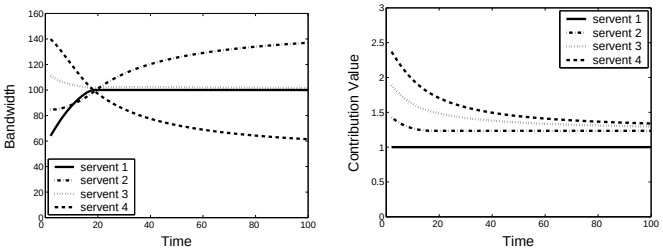

Fig. 5. a) Instantaneous bandwidth assignment and (b) instantaneous contribution values for competing servents.

bandwidth assignment also converges to the solution of the PF

algorithm, which provides maximized aggregate utility for the P2P network. Initially, servent 1 gains less bandwidth than the solution of the PF algorithm. Therefore, the CDPF algorithm does not decrease the contribution value of servent 1. For servent 2, 3, and 4, they gain a larger bandwidth than by the PF algorithm at the beginning. Afterwards, their contribution values converge to the same value. Therefore, their instantaneous bandwidth converges to the solution of the PF algorithm.

Experiment C: achieved utility under different resource distribution algorithms: We compare the efficiency of our incentive mechanism with that of the FCFS and processor-sharing disciplines. The average file transfer request rate matrix, Λ, is randomly generated in 10,000 experiments. There are fifty servents and they can make requests to each other. There are five different connection types and each servent has an equal probability of being any of the connection type. The file request rate and the file service rate are Poisson. Under the FCFS discipline, there are at most five servents receiving service at the same time. Any further requests are queued and served in FCFS order. Under the processor sharing discipline, each requesting servent gets an equal share of the available bandwidth from the provider servent. The distribution for the incentive mechanism is as described above. The probability density function for the aggregate utility under these three resource distribution algorithms are illustrated in Figure 6. The x-axis is the value of the aggregate utility and the y-axis is the frequency achieving the value of aggregate utility. The proposed incentive mechanism *always* gives a higher aggregate utility than the other algorithms.

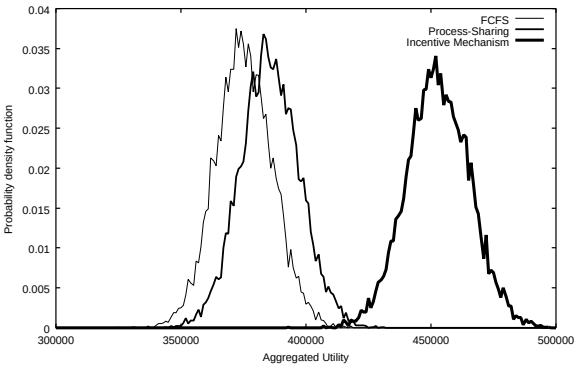

Fig. 6. Probability density function for aggregate utility under FCFS, process-sharing and incentive mechanism.

VII. Related Work

In [6], the authors address one possible mechanism for Napster-liked P2P network. Our work is different from theirs in the sense that our mechanism uses "virtual credit" so that it will not reduce the willingness of users to participate in a P2P network. The concept of *reputation system* is discussed in [8] for the application of P2P systems and ad-hoc networks. Zhong et al. [19] discuss some shortcomings of *micro-payment* and reputation system. They propose a cheat-proof, credit-based mechanism for mobile ad-hoc networks. In [5], the authors discuss the economic behavior of P2P storage networks. In [18], the authors model P2P networks as a Cournot Oligopoly game and give elegant control-theoretical solution focusing on global storage system. Our work focuses on the file-transfer and bandwidth allocation of a P2P system and we use the mechanism design approach in designing a competitive game in a P2P system. Lastly, algorithmic mechanism design [10], [11], [16] provides a theoretical framework for designing incentive mechanisms.

VIII. Conclusion

We have presented an incentive mechanism for P2P networks. Our mechanism distributes resources among servents based on each servent's utility function, connection type, and contribution. Our mechanism achieves both higher aggregate utility and fairness for a P2P network. Under our mechanism, the contribution value of a servent will be increased if it provides service to the P2P community. A servent who has a larger contribution value will receive a higher utility when it competes with other servents for file download services. Therefore, servents in the community have incentive to share information, thereby resolving the free-riding problem. Furthermore, our mechanism may decrease the contribution values of servents who access a congested resource. Therefore, it also provides incentive for servents to access information from non-congested servents and resolves the tragedy of the commons problem.

References

[1] The Gnutella Protocol Specifi cation v0.4 1, document revision 1.2.
[2] Limewire : a Gnutella client software.
[3] E. Adar and B. Huberman. Free riding on Gnutella. *Technical report, Xerox PARC, 10 Aug. 2000. FirstMonday*.
[4] J.-Y. Boudec. Rate adaptation, congestion control and fairness: A tutorial, `http://icapeople.epfl.ch/leboudec`.
[5] A. C. Fuqua, T. Ngan, and D. S. Wallach. Economic behavior of peer-to-peer storage networks. *To appear in Workshop on Economics of Peer-to-Peer Systems (Berkeley, California), June 2003*.
[6] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. *Proceedings of the 2001 ACM Conference on Electronic Commerce*.
[7] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann; 1st edition, 1993.
[8] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. *NOSSDAV*, 2003.
[9] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.
[10] N. Nisan and A. Ronen. Algorithmic mechanism design. *In Proc. 31st Annual Symposium on Theory of Computing (STOC99).*, 1999.
[11] D. Parkes. Chapter 2, Iterative Combinatorial Auctions: Achieving economic and computational effi ciency ph.d. dissertation, univesity of pennsylvania. May, 2001.
[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of ACM SIGCOMM*, 2001.
[13] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *Computer Science Dept., University of Chicago*, 2001.
[14] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
[15] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communication*, 13(7), September 1995.
[16] J. Shneidman and D. Parkes. Rationality and self-interest in peer to peer networks. *Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM*, 2001.
[18] W. Wang and B. Li. To play or to control: a game-based control-theoretic approach to peer-to-peer incentive engineering. *in the Proceedings of the Eleventh International Workshop on Quality of Service (IWQoS 2003)*.
[19] S. Zhong, Y. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks. *Technical Report, Department of Computer Science, Yale University, July 2002*.