Capitalizing on Free Riders in P2P Networks

Yuh-Jzer Joung[1], Terry Hui-Ye Chiu[12], and Shy MIn Chen[13]

[1] Dept. of Information Management, National Taiwan University, Taipei, Taiwan
[2] Dept. of Industrial Engineering and Management, MingChi Institute of Technology, Taishan, Taipei, Taiwan
[3] Dept. of Business and Management, MingChi Institute of Technology, Taishan, Taipei, Taiwan

Abstract. Free riding is a common phenomenon in P2P networks. Although several mechanisms have been proposed to handle free riding—mostly to exclude free riders, few of them have actually been adopted in practical systems. This may be attributed to the fact that they are nontrivial, and that completely eliminating free riders could jeopardize the sheer power created by the huge volume of participants in a P2P network. Rather than excluding free riders, in this paper we incorporate and utilize them to provide global index service to the files shared by other peers. The simulation results indicate that our model can significantly boost the search efficiency of a plain Gnutella, and the model is quite resistant to system churn rate and the ratio of free riding.

1 Introduction

Free riding is a common phenomenon in peer-to-peer (P2P) systems. The study by Adar et al. [1] in 2000 found that 66 percent of peers in Gnutella were free riders. Moreover, the top 1 percent of peers were responsible for 47 percent of requests, while the top 25 percent of peers provided for 98 percent. Follow up study by Hughes et al. [2] in 2005 found that free riders had increased to 85 percent, whereas the top 1 percent of peers were responsible for 50 percent of requests and the top 25 percent of peers still provided for 98 percent of requests.

Most people believe that free riding may degrade system performance and add vulnerability, possibly corruption, to the system. Therefore, several mechanisms have been proposed to handle free riding. These include *payment-based* [3, 4], where each resource has a price so that a peer needs to "pay" for downloading the resource; *reciprocal-based* [5, 6], where a peer's ability to consume resources is based on its contribution to the system;[4] *reputation-based* [7, 8], where a peer's ability to consume resources is based on it's reputation, which is evaluated by other peers based on past interactions with the peer; and *punishment* [9], where a free rider's request is eventually ignored to force it to disconnect from the network.

However, few of these proposals have been implemented in a practical system. For example, Gnutella, which has attracted more than two million users by

[4] In addition, BitTorrent's "tit-for-tat" peer selection strategy, which has been proven quite successfully in limiting free riding, may also be classified into this category. The BitTorrent platform, however, is very different from the Gnutella-like unstructured P2P networks we will be studying in the paper.

Dec. 2005, is still reluctant to implement any anti-free-riding measures. In fact, the above findings on Gnutella also show that even though the level of free riding has increased, the system can still sustain it. Moreover, even though free riders do exist, there are always some peers willing to share due to an altruistic or the so called "*glow worm*" effect. This has been supported by an empirical study on human behavior conducted by Gu et. al [10]. Another reason may be attributed to that existing measures for combatting free riders often have high overhead on communication, storage, and/or computation, and some even require an external centralized authority.

By observing human behaviors, we believe that free riders may always exist regardless of any action taken against them. On the other hand, a P2P network is often appraised by the sheer power created by the huge volume of its participants (for example, such a network is particularly useful for disseminating information, for, say, marketing and advertising). Completely eliminating free riders could jeopardize its power. Therefore, why not to make free riders part of the network and try to utilize them, rather than expelling them from the network?

Based on the studies in [1, 2, 11, 12], we can see that free riders have the following characteristics: (a) short connection time, (b) often requesting popular files, and (c) obviously, they do not like to share—if they have a choice. This motivates us to think of making them to contribute what they cannot control (e.g., the platform and the protocol) and, usually, do not care much.

In this paper we propose a Gnutella-like unstructured P2P network that utilizes free riders (those that share none) to serve as indexing nodes to normal peers (those that share some files). Since free riders often request popular files, they can easily point to the peers that have the target files one is looking for. Free riders' short connection time can be overcome by the large number of their pals. Our simulation results indicate that the network can significantly boost the search efficiency of a plain Gnutella. Moreover, routing and search costs are effectively shifted to free riders and, with a proper setting, the search success rate of normal peers can be much higher than that of free riders—all of these provide some incentive for peers to contribute files to the network. More importantly, our network is quite resistant to system churn rate and the increasing ratio of free riding. Below we present our system.

2 The System

2.1 Neighbor Tables

Like ordinary P2P networks, each peer u in our model maintains a *neighbor table* of contact information about peers to which u may forward a query. The contact information consists of node IP and port. We say that v is a *neighbor* of u if u maintains v's contact information in its neighbor table.

In addition, each entry in the neighbor table also contains a Bloom filter [13, 14] summarizing the files shared by the corresponding peer (see Fig. 1). With the Bloom filters in the neighbor table, a peer receiving a query can check if any peer it maintains in the neighbor table may have the queried file. If so, the query can be sent directly to the peer for a final check. Otherwise, the query will be resolved using our search mechanism that is to be described shortly in Section 2.3. The fields TTL and Version in a neighbor table entry are for maintenance of the entry. This will also be clear shortly.

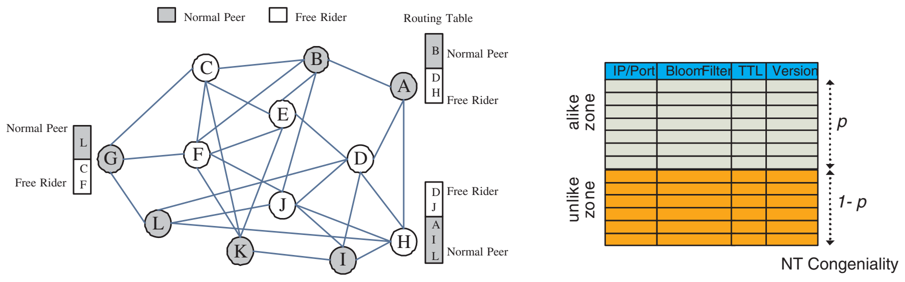

Fig. 1. Overview of network architecture (left) and the neighbor table structure (right).

A more important role of the Bloom filters is for a peer to select neighbors in its neighbor table, which allows some form of clusters to be established in the network, thereby speeding up search process. For our purpose, a peer u's neighbor table is partitioned into two zones: *alike* and *unlike*. Intuitively, the alike zone is for peers that are "akin" to u in terms of whether or not they wish to share, while the unlike zone is for peers that are "different" from u. Thus, a normal peer's neighbors in the alike zone are all normal peers, and its neighbors in the unlike zone are all free riders. On the other hand, a free rider's neighbors in the alike zone are all free riders, while its neighbors in the unlike zone are all normal peers.

When there are more candidates than a zone can fit, a replacement scheme is needed to help select better neighbors in the zone. There are several possible schemes. Here we select peers based on how "similar" their Bloom filters are (i.e., based on the keywords they have in their files). Specifically, let B_1 and B_2 be two Bloom filters, and let $B_1 \wedge B_2$ denote the Bloom filter B such that $B[i] = B_1[i] \wedge B_2[i]$. Then

$$sim(B_1, B_2) = \Big|\{i \mid B[i] = 1, \text{where } B = B_1 \wedge B_2\}\Big|$$

That is, $sim(B_1, B_2)$ is the number of bit-1 the two Bloom filters have in common.

The replacement scheme for the alike zone is as follows: u prefers v to w in filling u's alike zone if their Bloom filters B_u, B_v satisfy the following condition: $sim(B_u, B_v) > sim(B_u, B_w)$. If $sim(B_u, B_v) = sim(B_u, B_w)$, then the one that is *least recently used (LRU)* is replaced. In addition, we require that a peer v can fill in u's alike zone only if $One(B_u) > 0 \Rightarrow One(B_v) > 0$, and $One(B_u) = 0 \Rightarrow One(B_v) = 0$, where $One(B)$ denotes the number of bit-1 in B. Because a normal peer's Bloom filter is a nonzero vector, the condition implies that only normal peers can fill in a normal peer's alike zone, and only free riders can fill in a free rider's alike zone.

On the other hand, a peer that cannot fill in u's alike zone is a candidate for u's unlike zone, and if there is more than one candidate, the replacement scheme is simply based on LRU. Note that by the criterion given for the alike zone, only free riders can fill in a normal peer's unlike zone, and only normal peers can fill in a free rider's unlike zone.

We call the percentage p of a neighbor table that is allocated to the alike zone the *NT congeniality.* Thus, $1 - p$ is the percentage of the unlike zone in a neighbor table.

2.2 Network Construction and Maintenance

Our network, as shown in Fig. 1, is a typical Gnutella like unstructured P2P network. A peer u that wishes to join the network first prepares a Bloom filter to summarize the files it wishes to share. Then it sends PING-PONG messages via some hook-up node to the network. Like the Gnutella protocol, the purpose of the messages is to learn a list L of nodes in the network. ¿From the list, u can build its neighbor table according to the rules described in Section 2.1, thereby connecting itself to the network. Note that during the operation of the network, u may also learn more peers in the network, e.g., when other peers join the network and send PING-PONG messages to u, or when u involves in a search process in which query messages from other nodes arrive at u. The new peer information allows u to modify its neighbor table entries so as to improve search efficiency. Again, neighbor table entries are replaced according to the rules described in Section 2.1.

Because a peer may leave the network without notifying other peers that maintain it as a neighbor, neighbor table entries may become stale. Moreover, when peers change the files they wish to share, their Bloom filters must also be updated. So in the network each peer must periodically refresh its neighbor table entries. The TTL field in a neighbor table entry (see Fig. 1) allows the owner of the entry to determine when the entry needs to be verified with the peer specified in the entry. The version field allows both peers to verify if the Bloom filter in the entry is up to date.

2.3 Search Scheme

The basic idea of our system is to let free riders collectively provide index service to normal peers. To do so, we recall that by the neighbor selection mechanism, a normal peer will have normal peers in its alike zone and free riders in its unlike zone; while a free rider will have free riders in its alike zone and normal peers in its unlike zone. If the size of unlike zone is relatively large compared to the size of alike zone, then a normal peer connects to a lot of free riders, while each of the free riders connects to a lot of normal peers. By storing normal peers' Bloom filters at free riders, the free riders can collectively provide index service to normal peers.

To make use of the index service, the search scheme is simply as follows: Let u be a peer that receives a query. If u has the target file, then it returns the file to the querier and the search process ends. Otherwise, u checks the Bloom filters in its neighbor table to see if any of its neighboring peers may have the file. If so, it forwards the query message to the neighbor, say v, to confirm this. If v does have the file, it sends the result to u, which in turn forwards the result to the querier and then ends the search process. Note that v will not forward the query message for u if it does not have the file. In general, in the absence of false positives, normal peers do not involve in the search process unless they have the target files.

If none of u's neighboring peers has the queried file, u forwards the query message to all free riders in its neighbor table, and waits for the result. Each of

u's free rider neighbors, upon receiving the query message from u, processes the query in the same way as u does, and so on, until a certain depth of search path has been reached. If the target file is located at some peer w, then w returns the file along the search path back to u that initiates the query. Each peer in the returning path learns of w (and its Bloom filter), and may add u to its neighbor table if w can replace an existing entry.

3 Experimental Result

3.1 Evaluation Metrics

Observe that if objects are randomly distributed to a network of N nodes, then the probability for locating a unique object o from a randomly selected set of P nodes is $\frac{P}{N}$. By comparing to this random search, we can evaluate whether a system or a search mechanism can offer a better success rate than a blind search. Therefore, we define a metric called *search condensity* as follows: Let $\mathcal{A}$ be a search mechanism over a network of N nodes. Suppose that the search success rate is s when the search space size (number of nodes visited) is P. Then the search condensity SC of $\mathcal{A}$ is defined by

$$SC = \frac{s}{\frac{P}{N}} = s \times \frac{N}{P}$$

That is, SC is a measure of how effective a mechanism can "boost" the search success rate of a network as compared to a random blind search. Note that SC may vary with P and N, and so it is sometimes more meaningful to write SC as a function of P/N, i.e., $SC(P/N)$. Also note that $SC(1) = 1$.

The second parameter is the workload for a peer to process a query. We measure the workload of a peer by the number of query messages it has received during some observation interval.

The third parameter is to measure the duplication ratio of query messages. Observe that because search in an unstructured network is more or less a blind process, duplicated query messages may arrive at the same node. The *query duplication ratio* measures how many query messages are duplicated during a search process. It is defined as follows: Let P be the search space size of a query, and let M be the total number of query messages sent for the query. Then the query duplication ratio is defined by $\frac{M}{P}$.

3.2 Dataset

Since real P2P dataset is hard to obtain, we use Web proxy logs collected from Boeing[5] to simulate file distribution in a P2P environment. The logs basically record which clients have requested which URLs. We use the file path in a log to represent a queried file owned by the host IP in the URL. The dataset we used consists of 3,761,054 URLs belonging to 31,743 Web hosts, which then allows us to model 3,761,054 files distributed over 31,743 peers.

Because query in our system is by keywords, we need to extract keywords from each queried file to represent the file. To do so, we extracted words from the

[5] See http://www.web-caching.com/traces-logs.html.

Table 1. Statistics of the dataset obtained from the Boeing Proxy Log.

owner	owns	max	min	avg
Peer	File	103,880	1	87.28
Peer	Keyword	981	1	35.87
File	Keyword	137	1	4.47

Table 2. Default simulation settings.

Parameter	Default Value
Network Size	50,000
Free Rider Ratio	0.85
Routing Table Size	20
Bloom Filter Size	8,000
Median Session Time	36,000
Average Message Delay	2.84
Search TTL	3
Query Frequency	0.5 per time unit

file path using the following characters '/', '?', '-', '_', and '&' as delimiters. After the process, each word may contain some non-alphanumeric characters such as '%' and could have a length more than 20. For simplicity, we further removed non-alphanumeric characters, and truncated words that have length more than 20. The remaining words are then treated as keywords associated with the file. To avoid having a long Bloom filter, a host that has more than 1,000 keywords in its files was also removed from our dataset. In total, 61 out of 31,743 hosts were removed. Table 1 gives some statistics about the dataset. Moreover, on average the peer support of a file (number of peers having the file) is nearly 1.

3.3 Simulation Settings

Our simulator is written in Java. It uses a single-process event-driven architecture to simulate concurrent activities of peers, such as joining, querying, and message delivery. All time measured in the simulation will be relatively to the time unit of the simulator (which is calibrated to 0.1 second.) The default simulation settings are shown in Table 2.

To simulate the dynamics of P2P networks, each peer will be given a session time when it connects to the network. We use the method presented in [15] to model peer session time so that peers' behaviors are close to the empirical results studied in [16]. Moreover, free riders typically have shorter session time than normal peers [11]. Our network dynamics model will also reflect this.

In the simulation, when we need a network of size N, the network is constructed and maintained as follows. Initially, we let N nodes concurrently join the network. After the join process is completed, we start to time each peer's session, and remove peers from the network when their sessions end. To keep the network size roughly stable, when a peer u leaves the network, a random peer that is currently offline is "awakened" to rejoin the network. Similarly, u becomes offline until it is awakened by another peer. Note that when a peer first joins the network, it will randomly contact 3-5 hook-up nodes to build its neighbor table. When a peer leaves and rejoins the network, it refreshes its neighbor table by first checking if its previous neighbors are still alive, and also randomly contacts

3-5 hook-up nodes to find new peers that may fill in its neighbor table. Entries in the neighbor table are replaced in the same way as described in Section 2.1.

Free riders are chosen randomly from the network according to their population. For normal peers, we let each node randomly map to a unique Web host in our Boeing dataset so that the nodes simulate normal peers that own some files. All querying activities are performed after the join process is completed. To simulate the querying process, we randomly select an alive peer in the network as the originator of the query. The query target is again randomly selected from some existing file, and is represented by the keyword set of the selected file.

3.4 Performance Evaluation

We measure the performance of the system from various perspectives. Note that it takes time for each node to learn of other peers to fill in its routing table so as to meet the peer selection requirement in its alike and unlike zones. According to our preliminary test, for most parameters we shall be studying, the system becomes stable approximately after one query has been issued per peer. So, unless stated, otherwise all measurements in the experiments are taken after 50,000 queries have been issued to the system.

We first measure search condensity with respect to NT congeniality p. We vary $p = 0.0$ to 1.0 on a scale of 0.1. The results are shown in Fig. 2(a). We see that except for $p = 0.0$, small p generally implies high search condensity. Observe that search condensity is a measure of how "effective" query messages are issued to search the target. In general, high search condensity may be obtained when the search space size is small. However, search success rate may also be compromised due to this small search space. In Fig. 2(b) and (c) we have also drawn the success rate and the actual search space (the number of peers that have been queried per search) measured in the experiment for each p. We see that the overall success rate reaches its high 0.58 when $p = 0.5$, but drops to 0.31 when $p = 0.1$, and to 0.44 when $p = 0.9$ on the other side. The actual search space size, however, grows in polynomial with p. To see the trend of a search space size, recall that in our search scheme, query messages will only be forwarded to free riders to expand the search scope. So when a normal peer initiates a query, if the neighboring normal peers do not have the target file, then $(1-p)n$ queries will be forwarded to the neighboring free riders, where n is the neighbor table size. In contrast, if a query is initiated at a free rider, then $p \times n$ queries may be forwarded to its neighboring free riders. For TTL=3, a normal peer's query may expand to $(1-p)p^2n^3$ peers, while a free rider's query may expand to p^3n^3 peers. Taking the percentage of free riders and normal peers into account, on average a query should have a search space of size

$$(0.15(1-p) + 0.85p)p^2n^3. \tag{1}$$

So the size grows in polynomial with p. By combining (b) and (c), it is not surprising to see that search condensity drops almost in polynomial with p.

Note that $p = 0$ and $p = 1$ represent two extreme cases in our model. When $p = 0$, a normal peer's neighbors are all free riders, while a free rider's neighbors are all normal peers. So a query initiated by a normal peer will be forwarded to its neighboring free riders. If they happen to have indexed the target file, the search succeeds; otherwise, the search fails as no more free riders can be used to forward the query. On the other hand, if a query is initiated by a free rider, then

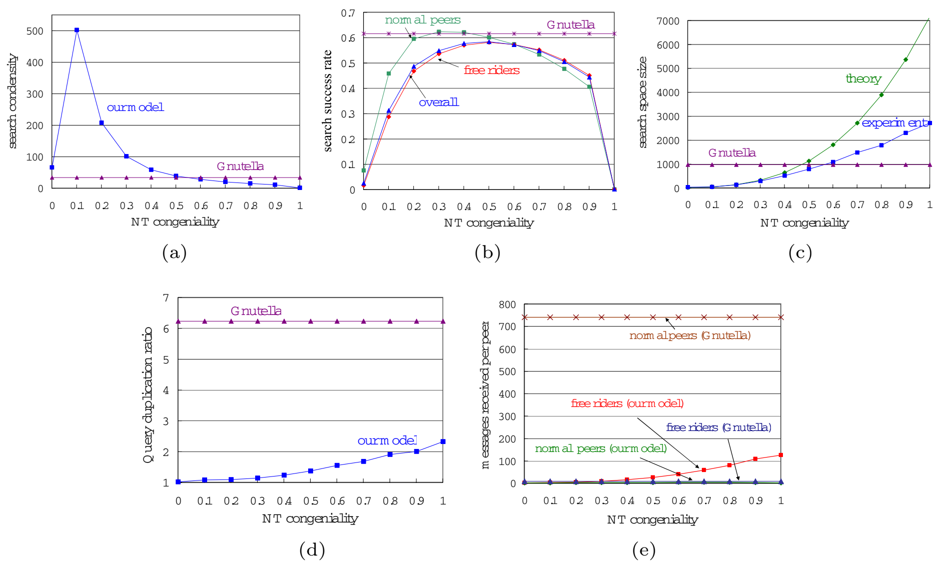

Fig. 2. NT congeniality vs. (a) search condensity; (b) success rate; (c) search space size; (d) query duplication ratio; and (e) workload.

the query is resolved if the free rider's neighboring normal peers have the target file; otherwise the search fails as the free rider has no neighboring free riders to forward the query. For $p = 1$, the situation is rather opposite. A normal peer's neighbors are all normal peers, while a free rider's neighbors are all free riders. So a query initiated by a normal peer can be resolved if the peer or its neighbors happen to have the target file; and a query initiated by a free rider will have no chance to be resolved, as all peers within the search range are free riders and they do not index any normal peers.

In Fig. 2(b), we have also identified the search success rate of normal peers and free riders. We see that when $p \leq 0.5$, normal peers have higher search success rate than free riders, and the difference is significant when p is small. Moreover, when $p = 0.2 - 0.5$, normal peers' search success rate is even higher than the maximum overall success rate. This suggests that when setting NT congeniality to these values, our mechanism can also create some incentive to normal peers, as they will have a higher search success rate. All together, we see that $p = 0.2$ appears to be an attractive setting, as it can yield quite high search condensity (206.6) without compromising too much in search success rate for normal peers.

The search space size in Eq. (1) assumes that no two peers within the search range have a common neighbor. Otherwise, the actual search space will be smaller, and query message duplication ratio will increase. From Fig. 2(c), it can be expected that the query duplication ratio will increase as p increases. The measured query duplication ratio is shown in Fig. 2(d).

The workloads of normal peers and free riders are shown in Fig. 2(e). The workload is measured by the number of query messages received per normal peer/free rider for every consecutive 1,000 queries issued to the system. We see that the workload of free riders grows in polynomial with p, while the workload of normal peers is barely observable. This is because in our search scheme, a normal peer x can receive a query forwarded from another peer y only when x's Bloom filter maintained by y indicates that x may have the target file. So the workload of normal peers is solely due to a successful search or false positive of the Bloom filter. On the other hand, free riders are responsible for forwarding and processing queries. As shown in Fig. 2(c), the search space grows in polynomial with p. Therefore, the workload of free riders also grows in polynomial with p.

For comparison, the search performance of the plain Gnutella is also shown in the figures. Since in the plain Gnutella a peer selects its neighbors based only on LRU, the NP congeniality parameter has no effect on the system performance. So all measured values remain constant at the y-axis. In Fig. 2(a), Gnutella's search condensity is about 32.5, which is considerably low as compared to our scheme at small p. Note that because a successful search will also cause all peers on the query-result returning path to update their neighbor tables, search in the plain Gnutella is not exactly a random search. So its search condensity is higher than 1. In Fig. 2(b), Gnutella's search success rate is about 0.61, which is slightly higher than ours at small p. (There is virtually no difference between normal peers and free riders in the success rate.) However, this high success rate is at the cost of large search space size. For example, in Fig. 2(c), Gnutella's search space is 958, as opposed to 118 needed in our scheme at $p = 0.2$. Moreover, Gnutella's query duplication ratio in Fig. 2(d) is considerably high. In Fig. 2(e), the workload of normal peers in Gnutella is also very high, while there is virtually no workload to free riders. This is because as time goes by, both normal peers and free riders will maintain only normal peers as their neighbors in their routing tables, thus yielding all query loads to normal peers. Because there are only 15% of normal peers, each normal peer may be connected to by a number of peers, and hence the high query duplication ratio.

4 Concluding Remarks

We have proposed a Gnutella-like unstructured P2P network that utilizes free riders to serve as indexing nodes to normal peers. To do so, when building the network, we partitioned a peer's neighbors into two groups: one that is "congenial" to the peer, and the other that has an opposite characteristic. As a result, a normal peer's congenial neighbors are normal peers, while its uncongenial neighbors are free riders. On the other hand, a free rider's congenial neighbors are free riders, while its uncongenial neighbors are normal peers. In addition, we used Bloom filters to summarize the files of the peers, and let each peer keep a copy of its neighboring peer's Bloom filter. Let p be the portion of the congenial neighbors within a peer's neighbor table. Then when p is small, free riders collectively can provide index service to the files shared by normal peers. Likewise, the search scheme is simply to forward queries to free riders, as they have more information about which normal peers may have the target files. This effectively shifts the routing and search costs to free riders. Hence, a peer either contributes some files to other peers, or provides indexing service to them. Note that the indexing service is part of the network protocol, and so it is out of a peer's control—unless it wishes to hack the protocol. Security threats are a general problem in P2P networks, and they are beyond the scope of the paper.

Our simulation results indicate that with a network of size 50,000, 85% of free riders, a neighbor table of size 20, and a search TTL of 3, setting $p = 0.2$

can significantly boost the search condensity of a plain Gnutella—206.6 vs. 32.5! Moreover, the search success rate of normal peers is also higher than that of free riders (0.59 vs. 0.47), thereby offering some incentive to normal peers. More importantly, our network is quite resistant to system churn rate and the increasing ratio of free riders (although, due to space limitation, this part of simulation result is not shown here). In fact, the higher the free riding ratio, the more the number of peers to provide indexing service, and therefore the higher the search success rate and search condensity.

In our model, we distinguish normal peers and free riders simply based on whether or not they have contributed any file to the network. Although this is a typical distinguishing criterion, a more practical (but sophisticated) approach is to judge a peer based on its real contribution to the others: for example, how many files it offers, how many of them have actually been requested, and how frequent of the requests. In the future we wish to modify our system so that a peer's search efficiency and workload can truly reflect how much it actually contributes.

References

1. Adar, E., Huberman, B.A.: Free riding on Gnutella. First Monday **5**(10) (October 2000) Available from http://firstmonday.org/issues/issue5_10/adar/index.html.
2. Hughes, D., Coulson, G., Walkerdine, J.: Free riding on Gnutella revisited: the bell tolls? IEEE Distributed Systems Online **6**(6) (2005)
3. Golle, P., Leyton-Brown, K., Mironov, I.: Incentives for sharing in peer-to-peer networks. In: ACM EC 2001, 264–267.
4. Figueiredo, D., Shapiro, J., Towsley, D.: Incentives to promote availability in peer-to-peer anonymity systems. In: ICNP 2005, 110–121.
5. Vishnumurthy, V., Chandrakumar, S., Sirer, E.G.: KARMA: A secure economic framework for peer-to-peer resource sharing. In: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems. (2003)
6. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: Incentives for combatting freeriding on P2P networks. In: Euro-Par 2003. LNCS 2790, 1273–1279.
7. Dutta, D., Goel, A., Govindan, R., Zhang, H.: The design of a distributed rating scheme for peer-to-peer systems. In: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems. (2003)
8. Hales, D.: From selfish nodes to cooperative networks - emergent link-based incentives in peer-to-peer networks. In: P2P 2004, 151–158.
9. Karakaya, M., Korpeoglu, I., Ulusoy, Ö.: A distributed and measurement-based framework against free riding in peer-to-peer networks. In: P2P 2004, 276–277.
10. Gu, B., Jarvenpaa, S.: Are contributions to P2P technical forums private or public goods? – An empirical investigation. In: Proceedings of the 1st Workshop on Economics of P2P Systems. (2003)
11. Yang, M., Zhang, Z., Li, X., Dai, Y.: An empirical study of free-riding behavior in the maze P2P file-sharing system. In: IPTPS 2005, LNCS 3640, 182–192
12. Beverly IV, R.E.: Reorganization in network regions for optimality and fairness. Master's thesis, MIT EECS. (2004)
13. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. CACM **13**(7) (1970) 422–426.
14. Cheng, A.H., Joung, Y.J.: Probabilistic file indexing and searching in unstructured peer-to-peer networks. Computer Networks **50**(1) (2006) 106–127.
15. Joung, Y.J., Wang, J.C.: Chord^2: A two-layer chord for reducing maintenance overhead via heterogeneity. Computer Networks **51**(3) (2007) 712–731.
16. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: MMCN 2002.