# P2P Contracts: A Framework for Resource and Service Exchange ⋆

Dipak Ghosal [a,*], Benjamin K. Poon [b], Keith Kong [c]

[a] *Department of Computer Science, University of California, Davis, CA 95616*

[b] *Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94704*

[c] *Shazam Integrated Systems, 2635 Glen Ferguson Circle, San Jose, CA 95148*

**Abstract**

A crucial aspect of Peer-to-Peer (P2P) systems is that of providing incentives for users to contribute their resources to the system. Without such incentives, empirical data show that a majority of the participants act as free riders. As a result, a substantial amount of resource goes untapped, and, frequently, P2P systems devolve into client/server systems with attendant issues of performance under high load. We propose to address the free rider problem by introducing the notion of a P2P contract. In it, peers are made aware of the benefits they receive from the system as a function of their contributions. In this paper, we first describe a utility-based framework to determine the components of the contract and formulate the associated resource allocation problem. We consider the resource allocation problem for a flash crowd scenario and show how the contract mechanism implemented using a centralized server can be used to quickly create pseudoservers that can serve out the requests. We then study a decentralized implementation of the P2P contract scheme in which each node implements the contract based on local demand. We show that in such a system, other than contributing storage and bandwidth to serve out requests, it is also important that peer nodes function as application-level routers to connect pools of available pseudoservers. We study the performance of the distributed implementation with respect to the various parameters including the terms of the contract and the triggers to create pseudoservers and routers.

*Key words:* P2P Networks, Incentives, P2P Contracts, Resource Exchange, Flash Crowds, File Sharing

⋆ This research was funded in part by NSF grant ANI-9714668

* Corresponding author.
*Email addresses:* ghosal@cs.ucdavis.edu (Dipak Ghosal), bpoon@cal.berkeley.edu (Benjamin K. Poon), keith_kong@sbcglobal.net

## 1 Introduction

Peer-to-Peer (P2P) is the current incarnation of distributed computing [1], [2]. What sets it apart from previous forms of distributed computing is its sheer scale - literally millions of nodes across the world working together for some common purpose. But the benefits of scale come with high costs. The nodes are extremely heterogeneous, running on different operating systems, which themselves, run on top of different hardware with different kinds of CPUs, disks, and network connectivity. The nodes are also not very reliable; they go on and off several times a day. They cannot even be trusted on to perform computations correctly. The nodes may been compromised by malicious hacker intent on disrupting the system.

The importance of such projects as SETI@Home[3], Napster [4], Gnutella [5], and now KaZaA[41] is demonstrating that the benefits of scale exceed the costs that go with them. By cleverly focusing on applications that attract participation and designing for the behavior of peers, these projects showed that P2P can be harnessed successfully. Their successes have inspired number of similar projects, including FreeNet[6] and FreeHave [7]. The challenge that lies ahead is demonstrating that P2P works outside of niche applications.

A number of efforts are underway to meet this challenge. They fall under one of two broad categories. One places more emphasis on providing a generalized computation utility. These include Globus [8], grid computing [9] [10], and Condor [11]. The other places more emphasis on providing a generalized storage utility [12]. These include FarSite [13], OceanStore [14] [15], PAST [16] [17] [18] and many others [19] [20] [21] [22] [23]. However, these next-generation, industrial-strength P2P systems are hard to build. It takes a lot of effort to write robust code that handles failures gracefully. It takes even more effort to write code that handles an adverse environment comprising potentially millions of nodes where failure is the norm. There is a lot of effort wasted simply by reinventing code every time a new P2P application is developed.

While current proposals for a P2P middleware provide many services useful to application programmers, they do not capture a crucial aspect of P2P systems: that of providing incentives for users to contribute their resources to the system. Without such incentives, empirical data show a majority of the participants act as free riders who do not contribute any resource. As a result, a substantial amount of resource goes untapped, and, frequently, P2P systems devolve into client/server system with attendant issues of performance under high load. We propose to address the free rider problem by introducing the notion of a P2P contract. In it, peers are made aware of the benefits they receive from the system as a function of their contributions. Such a system

(Keith Kong).

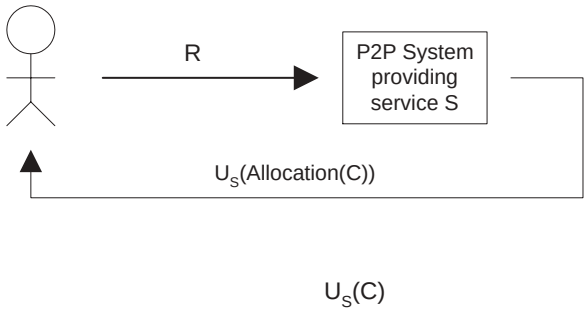


Fig. 1. The basic model of a P2P system.

rewards users who contribute resources. At the same time, it allows free riders so long as they do not unduly impact the quality of service for others.

In this paper, we first qualitatively illustrate the impact of free riders on a P2P system. We then review various incentive schemes that have been proposed in the literature and then outline the basic aspect of P2P contracts. We then present a formal utility-based framework to determine the terms of the P2P contract. We then address the implementation issues and study a centralized and a distributed architecture to the implement the system and compare their performance in the context of a flash crowd scenario.

## 2 Impact of Free Riders on P2P Systems

Without explicit incentives, users are lax to contribute any resource to a P2P system. The study on Gnutella reported in [24] provides compelling evidence for this. Through analysis of log files, the study concluded that 70% of Gnutella participants are free riders that consume resources without contributing any in return. As a result, all participants suffer from a greatly diminished quality of service. This section introduces a framework by which one can quantify the impact of this problem.

To illustrate the impact of free riders we consider a very simplified P2P system with some initial capacity $C$. A peer can add to this capacity by contributing resources to the system. The peer can also consume some allocation of capacity by obtaining service from it. By doing so, the peer derives some amount of utility $U_S$. Figure 1 illustrates this basic model. To further clarify the issues that motivate this framework, consider an example of a P2P system with an initial capacity of 16 resource units (RU). The utility of consuming some amount of system capacity is specified by the graph in Figure 2a. Note that no utility is obtained by consuming less than 1 RU. Likewise, no additional utility is obtained by consuming more than 4 RU of capacity. The shape of this curve reflects the well-known economic principle of diminishing marginal utility in

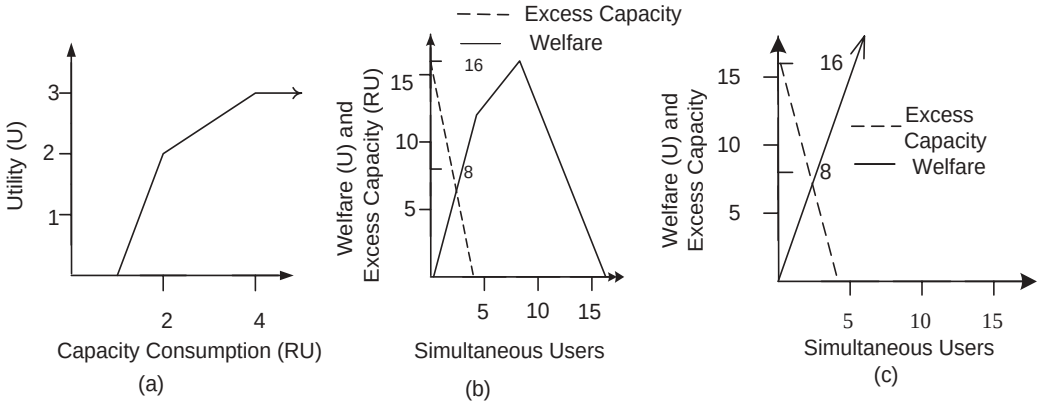


Fig. 2. Quantifying the impact of free riders.

consuming quantities of a good. Beyond a certain point, people simply derive less utility in consuming additional quantities of a good. In this case, the good is the service provided by a P2P system.

As peers obtain service, excess system capacity is consumed and the welfare of the system increases. These points are illustrated in Figure 2b. At four peers, the excess capacity has decreased to zero. Beyond this point, service degrades for existing peers as new ones obtain service from the system. Note that the welfare continues to increase; the additional utility derived by each newcomer more than offsets the service degradation caused by it. However, this no longer holds true beyond eight peers. The welfare decreases until it reaches zero at 16 peers, at which point the system no longer provides utility to any of its participants.

While free riders contribute to the welfare of a P2P system, at some point, they do more harm than good. This is a direct result of the utility function of the peers. In particular, each peer derives no utility by consuming less than some fixed amount of system capacity (in the example, 1 RU). By continuing to admit free riders, the system eventually reduces the utility it supplies to its participants to zero.

In traditional systems, this problem is addressed by admission control [25]. This mechanism works by preventing users from being admitted to the system when a complete collapse of the system becomes imminent. In doing so, admission control retains the utilities of users who have already been admitted. However, P2P systems offer the possibility that system capacity can be expanded as demand for it rises. With the right mechanisms in place, a P2P system is able to eliminate the negative impact of free riding while retaining the utilities free riders derive from using the system. This results in an increasing welfare curve, as shown in Figure 2c. The following sections discuss such mechanisms.

## 3 P2P Contract

A free rider should be admitted to the system so long as the benefit she derives from the service exceeds the cost her use imposes on other users. Calculating these quantities requires knowledge of utility functions for individual users as a function of the quality of service. These quantities, in turn, depend on how resources are allocated toward service provisioning. A model of these relationships would thus enable a system to determine when to apply admission control.

More importantly, a model of the relationship among resources and their allocation, quality of service, and utilities would enable the formulation of the P2P contract. This contract codifies the terms of an exchange between the requesting peer and the system. Specifically, the peer contributes some amount of resources, R, to the system. In return, the system provides some level of service, S, to the requester. Because S is a function of R, arriving at the terms of the contract reduces to the computation of R that satisfies both of the following constraints: 1) The welfare of the system increases and 2) the welfare of the peer increases. In general, multiple R's satisfy these constraints. We call this set of resource contributions the critical resource set (CRS). If the CRS is empty, then the user will not participate in the system. If the CRS has more than one element, then a further constraint needs to be considered to construct the P2P contract. One such constraint, for example, is maximization of the peer's welfare.

### *3.1 Other Schemes*

There is significant interest various incentive schemes for P2P systems [37] [38] [39]. There are currently two basic approaches for addressing the free rider problem. The first relies on altruism. The second uses some form of micropayment to provide economic incentive for resource contribution. The first approach relies on people's altruism for resource contributions. Users donate resources at their discretion, with no negative consequences for not doing so. Systems that take this approach hope that voluntary contributions are sufficient to provide good service to its participants. Napster [4], Gnutella [5] and FreeNet [6], among others [26] [27], showed that such systems do work in practice. Unfortunately, they operate suboptimally from the standpoint of maximizing the welfare of its participants. Service provisioning is done without regard to the utilities of users. Thus, free riders who may not derive much utility from the service are not discouraged from consuming resources, at the expense of the quality of service to users who value the service highly. Given that empirical evidence shows that as much as 70% of the peer are free riders

[24] [28], this is a pressing issue [29].

The second approach uses micropayments to provide economic incentives for resource contributions. Systems that use this approach belong to one of two categories depending on the type of currency used. A systems of the first type, such as Mojo Nation [30], uses an internal currency that cannot be redeemed for goods provided outside of the system. Participants of such a system earn money every time they contribute resources to the system. They spend money every time they consume resources from the system by obtaining service from it. Because users cannot obtain service without having contributed resources, free riders do not exist. A system based on an external currency, such as Popular Power [31], differs in that the currency can be redeemed for goods outside of the system. Thus, participants in such a system have incentive to provide resources even if they have no need to obtain services. Free riders do not exist in such systems either.

Golle [32] [33] used game theory to analyze the equilibrium behavior of the systems described above. For systems that rely on altruism, the steady-state behavior is highly asymmetric. Participants obtain service from the system using the resources of altruistic users. For systems that rely on internal currency, users contribute resources only if they need to obtain services from the system. For systems that rely on external currency, a much larger base of users contribute resources.

## 4 A Formal Utility-Based Framework

To compute the CRS, we consider a P2P system that offers a service $\underline{S} = [s_1, s_2, \ldots, s_M]$ where $s_i \geq 0, i = 1...M$ are the attributes of the service. For example, the P2P system offers keyword searches of files that can be characterized by two attributes - search time and the accuracy of the search. We assume that there are $K$ peers currently receiving service and consider the event when a new peer (peer $K+1$) makes a request for service.

Let $\underline{R}^{p2p} = [R_1, R_2, \ldots, R_N]$ denote the total resource currently available in the P2P system where $R_i > 0$ are the individual resource components. For example, $R_1$ could be the available storage capacity and $R_2$ could be the available bandwidth. These resources are an aggregate of resource contributions of users currently accessing service from the P2P system. We denote $\Delta\underline{R}^i$ to be the resource contributions from user $i$. Thus, $\underline{R}^{p2p} = \sum_{k=1}^{K} \Delta\underline{R}^k$ . Furthermore, let $\underline{R}^{K+1}$ denote the total resources of the new peer.

A service consumes resource $\underline{r} = [r_1, r_2, \ldots, r_N]$, where for $r_i \geq 0$ for $i = 1 \ldots N$. For each request for service, one can associate an $M \times N$ allocation

matrix, $A_{\underline{r}}$ with each $\underline{r}$ such that

$$A_{\underline{r}} = \begin{bmatrix} \underline{a_1} \\ \vdots \\ \underline{a_M} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix} \tag{1}$$

where $a_{ij} \geq 0$ for $i = 1 \dots M, j = 1 \dots N$, $\sum_{j=1}^{M} a_i j \leq r_i$, and $\sum_{k=1}^{K} \sum_{j=1}^{M} a_{i,j} \leq R_i$, $i = 1 \dots N$. The consumption by some service of a resource allocation can then be succinctly represented by

$$\underline{S} = S(A_{\underline{r}}) = \begin{bmatrix} s_1(\underline{a_1}) \\ s_2(\underline{a_2}) \\ \vdots \\ s_M(\underline{a_M}) \end{bmatrix} \tag{2}$$

The benefit of obtaining some service is represented by a utility function of that service. Again, a service is characterized by its attributes. Hence, the utility derived by the peer $k$ of using service is $\underline{S}$ is $U_{\underline{S}}^{k} = U_{\underline{S}}^{k}(s_1, s_2, \dots, s_M)$. We define the utility of consuming $\underline{r}$ in the context of some service $\underline{S}$ to be the allocation of $\underline{r}$ that maximizes the utility. In other words,

$$U_{\underline{S}}^{k}(\underline{r}) = \max_{A_{\underline{r}}} \{U^k(\underline{S}(A_{\underline{r}}))\} \tag{3}$$

We let $U_{\underline{o}}^{i}(\underline{R}^i)$ denote the utility that peer $i$ derives from all other services that it currently receives. The CRS for user $K+1$ is obtained by solving the following optimization problem: find all possible choices of $\Delta\underline{R}^{K+1}$ , such that both the following two constraints are satisfied:

(1) Peer Constraint: $U_{\underline{o}}^{K+1}(\underline{R}^{K+1} - \Delta\underline{R}^{K+1}) + U_{\underline{S}}^{K+1}(\underline{r}) \geq U_{\underline{o}}^{K+1}(\underline{R}^{K+1})$
(2) System Constraint: $\sum_{k=1}^{K+1} U_{\underline{S}}^{k}(\underline{r})$ is non-decreasing

If CRS is empty, deny the request. Otherwise, use additional constraints to determine the appropriate $\Delta\underline{R}^{K+1}$ for the P2P contract.

Finding the allocation matrix subject to various constraints is crucial to the decision-making processes for both the system and the peer. However, finding $U_{\underline{S}}^{i}(r)$ and its associated $A_{\underline{r}}$ is nontrivial. Nevertheless, under the following simplifications, dynamic programming can be used to find the solution:

(1) There is only one "compute resource" to be exchanged and allocated as opposed to multiple kinds such as CPU, storage, and network bandwidth, i.e., $\underline{r} = r$.
(2) The utility function is restricted in its form to a sum of the utilities of the service attributes, i.e., $U^i_{\underline{S}} = \sum_{j=1}^{M} U^i_j(s_j)$ and ,
(3) The utility functions are the same for all users, i.e., $U^i_{\underline{S}} = U_{\underline{S}}$ for all $i$.

With these simplifications, Equation (3) reduces to

$$U^i_{\underline{S}}(r) = \max_{\underline{a}_r} \{ \sum_{j=1}^{M} \Omega^i_j(a_j) \} \tag{4}$$

where $\Omega^i_j = U^i_j \times s_j$ subject to the constraint $\sum_{i=1}^{M} a_i \leq r$. This is precisely the form amenable to dynamic programming, which has been found to be computationally tractable in many applications [34].

## 5 Key Issues

There are three main issues in a system that employs P2P contracts: 1) accurately quantifying the utility function of peers for the services derived from the P2P systems and the dis-utility of contributing resources, 2) enforcement of contracts, and 3) determining where (and/or who) should determine the contract when a new peer joins the P2P system. Determining the utility function of a peer is a very difficult problem even for a simple P2P system such as file sharing. For example, one peer may benefit more from a specific file than another. In this paper, we do not address this problem in its generality. Rather, we consider a special case in which the utility function can be fairly accurately quantified. In particular, we consider a flash-crowd scenario in which there is a large demand for a single file. In such a scenario, utility of a peer participating in the flash-crowd can be approximated by a simple binary function: if the peer gets the file, it results in a utility of 1; if the peer does not get the file, it results in a utility of 0. Furthermore, we assume that the peer does not attribute any dis-utility to the resources that it contributes to the P2P system; i.e., it is willing to contribute the required resources to obtain a copy of the file.

The second important aspect is the issues of contract enforcement. While we do not address this issue in this paper, we make the following two comments in passing. First, there is a body of work dealing trusted third party server to negotiate and enforce electronic transactions. Some of the key ideas developed in those studies can be applied to enforce P2P contracts. Second, the resource allocation problem can take into account that there will be some

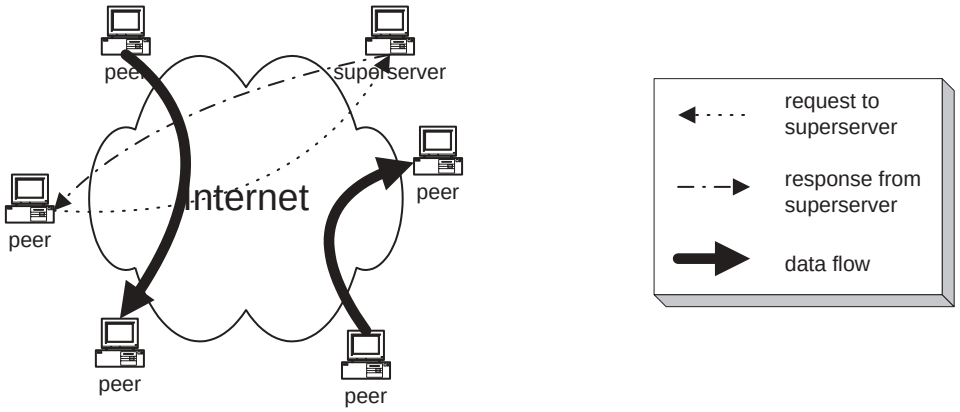


Fig. 3. Architecture of the pseudoserving system.

contract breaches. The impact of these breaches on the quality of service can be mitigated by setting the P2P contract in such as way that there is some amount excess resource in the system. While this does not eliminate the need to enforce contracts, it minimize the immediate impact of contract breaches on the quality of service.

In this paper, we address the issue of determining who should create the contracts, what should be the terms of the contract, which peers should be given contracts (and which peers could get a free ride), what are the triggers to initiate contracts, and how contracts should be communicated and agreed upon. The next two subsections discusses these issues for a centralized and a distributed implementation of a P2P file sharing system.

### *5.1 Centralized Architecture*

A centralized implementation of a system based on P2P contracts is the pseudoserving system described in [35], [36] and shown in Figure 3. It consists of two components: a superserver and a set of peers. Under "low-demand" conditions, the superserver functions as a traditional Web server and the peers function as traditional Web clients. Under "high-demand" conditions, the superserver draws a contract. In it, access is granted only to peers that agree to serve files they retrieve to $N$ other peers within $T$ seconds [1] . The access itself comes as a referral to where the requested file may be obtained. Peers who agree to serve the file become pseudoservers [35], [36].

Two aspects of the contract are worth clarifying. First, a peer begins to fulfill the $T$ aspect of the contract after it has received the file. Second, a peer is released from its obligation should it have held onto the file for $T$ seconds but

[1] Peer that do not agree to the contract are not admitted; admission control is thus implied.

less than $N$ other peers were directed to it. In a typical transaction, a peer first uses the referral to obtain the file. It then serves the file to other peers up to $N$ times. For periods where there is an increase in demand for the file, the peer typically serves other peer $N$ times, where $N \geq 1$. But for periods where there is a decrease in demand, the peer may not be asked to provide service at all.

In the centralized architecture, since the superserver receives all request, it has global knowledge of the demand. If the demand is greater than some prespecified threshold $S_{thresh}$, it starts handing out contracts to new requesters. If the demand is less than $S_{thresh}$ [2], the superserver does not set any contract and can either itself serve out the request or refer to a pseudoserver if one exists. It is also assumed that the superserver has global information about the network so that it can "optimally" refer a requester to its nearest pseudoserver.

### *5.2 Decentralized Architecture*

As opposed to the centralized scheme discussed above, the decentralized implementation has no central controller. It relies instead, on the ad hoc creation of routers and pseudoservers. Note that this implementation must be initialized with a set of pseudoservers that initially have the file. Because only pseudoservers have the target file, active peers must find pseudoservers to get a copy of the file. They do so by flooding queries to their neighbors since there is no central entity that knows the location of the current set of pseudoservers. If one of the neighbors is a router, the query is forwarded to the router's neighbors. When a pseudoserver receives a query, the pseudoserver contacts the querying peer directly and offers one of three choices: 1) offer the file to the active peer for free, 2) offer a routing contract to the active peer specifying that the peer will receive the file in exchange for an agreement to route $Q$ queries for $G_r$ generations [3], whichever comes first, or 3) offer a pseudoserving contract to the active peer specifying that the peer will receive the file in exchange for an agreement to serve $S$ peers for $G_p$ generations, whichever occurs first.

The pseudoserver decides which of the three actions to take based on the observed demand which is the number of active peers that request the file in the current generation. It maintains two thresholds, $P_{thresh}$ and $R_{thresh}$. If the demand is greater than $P_{thresh}$, the pseudoserver offers a pseudoserving contract which makes the requester a pseudoservers. If the demand is less than $R_{thresh}$, there is not much demand and pseudoserver gives the file away

[2] In general, the two thresholds may be different to induce some hystersis in the system.

[3] As explained later, our system evolves in discrete time-steps we call generations.

for free. When the demand is between $P_{thresh}$ and $R_{thresh}$, there is a medium level of demand and pseudoserver offers a contract to make the requester into a router. This allows for other peers to locate pseudoservers.

If an active peer does not receive any response for a request, the request is considered to have failed. On the other hand, the peer may recieve one or more responses. In that case, the peer acts in a purely selfish mode and accepts the least constraining response, i.e., accept the offer that requires the least amount of resource contributions. Note that the router contract $(Q, G_r)$ requires less resources than the pseudoserving contract $(S, G_p)$.

### *5.3 Central-index Without Contracts*

In addition to centralized and decentralized implementations, we also simulated a traditional client-server systems, in which only one peer (the server) has the target file and no contracts are made. This scheme is also referred to as the central-index without contracts. In this implementation, all active peers know the location of the server and they contact it directly to receive the file. Additionally, the server can only serve a predefined number of files per generation which is the server capacity.

## 6 Simulation Model

The simulator developed for this study was able to simulate both the centralized and the decentralized implementations. It was inspired in part by John Conway's Game of Life [40], in which a board is divided into a number of equal sized cells, where each cell represents a potential peer. Simulations have been carried out for the case in which there is a single target file which is requested by a large number of peers. It is straightforward to extend the simulation to multiple files. However, in this paper, we consider a flash-crowd scenario on a single file.

During the simulation, the behavior of the peer follows a specific pattern which is represented in a state diagram shown in Figure 4. All the peers start in the non-participatory state. The meaning of each state in the state diagram is described below:

- Non-participatory: the node does not want to obtain the target file;
- Inactive: the node wants the target file, but is not actively sending queries to search for it;

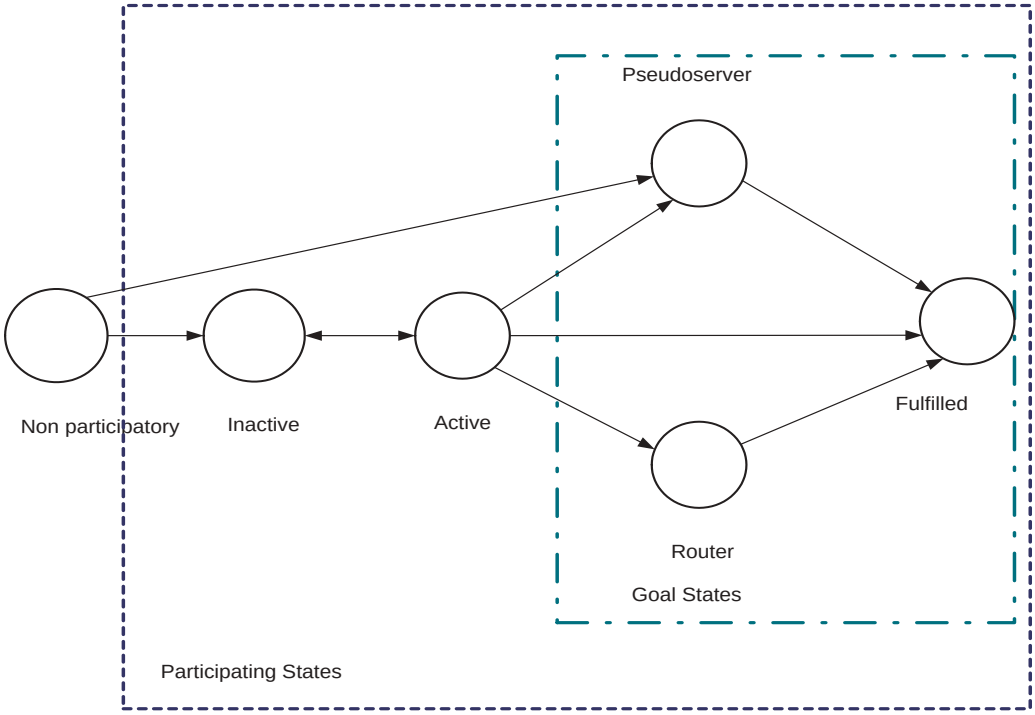


Fig. 4. Peer state diagram. The dashed boxes surrounding states represent a similarity among states and are shown merely to aid in discussion. Goal states = {fulfilled, router, pseudoserver} because all three of these states represent having the target file; participating states = {inactive, active, goal states} because all of these states represent participation in the simulation.

- Active: the node wants the target file and is actively sending queries to search for it;
- Router: in this mode the node receiving a query will forward the query to all its neighbors (note that this state exists exits only in decentralized implementation);
- Pseudoserver: in this state a node receiving a query will serves the target file to the requesting peer and in exchange may offer a contract which maybe a pseudoserving contract in the case of the centralized implementation or a pseudoserving contract or a router contract in the case of the decentralized implementation; and
- Fulfilled: in this state the node has received the target file and has fulfilled the terms of its contract; once in this state, the node does not play an active role for the remainder of the simulation.

To facilitate the following discussion, a peer in the inactive state will be referred to simply as inactive. This applies for peers in active, router, pseudoserver, and fulfilled states as well.

### *6.1 Initialization*

Before starting a simulation, the board is first initialized. This involves choosing two sets of random peers. The first set are the participants of the system. They are distinguished from non-participants by their initial state assignment. In particular, participants start with the "inactive" state, and non-participants start with the "non-participatory" state. The second set are participants in the system that have the file and play the role of either the pseudoserver or the superserver. As such, they are necessarily a subset of the first set. They are distinguished from other members of the first set by having an initial state assignment of "pseudoserver" instead of "inactive."

Initialization also involves defining each peer's neighborhood. Currently, we have created two methods of modeling neighborhoods. The first method defines a peer's neighbors as consisting of the cells in every direction including diagonals (a "Moore" neighborhood). This implies that 1) peers at the corners of the board have three neighbors, 2) peers on the edges of the board have five neighbors, and 3) peers in the center of the board have eight neighbors. The second method defines a peer's neighbors randomly, giving it a random number of neighbors in random positions on the board. The distribution of the number of neighbors can be changed to more closely model real-world networks.

### *6.2 Simulations*

Time evolves in discrete time steps called generations. A simulation is a set of generations. Each generation consists of three distinct phases: 1) creation of activity, 2) query dispatch and setting contract, and 3) calculation of each peer's next state given contracts made in the second phase. The first phase involves randomly changing inactive peers into active peers and vice versa. This is represented by the inactive to active transition in the state diagram. The second phase is characterized by all active peers sending queries to their neighbors (decentralized implementations) or the superserver (centralized implementations). Once any pseudoserver is contacted by such a query, the active peer and pseudoserver will undergo the process described in the previous step. The third phase simply changes the state of all the peers to the new state. A simulation ends when one of the following two end-conditions is met: 1) there are no more pseudoservers (for the decentralized implementation), or 2) there are no more inactive or active peers (for both implementations). The first ends a simulation because there are no more peers serving the file and the second because there are no more peers looking for the file.

### *6.3 Parameters*

The key parameters of the simulator are the simulation mode (central index without contracts, centralized, or decentralized), the board size, percentage of peers that are inactive, percentage of peers that are initialized as pseudoservers, the neighborhood type, the probability for an inactive peer to become active, the router contract parameters $(Q, G_r)$, the pseudoserver contract parameters $(S, G_p)$, the thresholds ($P_{thresh}$ and $R_{thresh}$ or $S_{thresh}$), and the server capacity. Among these variables, the router contract, the router threshold, the pseudoserver contract, and the pseudoserver threshold have been been investigated with respect to their impact on the performance of the system.

## 7 Results and Discussion

To compare the two implementations of P2P contract-based file sharing systems, simulations were conducted on each and the results are discussed in this section. The two architectures were compared using the following metrics: 1) the number of active peers (demand), 2) the number of failed requests, 3) the average percent failed requests, and 4) the number of peers unfulfilled. Simulation were conducted on a board of size 25x25 with a Moore neighborhood, and at each generation a cell that has not yet obtained the file had a 66% probability of making a transition from inactive to active state.

### *7.1 Central-Index Without Contracts*

For this implementation, the important parameter was the server capacity, which was set to 5. As seen in Figure 5, the demand falls off linearly with slope equal to the server capacity. The number of failed requests also falls linearly, but with some variation due to the randomness of when inactive peers become active. This linear decay occurs because, for every fulfilled peer, there is one less inactive or active peer, which reduces the population of possible requesters for the subsequent generations.

### *7.2 Centralized Implementation*

While the only important parameter of the client-server implementation was the server capacity, the centralized implementation introduces additional parameters: the pseudoserver contract values $(S, G_p)$ and the threshold value

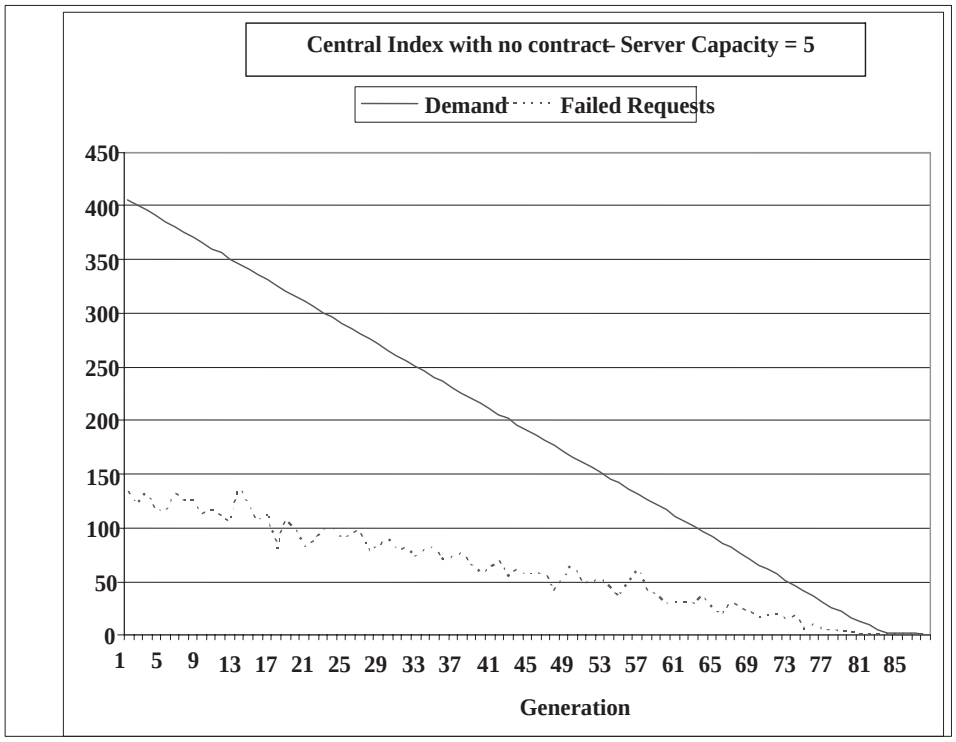

Fig. 5. Simulaton results of a client-server system.

$S_{thresh}$. In order to compare different pseudoserver contracts, we considered three different pairs of $(S, G_p)$ values: 1) $S = 2, G_p = 2$, 2) $S = 3, G_p = 5$, and 3) $S = 5, G_p = 7$. For each of these simulations, we set the superserver capacity to 4 and $S_{thresh} = 3$. The superserver capacity was made one less than in the client-server mode to account for the routing overhead incurred by the superserver.

For $S = 2$ and $G_p = 2$ (represented by the darkest lines with the largest value markers in Figure 6), the results show an almost exponential decrease in demand for the first six generations. The failed requests show similar behaviour. This can be attributed to the almost exponential increase in pseudoservers over time. There are, in fact, so many pseudoservers that, by the fifth generation, there are no more failed requests through the end of the simulation. This is a significant improvement over the client-server model. This result shows that even a modest contract can dramatically increase the performance of a P2P file sharing system.

Interestingly enough, the second simulation with $S = 3$ and $G_p = 5$ (represented by the grey lines with the medium-sized value markers in Figure 6) showed that an increase in pseudoserver contract parameters did not improve the performance significantly. The curves for the parameters $S = 2$ and $G_p = 2$ are merely shifted left by one generation implying that the demand is only fulfilled one generation sooner. The third simulation (represented by the lightest

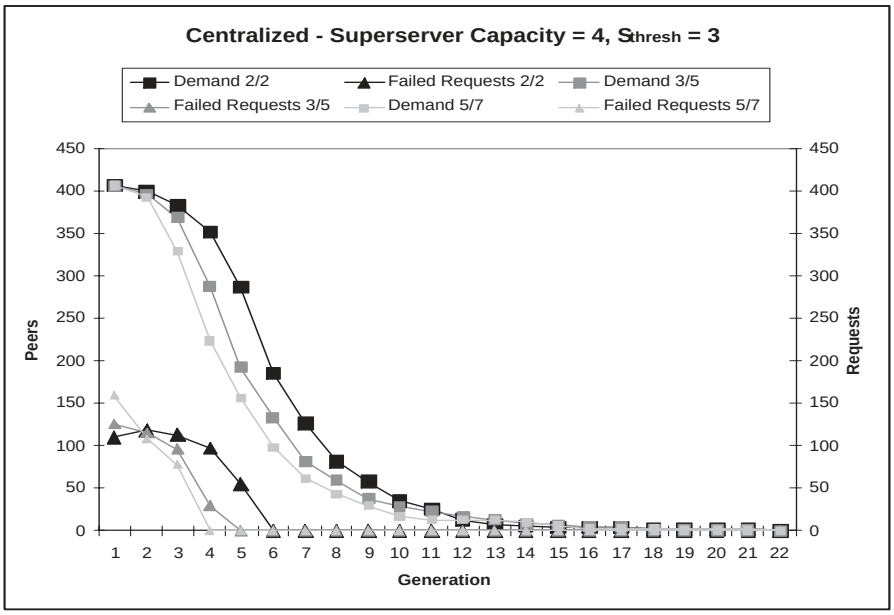


Fig. 6. Simulation results of a centralized implementation with different pseudoserver contract values.

lines with the smallest value markers in Figure 6) further illustrates that for the centralized implementation, increasing the pseudoserver contract values only results in only marginal improvement.

An area that we are currently exploring is the use of dynamic contracts, where the values $S$ and $G_p$ are changed dynamically depending on the demand experienced by the superserver.

### *7.3 Decentralized Implementation*

In the centralized implementation, the superserver has global knowledge of all existing pseudoserver, the status of their contracts and their locations, and the global demand at any given time. As a result, its performance serves as a reference (an upper bound) for the performance of the decentralized implementation. For the decentralized implementation, the important parameters include the router contract values $(Q, G_r)$, the pseudoserver contract values $(Q, G_p)$, and the two thresholds $R_{thresh}$ and $P_{thresh}$. For the same reasons as in the centralized implementation, the server capacity was set to 4 for all simulations.

The first set of experiments were performed to test the validity of the following

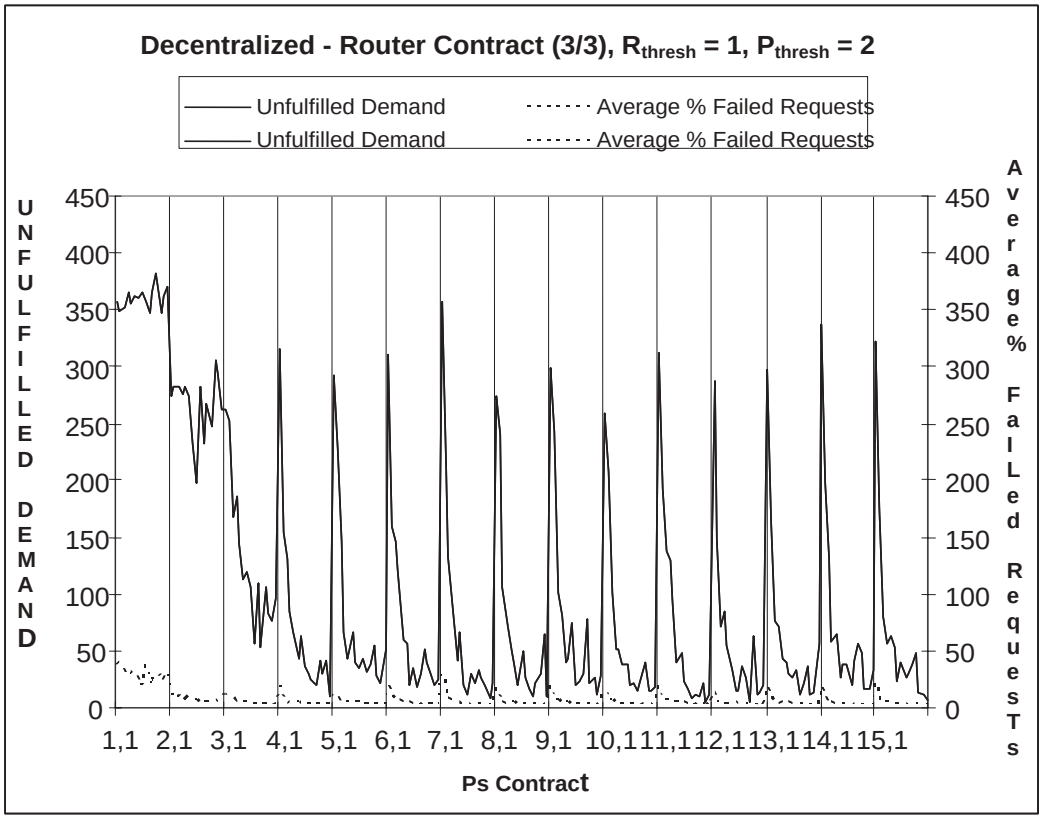


Fig. 7. Simulation results of a decentralized implementation with different pseudoserver contract values.

intuition: routers help performance more when pseudoservers have higher $G_p$ values. This makes sense intuitively because routers need to have pseudoservers that stick around long enough for them to send queries to. We tested this hypothesis by varying the pseudoserver contract values over different router contract values. For each pseudoserver and router contract pair, we recorded 1) the unfulfilled demand left at the end of each simulation and 2) the average percentage failed requests over the entire simulation. Figure 7 shows the results with router contract values $Q = 3$ and $G_r = 3$. One can see that, as $G_p$ is increased, both the average percent failed requests and the number of unfulfilled demand decreased substantially. (The jitter in the graph is due to the randomization of the model). This trend also occurred for all other router contract values, which supports the hypothesis. For our particular parameters, maximizing performance for minimal contract constraints results in an optimal $G_p$ value of 7. As for the value of $S$ in pseudoserver contract, it is interesting to note that when it is greater than 4 there is less unfulfilled demand. In Figure 7, with $S < 5$, there is an average of 200 unfulfilled peers at the end of simulations, whereas with $S \geq 5$, there is an average of 75 unfulfilled peers. Thus, there is an optimal value of $S$ that maximizes performance.

It is interesting to note that similar simulations with router contracts having

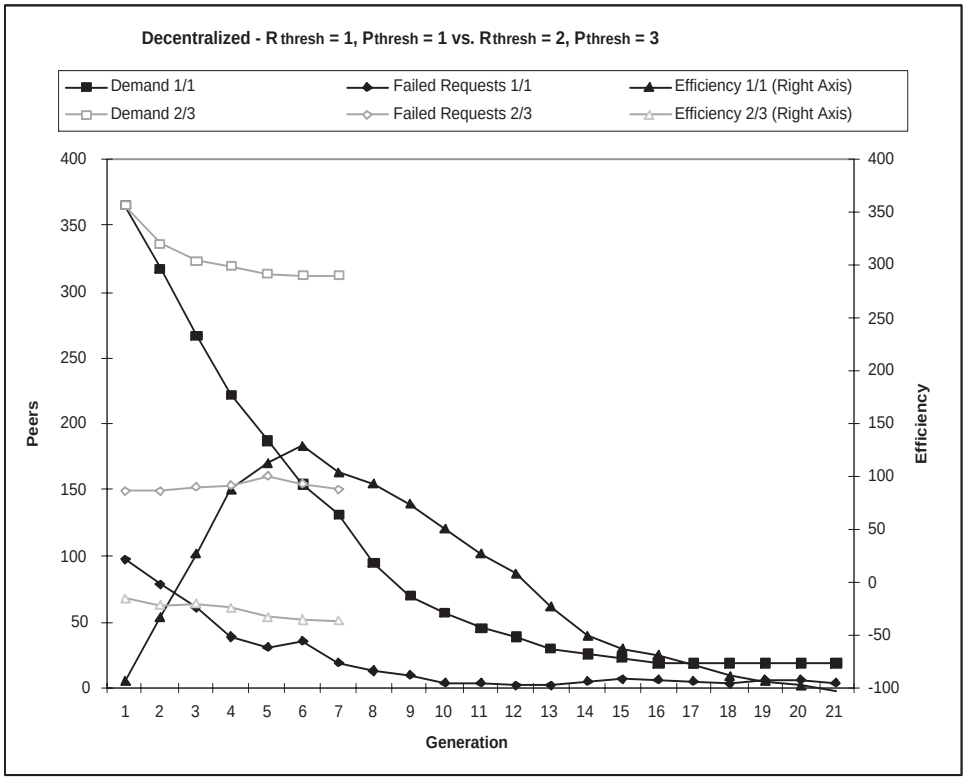


Fig. 8. Results of a decentralized implementation simulation comparing different thresholds.

$Q > 3$ and $G_r > 3$ showed only marginal improvement over the one shown in Figure 7. This is, in fact, similar to the result of increasing the pseudoserver contract values in the centralized implementation: increasing the routing contract values only results in marginal gains. For router contracts having $Q < 3$ and $G_r < 3$, while there was a similar values for the average percentage of failed requests, there was a marked increase in the number of unfulfilled demand at the end of simulations. This makes sense intuitively because router contracts are fulfilled too quickly to do enough work to fulfill requesters. Therefore, $Q$, like $S$, has an optimal value that maximizes the performance.

The results of the experiment with the $R_{thresh}$ and $P_{thresh}$ values are shown in Figure 8. To compare the impact of different thresholds, we ran a simulation with $S = 2$, $G_p = G_r = 2$, and two pairs of combinations of $R_{thresh}$ and $P_{thresh}$ denoted as 1/1 and 2/3 in the figure. Note that for $R_{thresh} = 1$ and $P_{thresh} = 1$ which implies that only pseudoservers are created and they are created any time an active peer contacts a pseudoserver, the demand and failed requests decrease over time, which is significant improvement over the system with central-index without contracts. Furthermore, the simulation reaches an end-condition (as a result of a lack of pseudoservers) before all peers are fulfilled. There are approximately 20 peers left without the file when the simulation

ends, which is undesirable but still acceptable. When $R_{thresh}$ and $P_{thresh}$ were increased to 2 and 3, respectively, the results were worse. In particular, the end-condition is reached by the seventh generation and the decay of the demand and failed requests is slowers as more peers (approximately 310) are left unfulfilled.

These results show that as $R_{thresh}$ and $P_{thresh}$ decreases, performance increases. However, this occurs at the price of efficiency (the third and sixth series in Figure 8) which is defined as the ratio of *the estimate of the appropriate number of pseudoservers given current demand* to *the current number of pseudoservers*. An estimate of the appropriate number of pseudoservers for the given demand is calculated by dividing the demand in a given generation by the average number of neighbors in the system which is 8 for the Moore neighborhood considered in this paper. Intuitively, this says that the appropriate number of pseudoservers for a given generation is equal to the minimum number of pseudoservers such that every demander has a pseudoserver next to it. Note that the optimal value of efficiency is zero.

Now, returning to the threshold experiment, it was observed that with low threshold values, routers and pseudoservers are generated very aggressively which causes too many pseudoservers to be created. This, in turn, results in higher inefficiency. Figure 8 shows that, with low thresholds, during the middle of the simulation there are too many pseudoservers while the during the end of the simulation there are too few.

In summary, the simulation results point to three major observations for decentralized implementation: 1) increasing the routing contract values only results in marginal gains, 2) router and pseudoserver contract parameters ($Q$, $G_r$, $S$, and $G_p$) have optimal values that maximizes performance, and 3) low thresholds values improve performance, but at the cost of higher inefficiency.

## 8 Conclusions

To address the free rider problem in P2P networks we propose P2P contracts which combines an incentive based scheme with an admission policy. In this paper, we presented an utility-based framework to determine the components of the contract and formulate the associated resource allocation problem. We consider the resource allocation problem for the flash crowd scenario and show how the contract mechanism implemented using a centralized server can be used to quickly create pseudoservers that can serve out the requests. We then study a decentralized implementation of the P2P contract scheme in which each node implements the contract using local load information. We show that in such a system, other than contributing storage and bandwidth to

serve out files, it is also important that peer node function as application level routers to connect pools of available pseudoservers. We study the impact of the various parameters of the distributed implementation including the terms of the contract and the various triggers to create pseudoservers and routers.

## References


[1] M. Parameswaran, A. Susarla, and A. Whinston: P2P Networking: An Information-Sharing Alternative. IEEE Computer, 34(7):31-38, 2001.

[2] A. Warfield, Y. Coady, and N. Hutchinson: Identifying Open Problems in Distributed Systems. 2001 European Research Seminar on Advances in Distributed Systems (ERSADS), Bertinoro, Italy, 2001.

[3] SETI@HOME: SETI@HOME: The Search for Extraterrestial Intelligence. URL http://www.seti.org.

[4] Napster. URL http://www.napster.com.

[5] Gnutella. URL http://www.gnutella.com.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hing: Freenet: A Distributed Anonymous Information Storage and Retrieval System. Proceedings ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, 2000.

[7] FreeHaven. URL http://www.freehaven.net.

[8] Globus: The Globus Project. URL http://www.globus.org, 2002.

[9] A. Iamnitchi and I. Foster: On Fully Decentralized Resource Discovery in Grid Environments. International Workshop on Grid Computing, Denver, Colorado, 2001.

[10] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson: Economic Models for Management of Resources in Peer-to-Peer and Grid Computing. Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, 2002.

[11] Condor: The Condor Project. URL http://www.cs.wisc.edu/condor/, 2002.

[12] P. N. Yianilos and S. Sobti: The Evolving Field of Distributed Storage. IEEE Internet Computing, 2001.

[13] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer: Feasibility of a Serverless File System Deployed on an Existing Set of Desktop PCs. Proceedings of ACM SIGMETRICS, Santa Clara, CA, USA, 2000.

[14] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz: Maintenance-Free Global Data Storage. IEEE Internet Computing, 2001.

[15] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiatowicz: OceanStore: An Extremely Wide-Area Storage System. University of California, Berkeley, Berkeley UCB/CSD-00-1102, 2000.

[16] P. Druschel and A. Rowstron: PAST: A large-scale, persistent peer-to-peer storage utility. Proceedings of HOTOS Conference, 2001.

[17] A. Rowstron and P. Druschel: Pastry: Sacalable, decentralized object location and routing for large-scale peer-to-peer syatems. Proceedings 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, 2001.

[18] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowston: One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. Proceedings SIGOPS, European Workshop, France, 2002.

[19] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos: A Protoype Implementation of Archival Intermemory. Proceedings of the Fourth ACM Digital Libraries Conference (DL '99), 1999.

[20] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica: Wide-area cooperative storage with CFS. Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Banff, Canada, 2001.

[21] S. Rhea and J. Kubiatowicz: Probabilistic Location and Routing. Proceedings at IEEE INFOCOM, New York, NY, USA, 2002.

[22] P. F. S. Ratnaswamy, M. Handley, R. Karp, S. Shenker: A Scalable Content-Addressable Network. Proceedings of ACM SIGCOMM, 2001.

[23] Y. Zhao, J. D. Kubiatowicz, and A. Joseph: Tapestry: An Infrastructure for Fault-Tolerant Wide-area Location and Routing. Technical Report, University of California at Berkeley, UCB/CSD-01-1141, April 2000.

[24] E. Adar and B. Huberman: Free Riding on Gnutella. First Monday, 2000.

[25] S. Shenker: Fundamental Design Issues for the Future Internet. IEEE Journal on Selected Areas in Communications, vol. 13, pp. 1176-1188, 1995.

[26] O'Reilly: P2P Directory. 2001.

[27] A. Oram, Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. O' Reilly & Associates, 2001.

[28] S. Saroiu, P. K. Gummadi, and S. D. Gribble: A Measurement Study of Peer-to-Peer File Sharing Systems," Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), 2002.

[29] R. Albert, H. Jeong, and A.-L. Barabasi: Error and attack tolerance in complex networks. Nature 406 , 378 (2000).

[30] J. McCoy: Mojo Nation Responds. URL http://www.openp2p.com/pub/a/p2p/2001/01/11/mojoht.html, 2001.

[31] M. Hedlund and N. Minar: Popular Power. URL http://www.popularpower.com, 2001.

[32] P. Golle, K. Leyton-Brown, and I. Mironov: Incentives for Sharing in Peer-to-Peer Networks. ACM Conference on Electronic Commerce, 2001.

[33] P. Golle and S. Stubblebine: Secure Distributed Computing in a Commercial Environment. Financial Crypto, Anguila, British West Indes, 2001.

[34] D. A. Pierre: Optimatization Theory With Applications. New York: Dover, 1969.

[35] K. Kong and D. Ghosal: Mitigating Server-Side Congestion Using Pseudoserving. IEEE/ACM Transactions on Networking, August 1999.

[36] K. Kong and D. Ghosal: An User Responsible Paradigm for Internet Access. Computer Network and ISDN Systems, September 1998.

[37] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster: To Share or not to Share' An Analysis of Incentives to Contribute in Collaborative File Sharing Environments. Workshop on the Economics of Peer-to-Peer Systems, Berkeley, CA 94709, 2003.

[38] K. Lai, M. Feldman, I. Stoica, and J. Chuang: Incentives for Cooperation in Peer-to-Peer Networks. Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, 2003.

[39] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina: Incentives for Combatting Freeriding on P2P Networks (Research Note). Euro-Par 2003, Klagenfurt/Austria, August 26 - 29, 2003.

[40] K. Sigmund: Games of Life: Explorations in Ecology, Evolution, and Behavior. Penguin, 1995.

[41] Kazaa: URL http://www.kazaa.com.