Non-Cooperation in Competitive P2P Networks*

Beverly Yang　Tyson Condie　Sepandar Kamvar　Hector Garcia-Molina

Abstract

Large-scale competitive P2P networks are threatened by the *non-cooperation problem*, where peers do not forward queries to potential competitors. While non-cooperation is not a problem in current P2P free file-sharing networks, it is likely to be a 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

Peer-to-peer networks have recently become a popular mode of sharing data across a widely distributed network consisting of many individual, autonomous peers. One well-known example are free-filesharing networks such as Gnutella and KaZaA. In file-sharing networks, the main costs of storage for files and bandwidth for transferring them are distributed across many peers, rather than being concentrated at a centralized server. Hence, file-sharing P2P networks can reach a scale otherwise unattainable by centralized systems. For example, the KaZaA network had roughly 4.5 million users sharing over 7 PB of data as of April 2003.

One important problem facing an operating P2P network is that of *incentives* for peers to cooperate. Because peers are autonomous and come from potentially competing organizations, we cannot always make the assumption that they will behave in the desired manner. Most existing work on incentives in P2P networks has focused on the *free-riding* problem: peers acting in their own best interest share no files, and hence, only a small fraction of altruistic users offer almost all the available content. However, free-riding is only an issue in the context of free file-sharing applications.

While peer-to-peer networks have risen to prominence due to the success of free file-sharing, 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. In an auction system, where the auction advertisement is analogous to a query and bids analogous to query responses, peers will want to submit bids. Even in a free file-sharing network where users share their original music or artwork, users have an incentive to share their files to increase publicity. In all these applications, 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 frameworks that rely on peers to forward queries (e.g., Gnutella [12], DHTs like [26], etc.),[1] 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.

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.

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 of avoiding and punishing cheating peers.
- We analyze (Section 5) the performance of the RTR protocol and show how it effectively addresses the

*A brief position paper on this topic appears earlier in [30].

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

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 because it has a naturally simple pricing model in which all files have the same value. Furthermore, 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 [26, 22] and GUESS [14]. The following discussion assumes the existence of an efficient micropayment scheme for P2P systems, such as that described in [29], and a public-key infrastructure.

2 Background and Motivation

The basic Gnutella search protocol 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.[2] 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. In a pay-per-transaction application, the service offered is the download of a file, and the querying peer will pay the selected peer(s) for each download transaction. The price per download may vary depending on the file. Further details of the Gnutella protocol can be found in [12]. Clearly, if peers do not forward queries, the search mechanism will fail.

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 the *no forward* model, a peer forwards no queries. Technically, a peer under purely rational behavior 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 [15] 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 if it 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.

	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

Table 1: All, competitive and no-forward behavior

Under our two models of non-cooperation, we would like to see the effect of non-cooperation on performance. In Table 1, we compare the performance of the Gnutella file-sharing network under the all, competitive and no-forward cases. A detailed description of the setup for this experiment can be found in Section 4. Here, we note that the experiment is over a sample network of 1000 peers, where each peer has a number of files across multiple categories. A query is for a specific file, but a peer will respond to the query if it contains any file in the same category. 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 4.4).

In Table 1, 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 fewer 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

[2] 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.

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. If a peer never receives any queries, then it can never provide its service to anyone. An analogous concept in real-life markets are companies that buy lists of emails or referrals from other companies, so that they have a new pool of potential customers.

Once a peer buys an RTR for a given query, it may do one or both of the following: (a) respond to the query and hope that it is chosen to upload its services, and (b) sell the RTR to other peers.[3] 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).

3.1 Basic Implementation

An RTR has the following format:

$$RTR = \{Q, ts, query\}_{SK_Q} \tag{1}$$

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.[4]

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

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

[3]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.

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

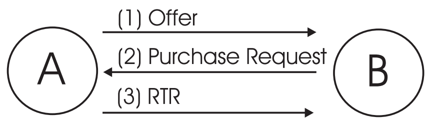

Figure 1: RTR Protocol

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, peer A may not indiscriminately offer any RTR to any of its neighbors. For example, if the offer is for an RTR that was issued a long time ago, or from a peer with a bad reputation, neighbor B may not wish to waste its bandwidth and processing resources to handle the offer. Instead, 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.

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. Or, a content filter may specify exactly what RTRs (i.e., for which exact files) will be purchased.[5] The semantics of content filters,

[5]Indeed, such a filter results in a *super-peer* relationship in which a

if specified, are that they comprehensively cover the types of RTRs a peer may respond to, or wishes to resell. Thus, if a peer "spams" a neighbor with irrelevant RTRs, that neighbor may punish the peer by disconnecting from it. Or, if a peer never purchases RTRs that fit its filters, his neighbor may disconnect and direct time and resources to a different, more profitable customer. We discuss further the issue of disconnecting from misbehaving neighbors in Section 3.3, and incentives for truthful filter reporting in Section 3.4.2.

3.2 Pricing

In this section, we discuss a simple pricing model for RTRs. We note that, 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 price their RTRs, and estimate the necessary parameters.

Model. An RTR has value for two reasons:

- The peer holding the RTR may respond to the query and be selected to upload the file (for which she is paid).
- The peer holding the RTR may resell the RTR to some or all of its neighbors.

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:

$$N_A = S_A + R_A \tag{3}$$

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:

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

query is only forwarded from super-peer to client if the client can definitely answer the query.

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:

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

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:

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

Parameter Estimation. The probability that A will be picked to upload the response r, $p_A(r)$, can be estimated in two simple ways. First, A can learn p_A over time by remembering how often it responded to a query for the same content, and how often it was chosen to upload. Second, we note that p_A can be estimated as the inverse frequency of file f across peers. Peer A can get an estimate of the frequency of its files through sampling, or perhaps statistics offered through a third party.

Estimating how many times an RTR can be resold, and at what price, can be aided by filters. If A knows, via content filter, that neighbor N owns responses to the query, then A can guess with fair confidence that N will purchase the RTR. Furthermore, the lower bound for the RTR should be $E(S_N)$. If A knows that N is a speculator who buys many RTRs, then A may also assume that N will buy RTR_f, but at a wholesale price. If the RTR does not fit N's filters, A can assume N will not purchase the RTR.

Note that by setting content filters, a peer N is "giving away" information about its preferences that can make it a target for higher prices. In Section 3.4.2, we will discuss incentives for peers to report filters truthfully.

In Section 4, we will describe an actual implementation of the above pricing model, including one concrete approach to parameter estimation.

3.3 Peer Interaction

One of the main decisions a peer makes is with whom to interact, and in what manner. Certain neighbors may be more profitable than others due to many factors, such as their trustworthiness, or simply an alignment of their demand with the peer's supply. To model how a peer makes these interaction 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 information:

- 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, either through selling B RTRs, uploading files as a result of an RTR bought from B, or uploading files directly to B. Peer A also keeps track of how much money he has spent on B, namely, the cost of RTRs bought from B, as well as the cost of RTRs bought from other neighbors for the sake of reselling to 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.

We note that with the given profile definition, a peer A does not need to perform any additional actions to determine a profile for peer B – all the information is a simple history of past interactions. In addition, note that a peer A does not need a keep a profile of every other peer in the network. Instead, A must 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. Peers have an incentive to disconnect from neighbors that are unprofitable. Hence, 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 of this connection 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.

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. In Appendix A we show adaptivity to be beneficial to individual utility. Hence, adaptivity follows as a reasonable model of peer decision-making. We leave as future work enhancements and verification of the adaptivity model.

We 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 5.1.1.

3.4 Cheating Peers

A classic problem in game-theory is to incentivize peers to act truthfully. If a peer can gain by lying or deceiving others, then under the assumption of rational peer behavior, it will do so. 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.

We point out the distinction between *rational* peers that cheat to maximize their utility, and *malicious* peers that desire to bring down the network. In this section we focus on rational but cheating behavior, which may not be as malignant to the system, but is likely to be much more widespread. Much existing work has studied techniques for countering malicious behavior (e.g., denial-of-service attacks [6], inauthentic file provision [16]).

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 makes inefficient use of the network overall.

A simple way to remove some incentive for bogus RTR attacks is to require that RTRs are free when they are forwarded by their owner. For example, if a peer Q wishes to submit a query, it must send the RTR for that query to its neighbors for free. Its neighbors may then resell the RTR for profit. Because a peer cannot make a profit off its own RTRs, it has no incentive to launch the bogus RTR attack.

However, a *collusion* of peers may still profit off the bogus RTR attack: a peer Q may send a bogus RTR to its friend R for free, and R may resell the RTR at a profit, which it shares with Q. Collusions are less likely than individual misbehavior, due to the overhead (both in real life and in network costs); however, a well-designed system still needs a way to prevent bogus RTRs.

We use two techniques to combat bogus RTRs: adaptivity, and *file buyer* reputation. Both of these techniques 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. Recall in the adaptivity model from Section 3.3 that peers disconnect from neighbors that are unprofitable, and connect to other peers that have been shown to have common interests. Because peers issuing bogus RTRs are not profitable, as their RTRs will never yield profit for the buyer, other neighbors will eventually disconnect from these cheating peers. In addition, because a peer P gives connection preference to those peers that have actually downloaded from P in the past, and a cheating peer M rarely or never downloads, M is not likely to be chosen to connect to good peers like P. Therefore, peers M get 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. The infrastructure for calculating and maintaining global reputations can be expensive (e.g., [16]). However, in a scenario where peers cannot choose their neighbors (e.g., in an ad-hoc network setting), or where the infrastructure for global reputations is already in place (e.g., to combat *malicious* peers as in [16]), the *file-buyer* reputation technique may be used with, or instead of, adaptivity. A peer's *file-buyer reputation* represents how consistent a peer is in sending out RTRs and then actually buying a file. This 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.

The file-buyer reputation of a peer is handled as follows: 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. 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.

In some cases, we may wish to differentiate between those peers that are new to the network, and have never had the opportunity yet to buy files, and those peers that frequently submit bogus queries. To do so, a peer P may decrease its opinion of Q by a small amount r each time it buys an RTR for a query submitted by Q. In this manner, the more frequently peer Q submits bogus queries, the lower its reputation will be, relative to a new peer.

Discussion. Both the file-buyer reputation and adaptivity techniques have strengths and weaknesses. The adaptivity technique is attractive because it is a natural result of local peer decisions that also improve the efficiency of the network. However, it does not protect against malicious peers – although selfish, *rational* peers are no longer able to make a profit off bogus RTRs, *malicious* peers who care only to destroy the network might still be able to flood peers within a small radius of itself. In contrast, with explicit file-buyer reputations, it becomes an easy matter to automatically drop any message with low reputation, thereby removing the risk of flooding. However, managing file-buyer reputations requires an infrastructure whose operations can be quite expensive [16]. In this paper, we will focus on the effectiveness of these techniques in making the bogus RTR attack unprofitable. As future work, we intend to see how existing techniques to prevent flooding attacks (e.g., neil's dos paper), other than global reputations, can be applied on top of the RTR protocol.

3.4.2 Reporting Filters

As we saw earlier, specifying filters makes peers 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 5.1.1, they can provide the basis for intelligent routing, improving the overall efficiency of the network. Here, we will discuss how to motivate peers to uniformly adhere to a type of filter, and then discuss incentives against deviant behavior, such as lying.

Selecting Filter Types. In a network running the RTR protocol, there are two possible approaches concerning peers' selection of filters:

1. All three types of filters are allowed, and peers may choose among them.

2. Only one type is recognized – i.e., the system is "initialized" with a client that only understands one type of filter.

Peers can be *initialized* to have a certain behavior if, for example, client software is distributed by a single source (which is typically the case in peer-to-peer systems). In order to produce a peer with deviant behavior, a user will need to rewrite the client software. Furthermore, P2P client software often encrypts communications between peers at the application level (e.g., [17]), making it even less likely (though still possible) to modify peer behavior.

The first approach is more complicated to analyze than the second. Here, we will focus on the second approach, because we believe it to be more likely to be used in practice due to its simplicity.

In the second approach, if a peer unilaterally decides to change its filter type, other peers will not understand this change, and will therefore be unable to interact. For example, if the system is initialized to handle category filters, then a peer who stops reporting filters will not receive any RTRs, and therefore will not earn any income. Such a scenario is a "weak equilibrium": Given that all other peers adhere to the protocol, a single peer's best course of action is also to adhere to the protocol. If enough peers (e.g., 50%) begin to use or recognize no filters, then *perhaps* using no filters will end up maximizing profit (although we have not run experiments to confirm this). Thus, we may assume that given the second approach, rational peers will adhere to the set filter type.

Lying. Given that all peers adhere to the same filter type, will they have an incentive to reveal their true filters? Consider the no filter case. Because peers are not asked to reveal any information about themselves, lying per se is not an issue. Consider category filters. If a peer unilaterally decides to not report one or more of its categories, it will not receive any RTRs (and thus, earn no income) for the unreported categories, while prices for RTRs in reported categories remain the same. Again, peers will not lie due to the weak equilibrium present.

Finally, consider the exact filters case. Here, the dominant strategy is indeed to lie. Consider a strategy in which a peer P reports a single file for each category it has. Peer P's filters will not preclude any of the RTRs that are relevant to it. Because P will receive most[6] of the RTRs it would have under truth-telling, at a lower price, its income increases.

Fortunately, we will see in Section 5.2.2 that in terms of overall behavior, category filters are just as effective as exact filters in network efficiency. Hence, from the protocol designer's perspective, we can safely eliminate exact filters from our design space. However, note that if exact filters were needed for some reason, we could put some mechanisms in place to discourage lying. As an example, we could have peers sign their responses to queries, and their filter reports. If a peer A notices discrepancies in the responses and reported filters from peer B – for example, if a response contains many files that are not reported in the filter specification – then B can shun peer A and share the "proof" to other nodes as well. We do not argue that such a mechanism with exact filters is preferable over a network using category or no filters; we only state that it is possible.

[6] When a peer specifies a file in a filter, its neighbors are more likely to sell it relevant RTRs, because they are more likely to buy such relevant RTRs at a higher price from their neighbors, given the high resell value.

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 as the specified file, 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 available responses M is:

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

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:

- *No filters*: No information is revealed.
- *Category*: The list of categories describing files in a peer's library.
- *Exact Files*: The list of files in a peer's library.

Recall that greater specificity results in higher prices, but also less traffic, and less obligation to buy RTRs that have a low expected yield. The variable f_{filter} is a per-peer variable that denotes which filter level a peer chooses. In Section 5.2, we will see how the choice of filter level affects the expected profit and risk a peer takes in purchasing RTRs. From these we can infer how a peer should select filter levels.

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.

Function *SellRTR* (shown above) takes as input an RTR and a peer (presumably a neighbor of the calling peer), and returns TRUE if the calling peer should sell the RTR to this neighbor, and FALSE otherwise. 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

SellRTR(RTR r, Peer N)
if (**not** matchesFilter(r, N)) **then**
  return FALSE;
if ($r.age < threshhold_{age}$) **then**
  return FALSE;
if ($r.reputation < threshhold_{rep}$) **then**
  return FALSE;
return TRUE;

PriceRTR(RTR r, Peer N)
return *uploadIncome*(N, r) $\cdot f_{adjust}$

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

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

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 a "fudge factor" probability that a neighbor has not already seen RTR r. As a result, p_{nodup} 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 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. We discuss the matter of estimation shortly.

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*, which requires an estimation of the neighbor's matches (discussed shortly)

Rather than estimating the resell value of the RTR to a neighbor, which would involve much guesswork, we simply use a price-adjusting factor f_{adjust}, a per-peer variable. If $f_{adjust} = 1$, then the calling peer only considers the first factor of income in the price, and assumes that $uploadIncome$ will accurately estimate this first factor. A peer can also set $f_{adjust} < 1$, to reflect the uncertainty in estimation, or $f_{adjust} > 1$, to reflect the additional resell value of the RTR to its neighbor.

Parameter Estimation. Let us first consider estimating the M parameter in Equation 7. To do this, a peer must guess the fraction f_{return} of all matching file instances that will be returned to the queryer as a response. Clearly, the choice of f_{return} will be determined by the peer's particular behavior model (e.g., how conservative a peer is). With this fraction, the peer can estimate the number of exact matches the queryer receives for a query for file f, $n_{exact} = filepop_f \cdot f_{return} \cdot N_{files}$, and the number of approximate matches similarly as $n_{approx} = catpop_c \cdot f_{return} \cdot N_{files}$, where f has category c. The denominator of the fraction in Equation 7 is therefore:

$$\sum_{r' \in M} score(r') = n_{exact} \cdot s_e + n_{approx} \cdot s_a \qquad (8)$$

Now let us consider estimating the number of exact and approximate matches a neighbor has (i.e., variable Q in *uploadIncome*) for a query on file f in category c. We use simple conditional probabilities. If no filters are specified, we assume the neighbor's content is a representative sample of the network. The expected number of exact matches he has is $filepop_f \cdot \#files$, and the expected number of approximate matches is $catpop_c \cdot \#files$, where "# files" is the number of files owned by the neighbor.

With category filters, we first estimate the number of files the neighbor owns in the file's category by assuming files in the neighbor's collection are distributed across categories according to the popularity of the category. Let us say we estimate n_c files in category c. Then the expected number of exact matches is $\frac{filepop_f}{catpop_c} \cdot n_c$ and the expected number of approximate matches is n_c.

Finally, with exact filters, we can determine exactly how many exact and approximate matches a neighbor has for the query.

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. In Table 2, we present three default configurations: RTR_{nf}, RTR_{cf} and RTR_{ef}.

4.3 Query Cycle Simulator

The results of our experiments were taken from simulations using the Query Cycle Simulator [24], which is only briefly described here due to lack of space. The simulation of peer interactions in a P2P network proceeds in *query cycles*. During each query cycle, a peer P may issue a query. All peers that receive this query are then calculated, (e.g., under the default Gnutella protocol, all peers within TTL hops of P will receive the query), and responses are generated according to the content model described in Section 4.1. Peer P then selects a download source among the nodes that responded and downloads the selected file.

The base settings for the simulator include the total number of content categories T_{cat}, the number of categories N_{cat} each peer is interested in, the files each peer owns, the distribution of files within a category, the total number of peers, and the query message TTL. The content category and file parameters are initialized based on distributions taken from [5, 23]. The total number of content categories is 20 by default, and each peer shares no more than 100 files in any given category. To limit the query message overhead, the TTL is set to 1/3 rd of the diameter of the network, determined experimentally to be 3.

At the beginning of a simulation, each peer begins with a fixed number of currency units. Peers use currency to buy RTRs, and gain currency by selling RTRs and uploading files. Again, we assume the existence of an efficient P2P micropayment system, such as [29]. Throughout the simulation, each peer keeps track of its *balance*. At the end of the simulation, those peers with higher balances have made a greater profit than those peers with lower balances. 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.

To make our simulations tractable we limit the number of peers to 1000, and run each simulation for 20 cycles (meaning each peer can submit up to 20 queries). We have experimented with longer runs and with more peers, and our conclusions continue to hold.

4.4 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.

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 com-

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_{adjust}	.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

parisons 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. Individual peer performance is measured mainly by a peer's money **balance** at the end of a run. A peer's goal is always to maximize balance.

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.

All figures in the following sections reflect results averaged over multiple runs. Unless explicitly shown, variance on results is small.

5.1 Overall Performance

First, let us revisit our motivating example in Section 2. Table 3 shows us the same example from before, but with results for the RTR_{ef} configuration as well. Recall from Section 2 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 3, we see that the RTR protocol achieves its goal remarkably well. In particular, it achieves 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 under a competitive environment, for the given example, the RTR protocol is necessary to ensure quality search results.

	Score	# Processed	# 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 3: A comparison of RTR protocol against all, competitive, and no forward

In this section we provide a more thorough investigation of the overall performance of the RTR protocol. In particular, our goal in this section is two-fold: (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. In this section, we will focus on the first 3 factors.

5.1.1 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 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

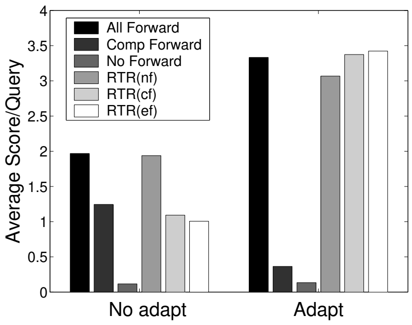

Figure 2: Competition becomes a greater problem with adaptive topologies

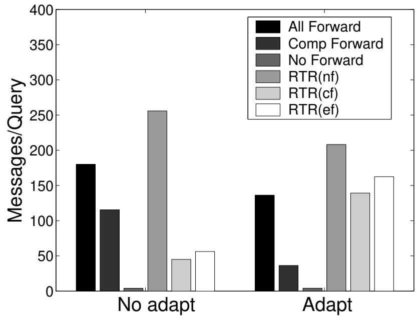

Figure 3: Messaging overhead decreases with adaptive topologies

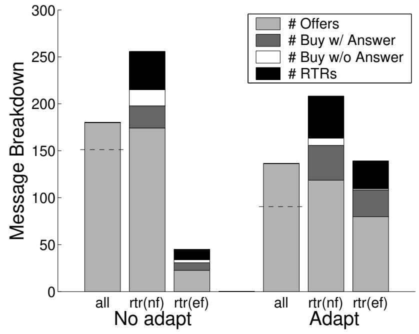

Figure 4: Breakdown of messaging overhead

with similar interests, it only *exacerbates* the competition problem. Therefore, results in other adaptive topology studies (e.g., [4]) 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 original 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. For example, 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 together 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.

5.1.2 Interest Overlap

Let us now consider the interest overlap between peers, in the case where there is *no adaptivity* (with adaptivity, competition is a problem regardless of interest overlap). Because peers always have an incentive to adapt under the RTR protocol, here we are considering a scenario in which adaptivity is not possible – e.g., an ad-hoc network. Recall from Section 4.3 that out of T_{cat} total content categories ($T_{cat} = 20$ by default), peers are interested in an average of N_{cat} categories, selected at the beginning of the simulation. Figures 5 and 6 show us the overall performance of the network in terms of score and messaging overhead, respectively, while we vary the number of category interests per peer, N_{cat}, along the x-axis. We show the performance of both all and competitive-forward, as well as the performance of RTR_{nf}.

As the average number of category interests increases, so does the likelihood that a peer will have interests similar to other peers (i.e., greater interest overlap), and therefore be in competition. Indeed, we see from Figure 5 that as the number of category interests per peer increases, the difference between competitive and all-forward increases dramatically. Scores under all-forward increase because more peers have more answers to a given query, whereas scores under competitive-forward decrease, because query messages are not being forwarded. When $N_{cat} = 16$, the average query score under competitive-forward is just 5.8% of the score under all-forward.

Fortunately, in Figure 5, we also see that the RTR protocol does an excellent job in maintaining high quality of results. For all values of N_{cat}, the performance of the RTR protocol closely mirrors all-forward. In terms of cost, Figure 6 shows that while the RTR protocol is more expensive than all-forward, it is still within reason for the general case. For example, in the default case where $N_{cat} = 2$, the RTR protocol is about 43% more expensive than all-forward in terms of messages sent.

5.1.3 Number of connections

In the competitive-forward scenario, a peer does not receive queries because competitive neighbors drop them. Intuitively, then, the greater the number of neighbors each peer has, the less likely it is for a query to be dropped prematurely. In the extreme case, if the topology is strongly connected, competitive-forward will have the same performance as all-forward.

In Figures 7 and 8, we show the results of several simulation runs in which the average number of neighbors varied across the runs. Because we could not control the number of neighbors directly, the average number of neighbors, shown along the x-axis, are not integers. For each number of average neighbors, these figures show us the performance of all-forward, competitive-forward and RTR_{ef}, measured in terms of average query score and messaging overhead, respectively.

Figure 7 confirms our intuition that competition becomes less of a problem as the number of connections increases. For example, when the average number of neighbors is roughly 8, competitive-forward has an average query score that is just 14% of all-forward and RTR_{ef}. However, when the average number of neighbors is 31, competitive-forward and all-forward have the same quality of results. In Figure 8, we see that competitive-forward also incurs less overhead than all-forward.

We do note, however, that as network size increases, more neighbors are required before competitive and all-forward have similar performance. When average neighbors is 16 in our simulations, all-forward is sending the query to every peer in the network; in a larger network, quality of results for all-forward will continue to improve with higher connectivity (> 16 connections). Therefore, it will also take higher connectivity for competitive-forward to "catch up" to all-forward. We also note that a network cannot always simply increase connectivity to overcome competition: it is generally not reasonable to allocate hundreds of open connections to a single application, and networks may face other real-world constraints on outdegree, such as proximity in an ad-hoc network.

Again, the RTR protocol does an excellent job at maintaining the same quality of results at a very low cost. For example, when average number of neighbors is 16, the RTR protocol results in almost the same quality of results as all-forward, but at just a fourth of the cost. Even when non-cooperation is no longer a problem (e.g., # neighbors is 31), the RTR protocol is still very useful in dramatically reducing messaging overhead by almost 90%. This surprising observation is again explained by the fact that the RTR protocol routes queries selectively, rather than generating many messages (and duplicates) by blind broadcast. In particular, almost the entire increase in cost for all-forward is accounted for by duplicate messages.

In conclusion, when peers adapt, interests overlap, or number of neighbors is limited, non-cooperation renders the network ineffective. We have shown the RTR protocol to effectively address non-cooperation in these cases, and to be very useful in all cases in reducing messaging overhead while maintaining excellent quality of results.

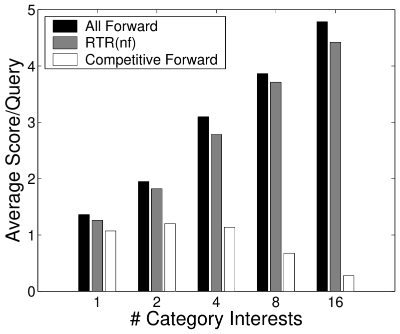

Figure 5: Competition becomes a greater problem when interest overlap increases

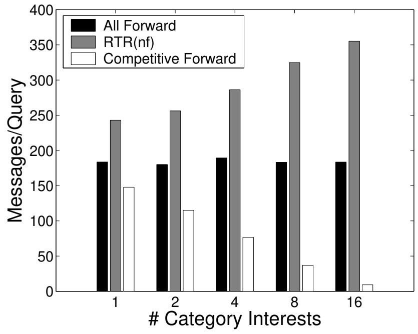

Figure 6: Messaging overhead

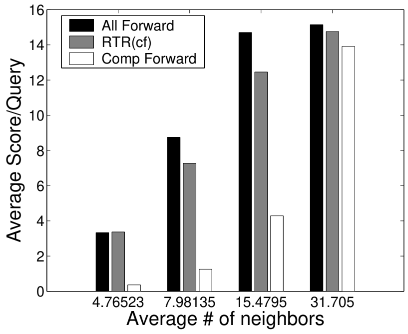

Figure 7: Competition becomes less of a problem with average neighbors increases

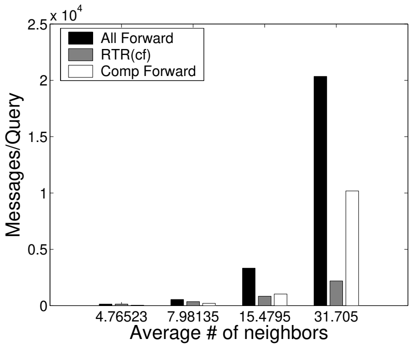

Figure 8: Messaging overhead is much smaller for RTR than for all-forward

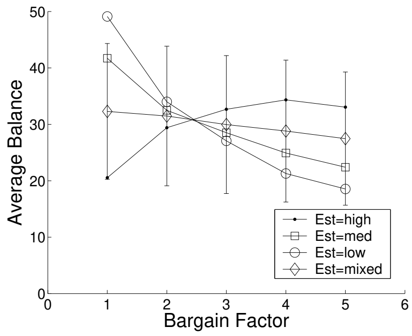

Figure 9: Individual balance for various values of $f_{bargain}$

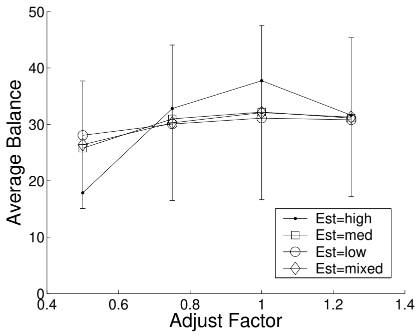

Figure 10: Individual balance for various values of f_{adjust}

5.2 Parameter Selection

In Section 4 we presented a behavior model with 5 parameters that define an individual policy: f_{filter}, f_{adjust}, $f_{bargain}$, p_{nodup}, and f_{return}. 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. We note that f_{return} and p_{nodup} are simple estimations of system-wide values, therefore a rational policy will always be to estimate these as well as possible. Therefore we focus on the remaining three parameters.

5.2.1 Individual Effect

A peer's policy is largely reflected through $f_{bargain}$ and f_{adjust}, which control the price at which an RTR is bought and sold, respectively, relative to the expected value. Unlike f_{filter}, which can be determined at design time (Section 3.4.2), $f_{bargain}$ and f_{adjust} may be freely chosen by each peer.

Figure 9 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. 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., some peers have many more files than others. Because the impact of $f_{bargain}$ depends largely 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. Estimates are influenced by choosing values of f_{return} above, at, or below true value.

Figure 9 shows us that, unless RTR value estimates are uniformly high, bargain-hunting is harmful. For medium and mixed estimations, many peers have a fairly accurate assessment of value; thus we find that expected income is maximized by dealing fairly, rather than hoping to "scoop" other naive peers. Furthermore, when RTRs are always underpriced (i.e., low estimates), bargain hunting makes even less sense than if they are well-priced.

Now consider when estimates are uniformly high. If a peer discovers that every one else's assessment is too high, the "correct" response is not to increase $f_{bargain}$ as Figure 9 might suggest, but rather to adjust f_{return} or p_{nodup} to reflect a more realistic model. In doing so, the peer still buys RTRs at a lower price (just as it would have if it had kept the incorrect model and just increased $f_{bargain}$). In addition, the peer will have a more accurate model with which to make informed decisions. Hence, as long as a rational peer believes his model to be reliable, he will choose $f_{bargain}$ = 1 to maximize income.

Figure 10 shows a similar experiment as Figure 9, but where peers only vary in f_{adjust}. Here we see that f_{adjust} has relatively little impact on peer balance (unless estimates are uniformly high) because increasing f_{adjust} presents a tradeoff between higher income per sold RTR, but fewer RTRs sold. For example, when estimations are uniformly high, peers with f_{adjust} = .5 sold an average of 1100 RTRs per simulation, while peers with $f_{adjust} = 1.25$ sold 630. This tradeoff does result in a slight maximum at $f_{adjust} = 1$. Hence, rational peers are most likely to select $f_{adjust} = 1$.

Discussion. For simplicity, the above analysis of f_{adjust} and $f_{bargain}$ assumed that each peer use a single parameter value for all decisions. However, in reality, a peer may use different values on a *per peer, per RTR* basis. For example, in a no-filter system, a peer B may charge neighbor A a higher price than A would likely pay if A did not own the file. If A buys the RTR, then B might conclude that A owns the file with some high probability. In return, a peer may also refuse to buy an RTR from a particular neighbor, even if the price is less than the expected value (i.e., act as a bargain-hunter), in order to fool its neighbor's "learning model" into charging lower prices in the future.

An analysis of peers' learning models over time, and their possible responses to fool these models, is outside the scope of this paper. We leave as future work a formal analysis of this interaction between f_{adjust} and $f_{bargain}$. For now, we just make the following high-level observation: a peer must be profitable to its neighbor, otherwise it will lose neighbors and be pushed to the edge of the network. To a certain degree a peer may "outsmart" its neighbors to maximize income. Over time, however, extreme behavior of either type – especially if the magnitude exceeds that of other peers in the system – will make the peer unprofitable to its neighbors. Thus, in the remainder of this paper we will consider the behaviors to have canceled each other out: Peers neither employ a learning model over neighbors' content, neither do they reject reasonably priced RTRs to fool neighbors' models.

5.2.2 Overall Impact

Let us now consider how parameters impact overall system performance. Ideally, we will find that the values chosen naturally by selfish peers will result in best overall performance as well.

First consider f_{adjust} and f_{filter}. In Figure 11 we see the outcome of 9 different experiments, each running the RTR protocol with different values of f_{filter} and f_{adjust}. For example, in the experiment where no filters are used and f_{adjust} = .5, all peers used these same values for f_{filter} and f_{adjust}, and default values for the rest.

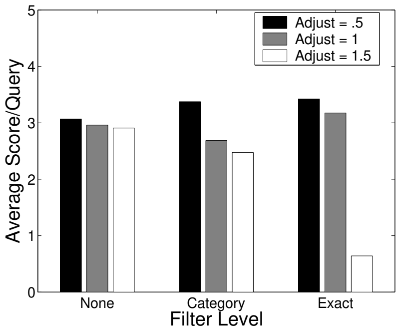

Figure 11: Average query scores for varying parameter values

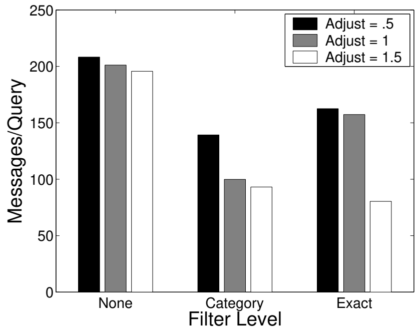

Figure 12: Messaging overhead for varying parameter values

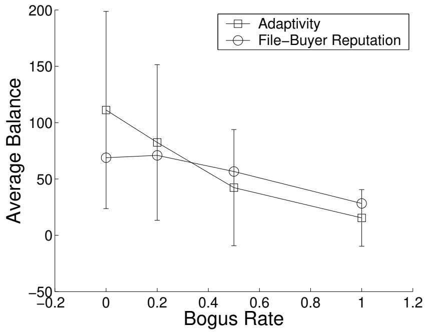

Figure 13: Individual peer balances using adaptivity

With regards to f_{filter}, Figure 11 confirms what we found earlier in Section 5.1.1: filters allow queries to be intelligently routed to those peers with answers; therefore message cost is greatly reduced. However, the results in Figure 11 are for the default case where peers adapt. We also saw in Section 5.1.1 that when peers cannot adapt, filters result in low quality of results; therefore null filters are more appropriate in the non-adaptive scenario. Filter levels can be set at design time by proper initialization.

We also find in Figure 11 that *price inflation* is harmful to system performance. When $f_{adjust} > 1$, peers charge more for an RTR than its estimated value, thereby "inflating" its price. When exact filters are used, price inflation ($f_{adjust} = 1.5$) results in just 19% the average query score achieved by $f_{adjust} = .5$. Note, however, that price inflation only comes into play when filters are used. Category and exact filters are sensitive to f_{adjust} because the more information available to peers when pricing RTRs (e.g., the content of their neighbors), the more accurately the price estimates the true value of the RTR to the neighbor. Thus, price inflation is more likely to cause neighbors to drop offers (due to price exceeding value), than if no filters are present.

With regards to $f_{bargain}$, we found low values of $f_{bargain}$ to result in best overall performance. In Figure 14, we see the average query score in the network as $f_{bargain}$ is varied. Each point in the figure represents a network in which all peers have the same value of $f_{bargain}$. From this figure, we see clearly that $f_{bargain} = 1$ has best overall effect. Larger values of $f_{bargain}$ are harmful because they stifle the economy: they reduce the number of peers who buy RTRs, and hence, the number of peers able to respond to queries. Furthermore, lower value for $f_{bargain}$ result in a better quality-cost ratio. While $f_{bargain} = 1$ results in average query score that is 8 times higher than $f_{bargain} = 5$, average messaging overhead is only four times higher (experiment not shown here).

Fortunately, we found in Section 5.2.1 that $f_{adjust} = 1$ and $f_{bargain} = 1$ result in highest utility. Therefore, we

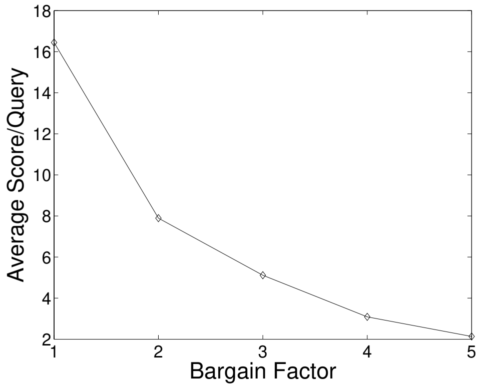

Figure 14: Average query scores for varying $f_{bargain}$ values

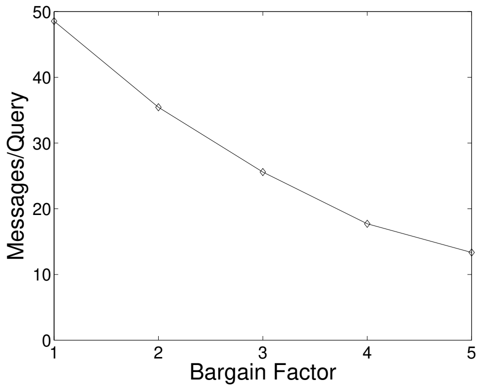

Figure 15: Average messaging overhead for varying $f_{bargain}$ values

see that under the RTR protocol, peers will tend to choose behavior that results in desired overall system 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. Here, we wish to analyze the effectiveness of these techniques. A technique is *effective* if peers can maximize income by minimizing the rate at which it issues bogus queries.

Figure 13 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, for clarity) is large because peers are heterogeneous – e.g., some peers have many more files to sell than others.

In Figure 13 we see that both adaptivity and file-buyer reputation are effective in rewarding good behavior and 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.

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. In addition, peers at the edge of the network receive lower scores for the valid queries they issue, due to weaker connectivity to the rest of the network. For example, peers with 0% bogus rate had an average of 11 neighbors each, and average individual query score of 5. Peers with 100% bogus rate had an average of 2 neighbors, and average score of 1.1.

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 13 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

Many areas in peer-to-peer content-discovery networks have been of interest to the systems and networking research communities. Examples include the design and analysis of structured overlays to provide efficient lookups with guarantees (e.g., [22, 26]); applications of DHTs such as information retrieval [28]; techniques for queries over unstructured overlays (e.g., [2, 3]) such as replication, adaptive topologies, and intelligent query forwarding; etc. While the solutions put forth in these references address critical problems, most cannot function in the face of non-cooperation. In this paper, we propose an initial solution to non-cooperation that simultaneously incorporates many of the advancements made in these recent results, specifically for unstructured overlays.

Our work is partially motivated by the emerging field of *algorithmic mechanism design* (AMD) (e.g., [7, 9, 19, 20, 21]). AMD has been successfully applied to issues such as optimal routing and resource allocation [8, 18]. Recent activity in the field of AMD has focused on P2P networks [10, 25], 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 providing economic incentives for peers to forward messages, such as in [1, 11, 32]. In each of these networks, peers are paid to forward messages. To ensure that peers are paid only as necessary, the Sprite [32] system uses a centralized server that processes a receipt of *every single* message that is forwarded in the network. In the the system proposed by [1], tamper-proof hardware is required at each peer to ensure proper payment is made for each forward. Finally, reference [11] requires that a peer sending a message knows exactly who will forward the message to its destination. Payments are then embedded in the message, and signed such that only the peer to whom the payment is made can redeem it. While each of these solutions are appropriate for their specific contexts, we cannot use them in the context of a general economic content-discovery system. 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, [13, 15]), and answering queries (e.g., [27]). 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. Because only a fraction of a peer's incoming messages are dropped under competitive-forward, these mechanisms cannot differentiate between peers who act competitively and peers who have, for example, one fewer neighbor than the average peer, as both will have a slightly lower output of queries.

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] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. ACM SIGCOMM*, 2003.

[4] T. Condie, S. Kamvar, and H. Garcia-Holina. Adaptive topologies in peer-to-peer systems, 2004. Under submission.

[5] A. Crespo and H. Garcia-Molina. Semantic overlay networks. Under submission, 2003.

[6] N. Daswani and H. Garcia-Molina. Query-flood dos attacks in gnutella. In *ACM Conference on Computer and Communications Security*, November 2002.

[7] P. Dutta. *Strategies and Games*. The MIT Press, 1999.

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

[9] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63, 21-41 2001.

[10] 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.

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

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

[13] 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.

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

[15] S Kamvar, M. Schlosser, and H. Garcia-Molina. Incentives for Combating Freeriding on P2P Networks. In *Proc. EURO-PAR*, 2003.

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

[17] KaZaA website. http://www.kazaa.com.

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

[19] N. Nisan. Algorithms for selfish agents. In *16th Annual Symposium on Theoretical Aspects of Computer Science*, 1999.

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

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

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, 2001.

[23] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of the Multimedia Computing and Networking*, January 2002.

[24] 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.

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

[26] 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.

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

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

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

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

[31] B. Yang, P. Vinograd, and H. Garcia-Molina. Evaluating guess and non-forwarding peer-to-peer search. In *Proc. ICDCS*, 2004.

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

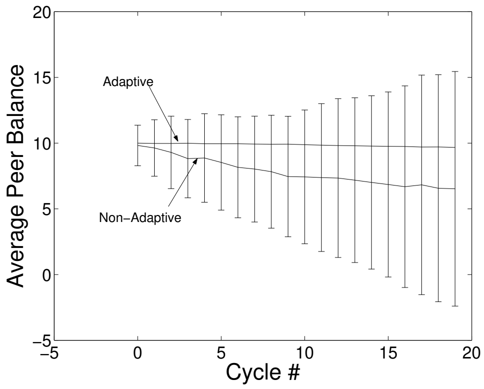

Figure 16: Adaptivity maximizes individual utility, when the majority of peers are adaptive

A Effective of Adaptivity on Individual Utility

In Section 3.3 we stated that adaptivity was beneficial to individual utility. As with the selection of filter levels, adaptivity results in a weak equilibrium, in which individual peers maximize utility by being adaptive, given that other peers are also adaptive. To illustrate this observation, we run an experiment in which 90% of all peers are adaptive, and 10% peers are not adaptive. In Figure 16, we show the average balance (which is the main utility metric) for each group of peers over time. As Figure 16 clearly shows, those peers that do not adapt have lower balance than those peers that do adapt. Furthermore, this difference in average balance increases over time. Therefore, if all peers are initialized to be adaptive, it is in each peer's best interest to remain adaptive.

We note that the standard deviation of balances in both groups is very large. For clarity, in Figure 16, we show only the standard deviation bars for one group. The reason deviation is so large is because peers are heterogeneous – some peers have many more files to share than others, and as a result, they gain more income by uploading files. If we view the balance of peers by "buckets" of number of files owned, deviation is expected to decrease significantly. In addition, we note that a clear trend is visible despite the large deviation, and that as time goes on, the projected trend will become increasingly significant.

Even when a relatively large fraction (e.g., 50%) of peers are non-adaptivite, adaptivity continues to be the best strategy, although the difference in balance between adaptive and non-adaptive peers is smaller. Only when a large majority of peers are non-adaptive does non-adaptivity become the best strategy.

B The GUESS Approach

Earlier we discussed why unstructured P2P networks such as Gnutella are more appropriate for non-cooperation than structured ones. However, one may also raise the question: why use any forwarding protocol at all? For example, why not use a protocol such as GUESS [14], in which a peer sends its queries directly to other peers?

Let us consider the GUESS protocol in further detail. Under GUESS, each peer maintains a large list of other peers' identities or addresses. When a peer submits a query, it will simply go down this list of peers, and send a query message directly to each one. In order to maintain this list, a peer relies on other peers to send it *pong* messages, which contain a small number of peer identities or addresses chosen by the peer sending the pong. For example, if peer P sends a pong message to peer Q, it will choose say 5 IP addresses from its list to Q. In reference [31], it is shown that intelligent selection of pong message content is crucial to the performance of the system.

While a GUESS-like approach would address the problem of non-cooperation with regards to query-forwarding, we note that it introduces a new non-cooperation problem: namely, that of maintaining peer lists via pong messages. Non-cooperating peers have no incentive to forward the identities or addresses of other peers in the network, because any peer could pose competition. As a result, a peer will know of only a few other peers in the network, and receive very few responses for its queries. Because it is not clear to us whether GUESS or Gnutella would be a better approach, and because Gnutella is much more widely used (GUESS has never been deployed), we choose to focus on a Gnutella-like protocol in this paper.