Non-Cooperation in Competitive P2P Networks

Beverly Yang Tyson Condie Sepandar Kamvar Hector Garcia-Molina
{byang@cs, tcondie@leland, sdkamvar@leland, hector@cs}.stanford.edu
Computer Science Department, Stanford University

Abstract

Large-scale competitive P2P networks are threatened by the non-cooperation problem,[1] *where peers do not forward queries to potential competitors. Non-cooperation will be a growing problem in such applications as pay-per-transaction file-sharing, P2P auctions, and P2P service discovery networks, where peers are in competition with each other to provide services. Here, we show how non-cooperation causes unacceptable degradation in quality of results, and present an economic protocol to address this problem. This protocol, called the RTR protocol, is based on the buying and selling of the right-to-respond (RTR) to each query in the network. Through simulations we show how the RTR protocol not only overcomes non-cooperation by providing proper incentives to peers, but also results in a network that is even more effective and efficient through intelligent, incentive-compatible routing of messages.*

1 Introduction

In recent years, the distributed systems community has shown great interest in resource-sharing over peer-to-peer (P2P) systems. Existing P2P systems have been able to reach enormous scales because of their ability to pool together and harness large amounts of resources. In addition, because peers come from different organizations, no one organization bears the cost of supporting the infrastructure, and peers can leverage data, services and expertise across organizations.

The key to the usability of a data-sharing peer-to-peer system is the ability to search for and retrieve resources such as files, records, service descriptions, etc. Indeed, much existing research (e.g., [2, 17, 19]) has focused on efficient and expressive *resource discovery* mechanisms. However, while the solutions put forth in these works address critical problems, most cannot function unless one can assume that peers cooperate with one another. Unfortunately, such an assumption will not hold in general P2P application scenarios, because peers are autonomous and come from potentially competing organizations.

In particular, in terms of resource discovery, one crucial problem facing future P2P applications is *competition* among peers to provide services (e.g., sharing data). Increasing emphasis is being placed on new *legitimate* applications of P2P, such as pay-per-transaction networks, P2P auctions, and P2P service discovery networks. In these applications, peers *gain* from answering queries. For example, in a pay-per-transaction file-sharing network where peers get paid for uploading files, peers will want to share files, because this generates income. Therefore, not only are peers eager to provide services (e.g. share files), but they are in *competition* with other peers to provide their services.

Competition is a serious problem in P2P resource discovery frameworks that rely on peers to forward queries (e.g., Gnutella [6], DHTs like [17], etc.),[2] because a peer acting in its own best interests will not forward queries to potential competitors. For example, a peer providing a car rental service might not forward a query for car rental services. Instead, it could answer the query and then drop it, so as to improve its chances of gaining business. As a result, the P2P network will no longer operate correctly due to non-cooperation, even though abundant services are available.

Furthermore, competition is much more subtle than such malicious behavior as indiscriminately dropping queries, since only a crucial fraction of messages are dropped competitively. Existing message-forwarding incentive mechanisms such as [18] tend to employ strategies in which peers reciprocate message-forwarding volume. However, such an approach cannot differentiate between peers who act competitively and peers who have, for example, one fewer neighbor than the average peer, as both will simply appear to have a slightly lower volume of forwarded messages; therefore, these existing mechanisms are insufficient to address competition.

In this paper, we propose an economic protocol to ensure that peers cooperate in the *operation* of P2P networks in the face of competition. The basic idea behind the protocol, called the *RTR protocol*, is to have peers *purchase* queries from one another. Peers have both an incentive to buy queries, since they provide potential business, and an incentive to sell them, because they are assigned a price. In addition to overcoming competition, the RTR protocol improves overall system efficiency through techniques that are compatible with individual peer incentives.

We note that the work presented in this paper is an early attempt at addressing a new problem. While much existing work has focused on performance of P2P systems, and others have looked at different types of incentives issues (e.g., free-riding, indiscriminately dropping queries), ours is the

[1] A brief position paper on this topic appears earlier in [22].

[2] The exceptions are systems that require no forwarding, such as the old Napster (http://www.napster.com).

first to address the dual issues of competition *and* performance in resource discovery. Therefore, much of this work is still exploratory in nature. Our goal is to lay a foundation for addressing this new problem by identifying the relevant issues, providing simple models by which to reason about the problem, and presenting one potential solution that we believe captures the necessary components of any general solution to the competition problem.

Our contributions in this paper are as follows:

- We identify (Section 2) a new and relevant problem facing economic P2P applications, the *non-cooperation* problem, and quantify its effects.
- We present (Section 3) the *RTR protocol*, a potential solution to the non-cooperation problem.
- We discuss (Section 3.4) the robustness of this protocol, and provide means to avoid and punish cheating peers.
- We analyze (Section 5) the performance of the RTR protocol and show how it effectively addresses the non-cooperation problem.

In this work, we illustrate our protocol on top of the Gnutella protocol for P2P search running a pay-per-transaction file-sharing application. This application is chosen as a starting point for our study for two reasons. First, free file-sharing has proven itself to be a popular application, but due to its often illegal activities, a push is being made towards legal solutions involving payment (e.g., [9]). We therefore assume our users are ones who are willing to pay for file download, as are the users of [9], and to sell files under the approval of the copyrighted owners. Second, file-sharing has a naturally simple pricing model in which all files have the same value.

We selected the Gnutella protocol because, in the face of competition, unstructured networks are more appropriate than structured ones, which rely heavily on coordinated, cooperative behavior. Important future work lies in extending these ideas to systems with more complex valuation of goods and over different search architectures, such as DHTs (e.g., [17]) and GUESS [8]. The following discussion assumes the existence of an efficient micropayment scheme for P2P systems, such as that described in [21], and a public-key infrastructure.

2 Background and Motivation

The basic Gnutella search protocol [6] works as follows: each user runs a client (or *peer*), which is connected to a small number of other peers (known as *neighbors*) in an overlay network. When a user submits a query, her peer will send the query message to all its neighbors, who will in turn forward the query to their neighbors, and so on. A peer that receives a query and finds that it can answer will send a response directly to the querying peer.[3] The querying peer will wait a period of time for responses to arrive, and then it will select one or more responding peers from which to buy services (e.g., paying to download a file). Clearly, if peers do not forward queries, the search mechanism will fail.

[3] In Gnutella, response messages are actually forwarded along the reverse path traveled by the query. We modify the protocol to be more efficient and respect the privacy of the responder.

	Score	# Peers Responded	# messages
All	19.0	29.0	120.0
Comp.	2.1	3.9	28.2
No	1.1	1.9	4.0
RTR_{ef}	19.3	29.5	122.5

Table 1. All, competitive and no-forward behavior, compared with the RTR protocol

Effects of Non-Cooperation. In the case where peers cooperatively forward queries to each other, we say the peers are following the *all-forward* mode of behavior. In the non-cooperation case, peers can refuse to cooperate under two models: (1) *no-forward*, and (2) *competitive-forward*. Under *no forward*, a peer forwards no queries. Under purely rational behavior, a peer will adopt no-forward behavior, because it would need to consume processing and bandwidth resources to forward queries, with no gain for itself.

However, no-forward behavior is easy to detect and punish; employing existing incentive mechanisms such as [18] may be enough to prevent peers from dropping all messages. A more subtle misbehavior is the *competitive-forward* model, in which peers do not forward queries only if doing so will increase competition for that peer. To illustrate, consider a peer P in a file-sharing P2P application that responds to a query q. Because P will gain income if it is chosen to upload a file for q, it can maximize its expected income by not forwarding q to any of its neighbors, even if P only has an approximate answer (one that has just a small chance of being uploaded). Without a global view of the incoming and outgoing messages of a peer, which is unavailable in almost any P2P network, existing mechanisms are unable to detect competitive-forward behavior.

In Table 1, we compare the performance of the Gnutella file-sharing network under the all, competitive and no-forward cases. For now, ignore the row describing the RTR protocol. A detailed description of the setup for this experiment can be found in Section 4. We measure performance in terms of quality of results and efficiency. Quality of results is reflected by how many peers responded to each query, on average, and the "score" of the response set (described in Section 4.3).

From this table, we see that both competitive and no-forward behaviors result in significantly degraded quality of results, when compared to all-forward. For example, in the no-forward case, the number of peers that process a query is 15 times smaller than in the all-forward case. Although competitive-forward outperforms no-forward, it still has an average response score that is just 12% of the score under all-forward behavior. Considering the importance of network variety to users of file-sharing networks, we see a clear need for overcoming non-cooperative behavior. We note that, not surprisingly, messaging overhead is roughly proportional to quality of results. However, since bandwidth and processing are renewable resources, we consider this cost small compared to the gain in user satisfaction.

Our goal in addressing non-cooperation is the following: *to allow the network to achieve the same quality of results and efficiency as all-forward, even in the face of competitive peers*. Indeed, we will see later that under the RTR protocol, the network adapts to form a more efficient topology over which we maximize the quality-to-cost ratio of the network.

3 RTR Protocol

At the core of our protocol is the concept of a *right to respond*, or RTR. An RTR is simply a token signifying that a peer has a right to respond to a query message. We choose this name ("right to respond") in order to emphasize that a query is really a commodity. Peers have an incentive to *pay* to receive the query, because that in turn brings in potential business. An analogous concept in real-life markets are companies that buy lists of emails or referrals from other companies, to obtain a new pool of potential customers.

Once a peer buys an RTR for a given query, it may (a) respond to the query and hope that it is chosen to upload its services, and/or (b) sell the RTR to other peers.[4] Peers can buy and sell RTRs with their neighbors only.

In this framework, selling an RTR is equivalent to forwarding a query. There is clearly an incentive to forward queries, since peers earn income in doing so. Of course, some peers may still choose to not forward any queries in order to increase the probability that they will be chosen to provide the service. However, their actions will be offset by those peers who hedge their risk by selling a few RTRs, and by those peers who speculate in RTRs (buying RTRs simply to resell them).

It is important to note that all actions taken by the peer are *automated* – buying and selling RTRs, pricing, filtering, making connections (described later), etc. While human users could theoretically make each decision, doing so would be tedious, error-prone, and not worth the time investment. Instead, the most reasonable implementation follows the model used by investment companies in the real world: users specify their high-level preferences, such as risk level, and the peer sets decision parameters to produce behavior accordingly. We will discuss these specific parameters in further detail later (Section 4.2).

3.1 Basic Implementation

An RTR has the following format:

$$RTR = \{Q, ts, query\}_{SK_Q} \qquad (1)$$

where Q is the identity of the querying peer, ts is the timestamp at which the query was first issued, and $query$ is the actual query string. These three values are signed by the querying peer's secret key SK_Q, so that RTRs cannot be forged. Hence, each query requires a single signature generation, and optionally one verification per forward.[5]

[4]If a peer sells an RTR, it may still respond to the query corresponding to the RTR. That is, a peer does not lose the right to respond to a query when it sells the RTR for that query.

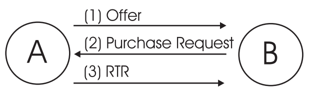

Figure 1. RTR Protocol

When a peer A forwards a query to a neighbor B, it will first send an *offer* containing partial RTR information and a price:

$$\mathit{Offer} = \{rep(Q), ts, query, price\} \qquad (2)$$

where $rep(Q)$ is the reputation of the querying peer (described below). Creating and sending an offer requires that peer A make several decisions. First, using quality and relevance filters discussed in further detail below, peer A must intelligently select which RTRs neighbor B will want to receive. Peer A must also determine a price at which to offer the RTR (discussed in the next section).

An offer contains enough information for B to determine whether to purchase the RTR, and whether the RTR is a duplicate B has seen before. However, because the identity of Q is not revealed, B cannot actually answer the query without purchasing the full RTR. If B decides not to purchase the RTR, he will simply drop the offer. Otherwise, B will send a *purchase request* to A, and peer A will forward the full RTR to B. The RTR protocol is summarized in Figure 1. After purchasing an RTR, a peer will respond directly to the querying peer.

In terms of overhead, 3 messages are exchanged in the RTR protocol for every successful query forward, as opposed to 1 in the Gnutella protocol. However, we will see in Section 5 that the RTR protocol is just as efficient, sometimes *more* efficient, than Gnutella due to intelligent and incentive-compatible decision making.

Filters. To prevent being "spammed" by useless offers, each time a peer connects to a new neighbor, it can specify flow control parameters in the form of *filters*, that specify the "content" and "quality" of the desired RTRs. Filters can be set on any of the three fields of an RTR: the query string, the reputation of the querying peer $rep(Q)$, or the timestamp. Filters on the reputation of querying peer Q and the age of the query (indicated by the timestamp) specify the *quality* of the RTR, and affect the probability that a responding peer will be chosen and paid for its services.

Filters on the query string specify the *content* of the RTR, and affect the probability that the purchaser can respond to the query. Content filters may have varying levels of restrictiveness. For example, a content filter may specify that the RTRs should only contain queries for a particular genre of music files. Users simply select the level of specificity desired, and the peer will automatically set filters according to the files or services offered. For certain types of filters

[5]Verification can be done on a random sampling basis to determine trustworthiness of a peer.

(e.g., categories or genres), we may assume the existence of a predefined taxonomy of services and files.

3.2 Pricing

In this section, we discuss a simple pricing model for RTRs. Like the pricing of any commodity in the real world, the pricing of RTRs will involve the estimation of many parameters, as well as a user's disposition (e.g., risk-averse). The purpose of the model is not to provide a straightforward price for the RTR, but to help us understand the factors that influence price, and pinpoint the parameters that need to be estimated. In Section 4.2 we describe how peers may use the model to automatically price RTRs and estimate the necessary parameters, based on high-level user inputs.

Model. Let *RTR*$_f$ denote an RTR corresponding to a query for file f. We assume any file f has a well-known price $price(f)$. Let N_A be the random variable denoting the income generated by the RTR for peer A, if A owns *RTR*$_f$. The income generated for A by holding *RTR*$_f$ has two components:

$$N_A = S_A + R_A \quad (3)$$

where S_A is the random variable denoting the income gained for *selling a file* to the querying peer, and R_A is the random variable denoting the income gained from *reselling the RTR*. The value of *RTR*$_f$ for peer A is simply $E(N_A)$, the expected value of N_A.

If A responds to the query and is selected to upload a file f, $S_A = price(f)$. Otherwise, $S_A = 0$. Let Q be the set of files that A owns that are possible responses to RTR_f ($|Q| > 1$ in the case of approximate matches). We assume that if A can respond to the query, then it will. Let $p_A(r)$ denote the probability that A's response r is picked for upload, given that A responded to the query with response r. The expected value of S_A is then:

$$E(S_A) = \sum_{r \in Q} p_A(r) \cdot price(r) \quad (4)$$

Let I_N be the indicator variable denoting whether neighbor N buys the RTR from A ($I_N = 1$), or not ($I_N = 0$). Note that the price at which the peer sells the RTR may be different from the price at which it bought the RTR, and it may differ on a per-neighbor basis. Assuming a peer N pays the expected value $E(\mathit{RTR}_f, N)$ for *RTR*$_f$, the income R_A generated by reselling the RTR is:

$$E(R_A) = \sum_{N \in nb} E(I_N | E(\mathit{RTR}_f, N)) \cdot E(\mathit{RTR}_f, N) \quad (5)$$

where nb denotes the set of neighbors of A.

Combining equations 3, 4 and 5, we can get the following formula for the cost of an RTR:

$$\begin{aligned} E(N_A) &= E(S_A) + E(R_A) \\ &= I_A \cdot p_A \cdot price(f) + \sum_{N \in nb} E(I_N | E(\mathit{RTR}_f, N)) \\ &\quad \cdot \; E(\mathit{RTR}_f, N) \end{aligned} \quad (6)$$

3.3 Peer Interaction

One of the main decisions a peer makes is with whom to interact, and in what manner. To model these decisions, we define the notion of a *profile*, and rules by which a peer uses profiles to guide their decisions.

Definition. The *profile* of a peer is defined on a pair-wise basis, and summarizes the interests of that peer. A profile of peer B compiled by peer A consists of the following:

- Whether peer B has downloaded from peer A.
- Peer B's past queries.
- The *profitability* of B, if B is a neighbor.

To determine the profitability of neighbor B, peer A keeps track of how much money he has made through interactions with B over time (e.g., through selling B RTRs, uploading files to B, etc.), as well as how much money he has spent on B (e.g., the cost of RTRs bought from B). Profitability is then the difference between the money made and spent with B. In addition, peer A may factor in the cost of resources consumed by that neighbor. For example, if B constantly spams A with RTRs that do not match A's filters, then the profitability of B decreases. Note that profiles are cheap to maintain as they are a simple history of past interactions. In addition, a peer need only keep a profile of each of its neighbors, and then profiles of just a handful of other peers, for connection purposes, described next.

Usage. Profiles are used to determine with whom connections should be made and broken. If the profile of a neighbor shows it to be unprofitable over a period of time, the peer shall drop that neighbor. When a connection is broken, typically, a new neighbor is chosen. Our model for choosing a new neighbor is as follows: A peer P connects to peer Q if Q has submitted queries for files that P owns, with preference given to Q if Q has bought a file from P. The rationale behind this rule is that peer Q is interested in the type of content that P shares, therefore P can increase his chances of receiving relevant RTRs if he connects to Q. The flip side is that P and Q are more likely to be in competition with each other, and selling each other RTRs may hurt their own chances of being chosen for an upload; however, we will see later that the benefits outweigh the costs. Overhead of connecting and disconnecting is low, given that peers only evaluate the quality of their neighbors periodically.

In the remainder of this paper, we will refer to the above connection strategies as the *adaptivity* model, since peers are adapting to the profitability of their surrounding peers. This concept is similar to existing work (e.g., [16]) where peers adapt according to the semantic similarity of content. However, we will see in Section 5.1 that such straightforward adaptivity without the RTR protocol results in very bad performance, due to competition. We also note that in some cases, a peer is unable to select with whom to interact – e.g., in an ad-hoc network where interactions are only possible with peers that are geographically close. We study this scenario as well in Section 5.1.

3.4 Cheating Peers

A classic problem in game-theory is to incentivize peers to act truthfully. In this section, we present an analysis of how peers might cheat under the current RTR protocol, and how to provide incentives against such cheating.

3.4.1 Bogus RTRs

A *bogus RTR* is an RTR for which the originating peer does not intend to buy a file. Selfish peers generate and sell bogus RTRs in order to make a profit off of other peers who believe the RTR is genuine and may lead to an upload. A certain number of "bogus" RTRs are to be expected. For example, a user may submit several queries before deciding what to download, without the intent of making money off of the RTRs. In general, however, bogus RTRs are harmful to other peers, and is inefficient overall.

We use two techniques to combat bogus RTRs, both of which punish peers *over time* (i.e., in a repeated-game setting). Hence, in the following discussion, we assume that it is not easy for peers to create new identities. Such an assumption is reasonable in a system that deals with currency – peers' identities must be tied to a real-world entity before they can be trusted to make a payment.

Adaptivity. In a scenario in which peers may choose their neighbors for interaction, peers already have an incentive to submit valid RTRs. From the adaptivity model in Section 3.3, where unprofitable nodes are disconnected from, we know that cheating peers will be pushed to the "edge" of the network with few neighbors. At the edge of the network, a peer receives both fewer results for its real queries (if it has any), and fewer RTRs from other good peers that it may respond to (assuming it is also trying to make a profit by selling files). We will see in Section 5.3 that in the end, such "edge" peers do not make as much profit as good peers that remain in the core of the network.

File-Buyer Reputation. A peer's *file-buyer reputation* represents how consistent a peer is in sending out RTRs and then actually buying a file. We may use a reputation mechanism such as [10] to track reputations, and protect against malicious peers "inflating" each others' scores. If peer Q downloads from P, then P knows that Q is a good file buyer; therefore, it will increase its opinion of Q by some amount s. Otherwise, P will decrease its opinion of Q by a small amount r, which may depend on how close or rare the match is. Over time, those peers that buy many files will have a good buyer reputation, and those who never buy files will have a poor buyer reputation.

Reputation is then used to affect two things: the estimated value of the peer's RTRs, and the likelihood the peer is chosen by other peers to *sell* files. Under the RTR protocol, the expected value of an RTR is multiplied by the probability that the originator of the RTR will indeed buy the file it is searching for, which is estimated by the originator's file-buyer reputation. If a peer's file-buyer reputation is low enough, then the profit that peer can make from selling bogus RTRs will not be worth the effort. In addition, when a good peer P submits a query and selects a peer for download, it will bias its selection towards those peers with good file-buyer reputations. By reciprocating good faith, those peers that contribute to the economy by buying items will also profit in the end, which we will show in Section 5.3.

3.4.2 Reporting Filters

Specifying filters may make a peer a target for higher prices; hence, peers have an apparent incentive to not specify their filters. However, filters are desirable because, as we will show in Section 5.1, they can provide the basis for intelligent routing, improving the overall efficiency of the network. Due to space limitations, we defer a discussion of this issue to [20]. In summary, we find that for certain useful classes of filters (such as categories), incentives to adhere to a standard truthfully exist naturally, due to weak equilibria.

4 Simulation Model

To evaluate our protocol, we simulate a P2P file-sharing application operating the RTR protocol. In this section we describe our simulator, which we then use to experimentally analyze the protocol in Section 5.

4.1 Query and Content Model

Each file has a unique identifier, as well as a *category*. One may think of a file's category as being, for example, the genre of music of the file (rock, pop, classical, etc.), or as the artist of the file. Queries are for specific files (i.e., they specify the unique identifier of a file). An *exact match* for a query is a response for the file specified in the query. An *approximate match* for a query is a response for a file that is in the same category, but has a different identifier. Approximate matches have some small probability of being uploaded, so a peer will respond to a query with all exact and approximate matches found in its library.

When a peer submits a query, it will typically receive multiple responses, from which it must select one to download. We assume that a peer's probability of downloading any file is independent of what responses it receives, with the exception of the case where no responses are received. When a peer chooses to download, we assume each response has a relative likelihood of being selected. All exact matches have a relative likelihood, or "score," of s_e. All approximate matches have a relative likelihood of s_a. The peer then rolls a weighted die to determine which file will be selected. The probability $p_{choose}(r, M)$ that a given response r will be selected from a set of responses M is:

$$p_{choose}(r, M) = p_d \cdot \frac{score(r)}{\sum_{r' \in M} score(r')} \tag{7}$$

Where $score(r)$ is the score of match r (s_a or s_e), M is the set of all matches received for a query, and p_d is the probability that the peer will download any file (e.g., if the peer submits bogus queries, then $p_d < 1$). Only one file is selected per query.

4.2 Behavior Modeling

In this section, we describe how we model the behavior of a peer, incorporating such decisions as selecting filters, setting prices, etc. Throughout our discussion, we will pinpoint the variables that define a peer's behavior.

We will make some simplifying assumptions in the model below. For a first cut, let us assume that peers have a general idea of how "popular" each file or category is. Future work can be done on estimating popularity (e.g., through past experience). Popularity of a category c, $catpop_c$, is defined by the the probability that a randomly chosen file among all file instances in the network has the category c. Similarly, popularity of a file f, $filepop_f$, is the probability that a randomly chosen file is an instance of file f. Let us also assume that all files have the same price, and that each peer knows how many files his neighbor has, and roughly how many files N_{files} exist in the entire network.

Filters. Each peer maintains a set of filters that tells other peers what RTRs they are interested in. For simplicity, we pre-define three types of filters: (1) *No filters*: No information is revealed. (2) *Category*: The list of categories describing files in a peer's library. (3) *Exact Files*: The list of files in a peer's library. The variable f_{filter} is a per-peer variable that denotes which filter level a peer chooses.

Buying and Selling RTRs. Three main decisions must be made when peer A wishes to sell an RTR to neighbor B. First, peer A must decide whether to sell the RTR to B. Second, if peer A does decide to sell the RTR, it must select a price. Finally, when peer B receives the offer, it must evaluate the value of the RTR to itself, and whether it wishes to purchase the RTR at the given price.

```
SellRTR(RTR $r$, Peer $N$)
 1: if (not matchesFilter(r, N)) then
 2:    return FALSE;
 3: if ($r.age < threshhold_{age}$) then
 4:    return FALSE;
 5: if ($r.reputation < threshhold_{rep}$) then
 6:    return FALSE;
 7: return TRUE;

PriceRTR(RTR $r$, Peer $N$)
 1: return uploadIncome($N, r$) $\cdot f_{greed}$

BuyRTR(RTR $r$)
  // First calculate $E(R_A)$ (Equation 5)
 1: $E_R$ = 0;
 2: for each neighbor $N$ do
 3:    $E_R$ += SellRTR($r$, $N$) $\cdot$ PriceRTR($r$, $N$) $\cdot p_{nodup}(r)$
    // self is a reference to the calling peer
 4: $E_S$ = uploadIncome(self, $r$)
 5: value = $E_S + E_R$
 6: return (value $> r.price \cdot f_{bargain}$)

uploadIncome(Peer $N$, RTR $r$)
  // Calculates $E(S_A)$ (Equation 4)
 1: $Q$ = set of all matches in $N$'s library
 2: $M$ = (estimated) set of all matches returned for $r$
 3: $E_S$ = 0;
 4: for each match $m$ in $Q$ do
 5:    $E_S$ += $p_{choose}(m, M) \cdot price(m)$
 6: return ($E_S$)
```

Function *SellRTR* (shown above) takes as input an RTR and a peer, and returns TRUE iff the calling peer should sell the RTR to this neighbor. The algorithm is very simple: if the RTR passes all content and quality filters, then sell the RTR. For simplicity, we assume a universal standard for RTR quality: all peers use the same thresholds for reputation and age. The function *matchesFilter* returns true iff the RTR matches the content of the neighbors at the filtering level chosen by the neighbor.

Given an RTR offer, the *BuyRTR* function returns true if the peer should purchase the RTR represented in the offer. A peer should purchase an RTR if the price is below the *expected value*, estimated according to the model in Section 3.2. The term $p_{nodup}(r)$ represents the probability that a neighbor has not already seen RTR r, which depends on the age of the RTR. Variable $f_{bargain}$ reflects the "bargain hunting" tendencies of a peer. A peer will only buy an RTR at a price that is a factor of $f_{bargain}$ *lower* than the expected value. The higher the "greed level" specified by the user, the larger the value used for $f_{bargain}$ by the peer.

The function *uploadIncome*(Peer N, RTR r) calculates the expected income from uploading a file for a given peer and RTR (Equation 4). This value is simply the sum of the expected incomes for each response m in N's library to RTR r. Note that in order to estimate $p_{choose}(r, M)$, we must estimate M – how many other files the queryer will receive. Due to space limitations, we defer a discussion of parameter estimation to [20].

Finally, the function *PriceRTR* returns the price for an RTR that the calling peer should set for a given neighbor. Again, the value of the RTR depends on the likelihood that the neighbor will be selected for download, and on the likelihood that the neighbor can resell the RTR to its own neighbors. The first factor can be estimated via a call to *uploadIncome*. The second factor we fold into an overall price-adjusting factor f_{greed}. Each peer sets f_{greed} according to how much they wish to "inflate" the price of an RTR – a "risky" action that may yield higher gains, but may also result in loss due to fewer sold RTRs. A peer may set $f_{greed} = 1$ conservatively, to price the RTR according to the known upload value, or $f_{greed} < 1$, to reflect the uncertainty in estimation, or $f_{greed} > 1$, to reflect the additional resell value of the RTR to its neighbor.

In summary, each peer has a vector of variables that define its behavior in the context of the model provided above. We list these parameters in Table 2. Note that we express the p_{nodup} variable as an array, where $p_{nodup}[i]$ is the probability that a neighbor has not yet seen an RTR that has already been passed through i hops. Parameter f_{return} is described in [20]. In Table 2, we present three default configurations: RTR_{nf}, RTR_{cf} and RTR_{ef}.

4.3 Metrics

We use three main metrics to measure the *overall performance* characteristics of the network, each of which are averaged across all queries within a run.

Name	RTR_{nf}	RTR_{cf}	RTR_{ef}	Description
f_{filter}	1	2	3	Filter level (1 = no filter, 2 = category filter, 3 = file filter)
f_{greed}	.5	“	“	Price adjustment in *PriceRTR*. Accounts for resell value, and estimation uncertainty
$f_{bargain}$	1	“	“	“Inflation” of RTR value for bargain-hunting peers
p_{nodup}[]	[1,.6,.4,0...]	“	“	Probability that a neighbor has not yet seen this RTR
f_{return}	.1	“	“	The fraction of all possible results in the network that will be returned for a query

Table 2. Behavior Model Variables

Average Score/Query: The “score” of a query is a weighted sum of the scores of responses received for that query. Recall that the score of exact and approximate matches are s_e and s_a, respectively. By default, we set $s_e = 1$ and $s_a = .01$; however, unless otherwise noted, the relative values of this metric in our performance comparisons are insensitive to these values.
Average Peers Processed/Query: the average number of peers that process a query. A peer processes a query if it has an exact or approximate answer.
# Messages/Query: the total number of offer, purchase request and RTR messages passed per query.

Sometimes, we may refer to the **quality-to-cost** ratio of a network, which we define to be the ratio of the average score per query to the average number of messages per query.

Peers begin the simulation with a fixed number of currency units, which are then used to buy RTRs, and are gained by selling RTRs and uploading files. Again, we assume the existence of an efficient P2P micropayment system, such as [21]. Individual peer performance is measured mainly by a peer’s money **balance** at the end of a run. We note that it would be unfair to call a peer “unprofitable” if it buys files that it desires; therefore, we assume the funds used to *purchase files* come from a separate source.

5 Results

In our experiments, we attempt to answer three important questions regarding the RTR protocol: (1) In which scenarios is the RTR protocol most needed, and how useful is it in these scenarios? (2) Which configurations of the protocol (e.g., parameter values) are most beneficial to the network as a whole, or to individual peers? (3) How well can the RTR protocol withstand the bogus RTR attack? We address each question in the following sections.

Our experiments are run using the Query Cycle Simulator [14]. To make our simulations tractable we limit the number of peers to 1000 (arranged in a power-law topology with average outdegree of 4), but we have experimented with larger networks, and our results continue to hold. Peers store an average of 100 files belonging to 2 out of 20 possible categories. See [20] for full parameter details. All figures in the following sections reflect results averaged over multiple runs. Unless explicitly shown, variance is small.

5.1 Overall Performance

First, let us revisit our motivating example in Section 2. As before, Table 1 shows us the performance of all, competitive and no-forward behaviors, as well as results for the RTR_{ef} configuration. Recall that due to non-cooperation in an competitive environment, quality of results degraded dramatically compared to the cooperative all-forward scenario – by a factor of 8 under competitive-forward, and by a factor of 17 under no-forward. Our goal for the RTR protocol was to achieve the same high quality of results and messaging efficiency as seen when peers are not competitive – i.e., all-forward. From Table 1, we see that the RTR protocol achieves its goal remarkably well. In particular, it has the same quality of results *and* messaging overhead as all-forward – a surprising result given the exchange of messages required by the RTR protocol. It is clear, then, that in a competitive environment, for the given example, the RTR protocol is necessary to ensure quality search results.

In this section we provide a more thorough investigation of the overall performance of the RTR protocol. Our goals in this section are (1) to illustrate, via experiments, the magnitude of the non-cooperation problem in different scenarios (if the magnitude is small, competition is not a problem), and (2) to show how RTRs can address non-cooperation.

We have identified four factors that affect the need for a solution to non-cooperation: (1) whether peers adapt (as described in Section 3.3) (2) the *interest overlap* between peers (without adaptivity), (3) the number of neighbors per peer, and (4) as mentioned earlier, whether peers forward *competitively*, or not at all. Due to space limitations, we can only present the first factor here. We strongly encourage readers to see our extended report [20] for additional results. In summary, we show how competition can become a serious problem in many realistic scenarios, and how the RTR protocol effectively address non-cooperation in these cases – reducing messaging overhead while maintaining excellent quality of results.

Adaptivity. Under our adaptivity model described in Section 3.3, not only are individual utilities maximized under the RTR protocol, but the topology also evolves into a more efficient one in which “clusters” are formed around common interests. Figures 2 and 3 show the performance of all, competitive and no-forward networks, as well as a network running the RTR protocol under our default configurations in Table 2. For each type of network, we show performance, measured by average score and messaging overhead, both with and without adaptivity. As expected, in the cooperative all-forward scenario, we see that adaptivity decreases overhead by over 24% (Figure 3), while increasing average score by over 69% (Figure 2).

However, the quality of results of competitive-forward

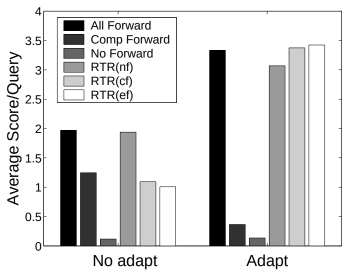

Figure 2. Competition becomes a greater problem with adaptive topologies

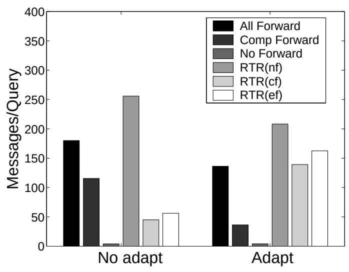

Figure 3. Messaging overhead decreases with adaptive topologies

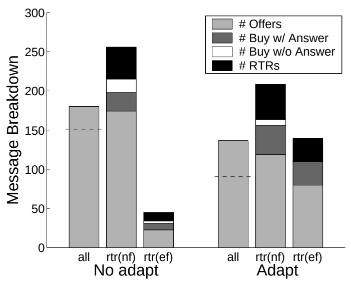

Figure 4. Breakdown of messaging overhead

drops sharply with adaptivity, to just 11% of the score under all-forward, and 30% of the score under competitive-forward in the non-adaptive scenario (Figure 2). Because adaptivity causes peers to be surrounded by other peers with similar interests, it only *exacerbates* the competition problem. Therefore, results in other adaptive topology studies (e.g., [16]) do not hold in the non-cooperative context.

Fortunately, under both the adaptive and non-adaptive scenarios, we can use the RTR protocol to maintain excellent quality of results at a very reasonable cost. In Figure 2, with no adaptivity, the RTR_{nf} configuration achieves 100% of the score possible under all-forward, at a 42% increase in messaging overhead. Such an overhead is reasonable given that for every peer that actually buys the query, three messages must be exchanged instead of one. Furthermore, notice that the RTR_{ef} and RTR_{cf} configurations, while not achieving as good quality of results, have a better quality-cost ratio than the all and competitive-forward cases. RTR_{cf} achieves almost 56% of the score of competitive-forward, at just 25% of the messaging cost. Under the adaptive scenario, the strengths of the RTR protocol become even more apparent. For example, the RTR_{cf} configuration actually achieves better quality of results than all-forward, at roughly the same cost.

Breakdown of Messaging Overhead. The RTR protocol can achieve such good quality-cost ratios, despite the need for 3 messages per bought query, due to efficient routing of queries to those peers who can provide answers. Figure 4 shows us the breakdown of message types for three configurations: all-forward, RTR_{nf}, and RTR_{ef}. We show all configurations with and without adaptivity. Messages are divided into four categories: offer messages, purchase request messages from peers that have answers to the query, purchase request messages from peers without answers, and RTR messages (see Figure 1). Under the all-forward configuration, there is only one type of message – queries. In the figure, for all-forward, we draw a dotted line such that the portion *above* the line denotes useful queries (i.e., sent to a peer for the first time, and the peer can answer the query).

From Figure 4, we make three important observations. First, looking at all-forward, a large number of query messages are not useful. For example, in the no-adaptivity case, less than one sixth of all query messages are useful. Useful queries are the only messages that contribute to the quality of results returned for queries; therefore, if the RTR protocol can intelligently route queries only to peers that can answer them, then the protocol can decrease cost while maintaining high quality.

Second, recall that each query that is purchased results in an additional overhead of 2 messages per query. Since most purchased queries are useful ones (i.e., the purchaser can answer the query), and since the relative number of useful queries is low, the overhead of the RTR protocol is also low. For example, consider the RTR_{nf} configuration without adaptivity in Figure 4. Roughly 1 out of 4 queries (or offer messages) are purchased, so additional overhead is about 42% the cost of forwarding queries, rather than 200%.

Finally, and most importantly, we observe that the RTR protocol can indeed reduce the fraction of query messages that are useless. Consider the RTR_{ef} configuration with no adaptivity. Roughly 1 out of 2 query messages are useful, compared to 1 out of 6 under all-forward. The difference between RTR_{nf} and RTR_{ef} is that RTR_{nf} uses no filters, and RTR_{ef} uses exact filters. Hence, we see that the presence of filters allows us to intelligently route messages.

The problem with RTR_{ef} without adaptivity is that the absolute number of useful query messages is *less* than under all-forward or RTR_{nf}, thus resulting in worse quality of results. Under RTR_{ef} query messages can be prematurely dropped if none of a peer's neighbors have filters that match the query. With adaptivity, however, a peer is clustered with many neighbors with the same interests. Thus it is unlikely for a query to be dropped before it has reached most peers that are able to answer it. Indeed, we see in Figure 4 that with adaptivity, RTR_{ef} achieves the same number of useful query messages as all-forward and RTR_{nf}. Furthermore, because RTR_{ef} is still able to intelligently route messages within these clusters via filters, it results in much fewer useless messages generated under all-forward.

5.2 Parameter Selection

In a system where peers are autonomous, peers may select their own policies. In this section, we study how peers will choose parameter values, and the effect these choices have on overall network behavior. In particular, we focus on f_{filter}, $f_{bargain}$ and f_{greed}. All remaining parameters

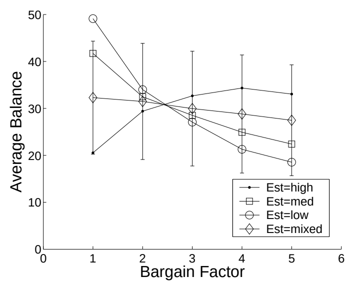

Figure 5. Individual balance for various values of $f_{bargain}$

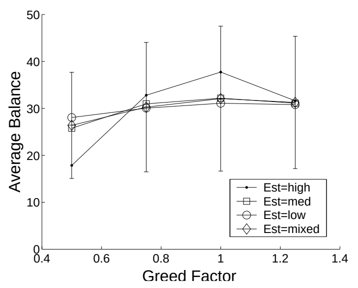

Figure 6. Individual balance for various values of f_{greed}

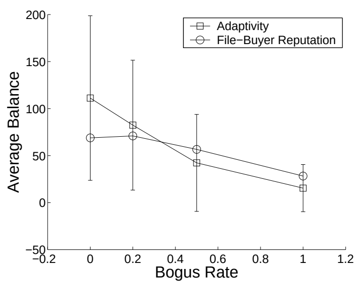

Figure 7. Individual peer balances using adaptivity

are estimations of system-wide values; therefore the rational policy is always to estimate these as well as possible.

Individual Effect. A peer's policy is largely reflected through $f_{bargain}$ and f_{greed}, which control the price at which an RTR is bought and sold, respectively, relative to the expected value. Unlike f_{filter}, for which incentives exist for all peers to adhere to the same value (Section 3.4.2), $f_{bargain}$ and f_{greed} may be freely chosen by each peer, depending on preferences stated by the user.

Figure 5 shows an analysis of the impact of $f_{bargain}$ on peer balance. Each curve represents an experiment where each peer chooses its own value for $f_{bargain}$ (out of a few choices, shown along the x-axis), which it keeps for the entire simulation. Remaining parameters are assigned default values from RTR_{cf}. At the end of simulation, we calculate average ending balance of peers grouped by $f_{bargain}$ value. Because the impact of $f_{bargain}$ depends on the quality of estimated values of RTRs, we ran four experiments (one per curve) where peers have uniformly high, medium, low, or mixed estimates.

In Figure 5 we find that standard deviation (shown by vertical bars for only the `Est = mixed` curve, for clarity) is large because of the heterogeneous nature of peers – e.g., peers with more files tend to have a larger balance, while peers with fewer files tend to have a smaller balance. This fact actually works to our advantage, as peers do not have an incentive to lie in order to increase income. Furthermore, looking only at mean values (e.g., peers with a mean number of files) we find that expected income is maximized by dealing fairly, rather than bargain hunting, especially when estimates are low to medium.[6] Thus, rational peers are likely to accept the default $f_{bargain}$=1, as choosing otherwise would not benefit the peer.

Figure 6 shows a similar experiment as Figure 5, but where peers only vary in f_{greed}. Here we see that f_{greed} has relatively little impact on peer balance. However, in terms of mean values, increasing f_{greed} presents a tradeoff between higher income per sold RTR, but fewer RTRs sold, resulting in a maximum at f_{greed}=1. Hence, rational peers are most likely to select $f_{greed} = 1$.

[6]When estimates are uniformly high, expected income is not actually maximized by increasing $f_bargain$, but to adjust f_{return} or p_{nodup} to reflect a more realistic model.

Overall Impact. In experiments omitted due to lack of space (see our extended report [20]), we find that overall system performance (cost and quality of results) is maximized with f_{greed} = 1 and $f_{bargain}$ = 1. In addition, we find that category filters are most effective in minimizing message overhead.

Fortunately, we found earlier that $f_{greed} = 1$ and $f_{bargain} = 1$ result in highest individual utility. Furthermore, natural incentives exist for truthful reporting of category filters (Section 3.4.2). Therefore, we find that under the RTR protocol, peers will indeed tend to choose behavior that results in desired overall outcomes.

5.3 Handling Bogus RTRs

In Section 3.4, we discussed two techniques to combat bogus RTR "attacks" on the RTR protocol: adaptivity and file-buyer reputation. Figure 7 shows us the individual balances of peers that issue bogus queries at varying rates, when adaptivity only, and file-buyer reputation only (with *no* adaptivity), are used. For these experiments, peers issue bogus RTRs at different rates: 0%, 20%, 50%, and 100%. A x% bogus rate means x% of a peer's queries are bogus. We show the average balance of all peers with a given bogus rate at the end of our simulations. Again, standard deviation (denoted by vertical bars, shown for the adaptivity curve only) is large b ecause peers are heterogenerous – e.g., some peers have many more files to sell than others.

In Figure 7, we can still see that both techniques are effective in punishing bad behavior, with adaptivity more prominently rewarding good peers. For example, average balance of a good peer (with a bogus rate of 0%) is almost five times higher than the average balance of a completely cheating peer (with bogus rate of 100%) under adaptivity. Combining the two techniques results in an even larger gap between honest and cheating peers.

Adaptivity is effective because, as expected, it pushes misbehaving peers to the edge of the network. Although all peers start with the same number of neighbors on average, at the end of the simulation, peers with high bogus rates have much fewer neighbors than peers with low rates. Similarly, file-buyer reputation is effective because reputation scores are accurate: in our simulations, a peer's reputation was roughly linear with its bogus rate. However, observe in Figure 7 that while peers with extremely high bogus rates are punished, those peers with moderate bogus rates (e.g., 20%) have nearly as good performance as good peers that never submit bogus queries. We believe such an effect is actually *good*. Even good users will likely submit bogus queries occasionally – e.g., if they do not find a satisfactory answer to their query. Therefore, we want the punishment for bogus RTRs to be "light" until bogus rate becomes excessive (e.g., $> 50\%$).

6 Related Work

Our work is partially motivated by the field of *algorithmic mechanism design* (AMD) (e.g., [12, 13]). AMD has been successfully applied to issues such as optimal routing and resource allocation [3, 11]. Recent activity in the field of AMD has focused on P2P networks [4, 15], since the autonomous nature of peers makes proper incentives crucial. However, while we apply many ideas from AMD and classic game theory, our problem cannot be fully described by the existing tools. For example, there is no precise way of capturing the adaptive nature of the network topology.

Researchers in ad-hoc and anonymity-preserving networks have also looked at economic incentives for peers to forward messages, such as in [1, 5, 23]. In each of these networks, peers are paid to forward messages. The Sprite [23] system uses a centralized server that processes a receipt of *every single* message that is forwarded in the network. Tamper-proof hardware is required in [1] at each peer to ensure proper payment is made for each forward. Finally, reference [5] requires that a peer sending a message knows exactly who will forward the message to its destination, so that payments can be appropriately embedded and signed. While each of these solutions are appropriate for their specific contexts, we cannot use them for general economic content-discovery systems. For example, we cannot expect a peer trying to discover content to know a priori which peers will be needed to route the query. The RTR protocol differs from these approaches because it is designed specifically for use in *economic applications*. Hence, it can utilize the key fact that queries have *value* to peers.

Finally, there exist studies on incentives for many other aspects of P2P networks, such as sharing files (e.g, [7]), and answering queries (e.g., [18]). As we observed in Section 2, while many of these mechanisms can be modified to encourage message forwarding (e.g., to prevent *no-forward* behavior), they are unable to deter the subtle but harmful *competitive-forward* behavior.

7 Conclusion

In summary, we have shown that the *non-cooperation problem* presents a significant challenge to competitive P2P networks. We present one promising solution, the *RTR protocol*, that gives peers the incentive to cooperate in the operation of the network, even in the face of competition for providing services. We have shown how our protocol enjoys a higher quality-cost ratio than even the non-competitive scenario, by efficiently routing queries to peers that can provide good answers, and how the protocol is robust against cheating peers. In the future, would like to gain a better understanding of individual peer choices through the application of game theory, and also to consider solutions for non-cooperation over alternative search architectures, such as GUESS and DHTs.

References

[1] L. Buttyan and J. Hubaux. Stimulating cooperation in self-organizing modile ad hoc networks. In *ACM/Kluwer Mobile Networks and Applications*, 2003.

[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. ACM SIGCOMM*, 2003.

[3] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. In *Proc. SOSP*, 2002.

[4] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. 6th Intl. Workshop on discrete Algorithms and Methods for Mobile Computing and Communications*, 2002.

[5] D. Figueiredo, J. Shapiro, and D. Towsley. Using payments to promote cooperation in anonymity protocols. Technical report, University of Mass., Amherst, 2003.

[6] Gnutella website. http://gnutella.wego.com.

[7] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. In *Proc. ACM Conference on Electronic Commerce*, 2001.

[8] GUESS protocol specification. http://groups.yahoo.com/group/-the_gdf/files/Proposals/GUESS/guess_o1.txt.

[9] Apple - iTunes - Music Store website. http://apple.com/itunes/store.

[10] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proc. WWW*, 2003.

[11] J. Kurose, M. Schwartz, and Y. Yemini. A microeconomic approach to decentralized optimization of channel access policies in multiaccess networks. In *Proc. ICDCS*, 1985.

[12] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. ACM Symposium on Theory of Computing*, 1999.

[13] C. Papadimitriou. Algorithms, games, and the internet. In *Proc. ACM Symposium on Theory of Computing*, 2001.

[14] M. Schlosser, T. Condie, S. Kamvar, and H. Garcia-Molina. Simulating a file-sharing p2p network. In *First Workshop on Semantics in P2P and Grid Computing*, 2002.

[15] J. Shneidman and D. Parkes. Rationality and self-interest in peer to peer networks. In *second IPTPS*, March 2003.

[16] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer. In *INFOCOM*, 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. In *Proc. ACM SIGCOMM*, 2001.

[18] Q. Sun and H. Garcia-Molina. Slic: A selfish link-based incentive mechanism for unstructuresed p2p networks. In *Proc. ICDCS*, 2004.

[19] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overalays. In *Proc. ACM SIGCOMM*, 2003.

[20] B. Yang, T. Condie, S. Kamvar, and H. Garcia-Molina. Non-cooperation in competitive p2p networks. Extended technical report. Available at http://dbpubs.stanford.edu/pub/2004-29, 2004.

[21] B. Yang and H. Garcia-Molina. Ppay: Micropayments for peer-to-peer systems. In *Proc. CCS*, 2003.

[22] B. Yang, S. Kamvar, and H. Garcia-Molina. Addressing the non-cooperation problem in competitive p2p systems. In *Workshop on Economics and Peer-to-Peer Systems*, 2003.

[23] S. Zhong, J. Chen, and Y. Yang. Sprite: A simple, cheat-proof, credit-based system for model ad-hoc networks. In *Proc. INFOCOM*, 2003.