Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 52 (2008) 675–694

Computer Networks

www.elsevier.com/locate/comnet

Counteracting free riding in Peer-to-Peer networks [☆]

Murat Karakaya [*], İbrahim Körpeoğlu, Özgür Ulusoy

Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey

Received 11 January 2007; received in revised form 10 October 2007; accepted 4 November 2007
Available online 17 November 2007

Responsible Editor: Y.C. Hu

Abstract

The existence of a high degree of free riding is a serious threat to Peer-to-Peer (P2P) networks. In this paper, we propose a distributed framework to reduce the adverse effects of free riding on P2P networks. Our solution primarily focuses on locating free riders and taking actions against them. We propose a framework in which each peer monitors its neighbors, decides if they are free riders, and takes appropriate actions. Unlike other proposals against free riding, our framework does not require any permanent identification of peers or security infrastructures for maintaining a global reputation system. Our simulation results show that the framework can reduce the effects of free riding and can therefore increase the performance of a P2P network.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Free riding; Peer-to-Peer networks; Gnutella; Distributed computing; Performance evaluation

1. Introduction

The Peer-to-Peer (P2P) computing paradigm has attracted a significant amount of interest because of its capacity for resource sharing and content distribution. Although there are different architectural designs and applications for P2P computing, file sharing is the most commonly used application; in nearly all P2P file sharing systems files are stored at peers, searched through the P2P network mechanisms, and exchanged directly between peers using the underlying network infrastructure and its protocols. In the ideal case, a file that is downloaded by a peer is automatically opened for sharing with other peers. However, peers may be reluctant to share a downloaded file to save their own resources. Therefore, the primary property of P2P systems, the implicit or explicit functional cooperation and resource contribution of peers, may fail and lead to a situation called free riding.

As a P2P concept, *free riding* means exploiting P2P network resources (through searching, downloading objects, or using services) without contributing to the P2P network. A *free rider* is a peer that uses the P2P network services but does not

[☆] This work is partially supported by The Scientific and Technical Research Council of Turkey (TUBITAK) with grant numbers EEEAG-104E028 and EEEAG-105E065.

[*] Corresponding author. Tel.: +90 312 290 2599; fax: +90 312 266 4047.

E-mail addresses: muratk@cs.bilkent.edu.tr (M. Karakaya), korpe@cs.bilkent.edu.tr (İ. Körpeoğlu), oulusoy@cs.bilkent.edu.tr (Ö. Ulusoy).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.11.002

contribute to the network at an acceptable level. A *contributor*, on the other hand, is a peer that contributes to the network by sharing its resources with other peers.

Researchers have observed a high degree of free riding in P2P networks, and they argue that it is an important threat against efficient operation of P2P networks [1,12]. Free riding causes several negative side effects. In a free riding environment, a small number of peers serve a large number of peers; many download requests are directed towards a few serving peers and this may lead to scalability problems [3]. This also leads to a more client–server like paradigm [8,9] and adversely affects P2P network advantages. For example, the fault-tolerance property of P2P networks may be weakened when a very small portion of the peers provides most of the content. Renewal or presentation of interesting content may decrease in time, and the number of shared files may grow very slowly. The quality of the search process may degrade due to an increasing number of free riders on the search horizon. As the peers age in the network, they may stop finding interesting files and may leave the system for good with all the files they shared earlier [3,10]. Moreover, the large number of free riders and their queries will generate a lot of P2P network traffic, which may lead to degradation of P2P services. Furthermore, underlying available network capacity and resources will be occupied by free riders; this will cause extra delay and congestion for non-P2P traffic.

Our focus in this paper is unstructured (pure) P2P networks, specifically the Gnutella network [17], and we provide a remedy to free riding in such networks. In an unstructured P2P network, there is no central coordination and central indexing mechanism for shared resources [4,24,25]. No peer has a global view of the network, and global behavior of the network emerges from local interactions. While these features enable unstructured P2P networks to be very successful, they also bring some problems. Among the problems of such networks is the so-called reputation problem. In an unstructured P2P network, peers interact with unknown peers and have no information about their reputations. In other words, they do not know to what extent they can trust the other peers and the data provided by them. As a result, it is not easy to detect free rider peers and act against them.

Our proposed framework consists of two mechanisms. The first mechanism is to be used for locating and detecting free riders. The second one is provided to take discouraging counter-actions against free riders. The mechanisms are distributed and localized, where each peer is only required to monitor its neighbors and make decisions and take actions based on this localized monitoring. Our solution also requires minimal changes to the current protocol processing rules and it does not require any architecture changes. As opposed to many solutions that execute the counter-actions at the download request phase, our solution executes some counter-actions at the query forwarding phase, i.e. during the search operation. In this way, our solution reduces not only the downloads performed by free riders, but also the query messages flowing in the network due to free riders. This considerably reduces the network traffic overhead.

Free riders can conceive various attacks to bypass our framework. For example, to cheat the detection mechanism free riders may behave as if they share files by submitting fake query hit messages or by sharing fake files. To avoid the counter-actions free riders can continuously change their neighbors. We provide a detailed discussion on possible attacks and how the proposed framework copes with them. Simulation experiments prove that most of the possible attacks cannot render our mechanisms obsolete.

The organization of the paper is as follows. In Section 2, we discuss the related work. The mechanisms used for locating free riders and taking actions against them are described in Section 3. The simulation model used in performance experiments and the results obtained are presented in Section 4. We discuss some possible attack scenarios from free riders to our framework and provide some possible solutions in Section 5. Finally, in Section 6, we summarize our conclusions.

2. Related work

Adar and Huberman were the first researchers to extensively analyze the user traffic on the Gnutella network and to point the high level of free riding [1]. They found out that 70% of the peers do not share any files at all. Furthermore, 63% of the peers who share some files do not get any queries for these files. It is interesting that 25% of all the peers provide 99% of all the query hits in the network.

After Adar and Huberman's work, there have been other studies observing free riding in P2P networks [9,37–39]. In general, the results indicate an increasing level of free riding. For example, Hughes et al. observed that 85% of peers share no files at all

[37]. Similarly, Yang et al. reported a high level of free riding in the Maze P2P network in spite of the incentive mechanism provided by the system [38]. Recently, Handurukande et al. observed free riding in eDonkey P2P network and concluded that the free riding phenomenon is common to most Peer-to-Peer file sharing systems, and the eDonkey P2P network is not an exception [39].

Some researchers have attempted to solve the free riding problem by following incentive-based approaches. For example, in [3], Ramaswamy and Liu propose calculating a utility function for each peer in order to estimate its usefulness to the whole community. With this method, free riders cannot download files from the system if their utility value is lower than the size of the requested file. The proposed solution depends on accurate information about peers which is provided by the peers themselves. Such a P2P network can be cheated by malicious client programs. Therefore, this method cannot fully prevent free riding. Any method to prevent free riding should be designed so that it does not solely depend on user-submitted information, or it should create incentives for the peers to report accurate information [9,18,19]. As another example, Vishnumurthy et al. [13] suggest using a single scalar value called Karma to evaluate a peer's utility to a system as in [3]. A peer's account is replicated by a group of peers, called the bank-set, in order to secure the Karma against tempering or being lost. Each of the above schemes that depends on micro payments has limitations when applied to many common P2P network architectures [11,21,36] due to the requirement of an infrastructure for accounting. In general, incentive schemes based on persistent identifiers are complicated by the anonymity of peers, by collections of widely dispersed peers, and by the ease with which peers can modify their online identity [11,21,36]. For example, to make the scheme work, a group of peers must be known to store Karma value. Whenever a peer's Karma changes, a predefined number of these peers should be reachable. Therefore, the identification of the peers should be known and be permanent. However, unstructured P2P networks do not support permanent and reliable identification mechanisms.

In our work, we do not propose to use any scoring value for a peer's utility to the system. Thus, we do not have to bother with storing, retrieving, and saving a utility value. Each peer just stores information about the neighbors' messages which are routed through it. Furthermore, we do not require the explicit cooperation of any group of peers to make the system work. Each peer executes the same kind of mechanisms alone and does not depend on any other peer's cooperation. Our approach can be implemented on both types of P2P networks, i.e., structured and unstructured. In this work, we focus on implementing it on unstructured P2P networks.

There are a number of reputation systems, such as [33–35], proposed to form a basis of an incentive system and to assist peers in their decision making while interacting with other peers. Tracking peer reputations in decentralized P2P networks may be problematic due to lack of a central authority. In the proposed solutions a certain amount of centralization is required to realize the reputation system, which contradicts with the principles of unstructured and decentralized P2P networks, such as Gnutella. The main difference between these approaches and our proposal is that our mechanisms are fully decentralized and distributed conforming to the principles of unstructured decentralized P2P networks.

In [26], the authors propose an incentive model, SLIC, to encourage cooperation in unstructured P2P networks, which also depends on the local interactions of peers. In SLIC, each peer assigns and updates weights to its neighbors based on the number of query hits it receives via its neighbors. Those weights determine the amount of messaging capacity assigned to each neighbor. Our framework does not do such a capacity assignment for neighbors. Instead, it focuses on detection of neighbors that are free riders and taking counter-actions against them. Moreover, our framework counts both query hits and query messages, and considers also the originator and receiver of these messages. Based on these information peers make a decision about their neighbors. Our framework also categorizes the free riders into several categories. This enables us to apply several different counter-actions that are tailored to the types of free riding. Additionally, while we assess the contribution of each individual neighbor to the monitoring peer and the overall system, SLIC evaluates the contribution of the sub-network reachable via each neighbor.

3. Our framework and mechanisms against free riding

In our framework against free riding, peers' contribution to the network will be monitored; if they want to use the network's services and resources, they will be forced to act cooperatively. The goal of our framework, however, is not to eliminate all possible kinds of free riding. For example, we do not aim to promote or enforce new content contribution by peers, as this may not be feasible. Our low-overhead framework aims to improve the current situation and reduce the ill-effects of free riding by detecting free riders and reducing the amount of service they get from the network. In this way, peers will be indirectly forced to cooperate in order to use the services a P2P network provides.

3.1. How an unstructured P2P network operates

Before going into details about the proposed mechanisms, we would like to summarize the important components and processes in an unstructured P2P network. Gnutella [17] represents an important class of these P2P networks. Below we mainly use the Gnutella protocol for explaining operations in unstructured P2P networks.

In an unstructured P2P network, to become a member of the network, a user (peer) has to open one or more connections with other peers that are already in the network. Once connected to the network, the user can search the network by sending a Query message (request) to its neighbors. Each neighbor then forwards the request to all its neighbors, and these peers in turn forward the request, and so on, until the message is forwarded by a predetermined number of "hops" from the sender. To restrict the broadcasting of a Query message, the number of hops is controlled using the Time-To-Live (*TTL*) field of the Query message. Each forwarding peer decreases TTL value by one. If TTL value of a Query is zero, peers drop this message instead of forwarding it. If a Query turns up a result, the peer that has the result sends back a Query Hit message to the originating peer. The Query Hit message is sent back along the opposite path the Query came in through. The Query Hit message contains the IP address and port number of the answering peer. If the user decides to download the file, it requests the file from the provider through a direct connection.

A two-tiered overlay structure which divides peers into two groups (ultrapeers – or superpeers – and leaf peers) has also been proposed. Leaf nodes are located at the "edge" of the network and they are not responsible for any routing. The leaves are connected to the overlay through a few ultrapeers. On the other hand, the nodes which have high-bandwidth and are not behind firewalls are selected as ultrapeers. Ultrapeers accept leaf connections and route their queries. This approach reduces the number of messages forwarded towards leaf peers which in turn increases the scalability of the network. In this paper, we focus on the flat unstructured P2P networks.

3.2. Our approach

Our approach against free riding requires every peer to passively monitor its neighbors. Two roles are defined for each peer: *monitoring* and being *controlled*. A peer takes both roles at the same time. As a monitoring peer, a peer monitors and records the number of messages coming from and going towards its neighbors (i.e. keeps some statistical information). The neighbors are controlled peers. At the same time, the peer is also a controlled peer, which implies that its messages are monitored and recorded by its neighboring peers. By monitoring the messages of its neighbors, a monitoring peer can decide if a neighbor is acting like a free rider. Upon deciding that the neighbor is acting as a free rider, the monitoring peer can take counter-measures against that neighbor to reduce the adverse effects of free riding.

3.3. Counters

The statistical information[1] that a monitoring peer maintains about a controlled peer P consists of a set of counters that are shown in Table 1. These counters are maintained and updated by the monitoring peer as follows.

- QR_P, the number of `Query` messages `routed` by peer P, is incremented whenever the monitoring peer receives a Query message from peer P in which the TTL value is less than the fixed max TTL. The Queries originating from peer P are not counted; only the Queries originated at somewhere else and routed by peer P are counted. The monitoring peer decides if the Query was originated by the neighbor or not by looking at the TTL value. If the neighbor P has originated the Query, then the Query message would have a TTL value equal to the fixed max TTL.

[1] Due to the power-law distribution of node degrees observed in P2P networks [6], we expect the average number of neighbors of a peer to be around 3–4, and therefore the overhead imposed by the solution on each peer will not be very large. Even if the number of neighbors is larger than the average, the space and processing requirements are very low.

Table 1
Observed counters

Symbol	Description
QR_P	Number of `Query` messages `routed` by peer *P*
QT_P	Number of `Query` messages routed `towards` peer *P*
QH_P	Number of `QueryHit` messages submitted by peer *P*
QHR_P	Number of `QueryHit` messages `routed` by peer *P*
QHS_P	Number of `QueryHit` messages `satisfying` queries of peer *P*

- QT_P, the number of `Query` messages routed `towards` peer *P*, is incremented whenever the monitoring peer sends a Query message to the neighbor *P*. Both the Query messages originated at the monitoring peer and the Query messages just forwarded by the monitoring peer are counted.
- QH_P, the number of `QueryHit` messages submitted by peer *P*, is incremented whenever the monitoring peer receives a `QueryHit` message from peer *P*. The message must be originated (not forwarded) by peer *P*. The monitoring peer can decide this by looking at the IP address field of the message, which stores the IP address of the originator of the message.
- QHR_P, the number of `QueryHit` messages `routed` by peer *P*, is incremented whenever the monitoring peer receives a `QueryHit` message from peer *P* in which the IP Address field in the message contains an IP address different than that of the peer *P*. `QueryHit` messages originating at peer *P* are not counted.
- QHS_P, the number of `QueryHit` messages `satisfying` queries of peer *P*, is incremented whenever a Query message formerly submitted by peer *P* receives a `QueryHit` through or from the monitoring peer. To observe this, whenever the monitoring peer receives a `Query` message whose TTL is the fixed max TTL, it records in its internal table (using the *message ID* of the `Query` message) that the `Query` originated from the neighbor *P*. Then, after receiving a `QueryHit` message with the same message ID, the monitoring peer decides that the `QueryHit` message is for that controlled neighbor and increments the counter QHS_P. The monitoring peer counts only once for all the `QueryHit` messages received for the same query.

The values of these counters indicate both whether the neighbor is a free rider and the type of free riding. A different set of counters is maintained for each neighbor. The details of how we employ these counters are explained in the following sections.

We need to consider the issue of whether there is enough time during a typical monitoring process to collect sufficient information about the neighbors to make correct decisions about their behavior. In one study [8], about 40% of peers in a Gnutella network leave the network in less than 4 h; only 25% of the peers are alive for more than 24 h. In another work [9], the average session duration of both Napster and Gnutella network clients is reported to be about 60 min. A similar work [10] found that 90% of Kazaa clients have sessions averaging 30 min in length. All these studies show that most peers in a P2P network stay connected long enough for monitoring peers to collect enough information to make correct decisions.

Another issue is whether a monitoring peer can monitor enough messages. In one study [2], the average number of queries received per second for three peers located at three different locations is about 50. In that same study, each peer receives or sends an average of 30 query responses per second and the query response ratio per peer is around 10–12%. This study shows that a monitoring peer will have enough messages forwarded through itself to or from a neighbor to judge if the neighbor is a free rider.

3.4. Free riding types

Previous works on free riding [12–14,21,22] have generally assumed that only one type of free riding is exhibited in a P2P network. However, studies [1–3,9,10] on P2P network traffic and user behavior suggest that not all free riders behave the same. Therefore, in this paper we define three types of free

Table 2
Summary of free riding types and their properties

FR type	None	Non-contributor	Consumer	Dropper
Sharing content?	Yes, much	No	Yes, but little	No
Replicating content?	Yes	No	No	No
Routing messages?	Yes	Yes	Yes	No
Request generation rate	Normal	Normal	Higher	Normal

riding (non-contributor, consumer, dropper) with different properties as summarized in Table 2. The types of free riding that we define here are not exhaustive. It is possible to define new types of free riding with different properties [20]. We believe that three types are sufficient for developing a general framework, and these free riding types that we focus on in this paper constitute a large fraction of all free riders. A detailed description of each type is given below.

3.4.1. Non-contributor

If a peer does not share anything at all or shares uninteresting files, it is identified as a non-contributor. A controlled peer *P* exhibiting this type of free riding can be detected by a monitoring peer who counts the `QueryHit` messages (QH_P) originating from the neighbor and compares them to the number of `Query` messages (QT_P) sent to the neighbor (Table 1).

If the number of `QueryHit` messages received is very few compared to the number of `Query` messages sent, then the neighbor is identified as a non-contributor. More precisely, if the ratio (QH_P/QT_P) is below a threshold value, then the peer is identified as a non-contributor.

Not receiving (or receiving very few) `QueryHit` messages from a neighbor may indicate that the neighbor is either not sharing any files at all, or is sharing files that are not interesting and therefore they do not match the search queries. Unfortunately, a method like this, which is based on counting the `QueryHit` messages, cannot distinguish between these two types of reasons of not responding.[2] Different approaches for setting up a threshold value can be used. Whatever the approach, however, the proposed framework enables a monitoring peer to judge if a neighbor is a non-contributor just by observing the neighbor's existing protocol messages, without requiring that any new control message be defined for detection of free riders. Below, we formulate our method to detect non-contributors as a condition that is evaluated whenever an update is performed on the values of the respective counters. We have used this formula in our simulation experiments.

```
if $(QT_P > \tau_{QT})$ and $\left(\frac{QH_P}{QT_P} < \tau_{\text{non-contributor}}\right)$ then
    peer $P$ is considered as a non-contributor
endif
```

To eliminate the warm-up period and to obtain valid statistical information we propose using a threshold value, τ_{QT}, for the number of forwarded `Query` messages to the controlled peer. A monitoring peer starts making a decision about the controlled peer after this threshold is exceeded.

3.4.2. Consumer

Peers may contribute some content to the network. They are not therefore non-contributors, but the services they use may greatly exceed their contribution. This is not a desirable behavior in terms of the long term stability of the P2P network and fairness to other peers.

To identify whether a controlled peer *P* is a consumer, a monitoring peer counts the `QueryHit` messages that originate from the neighbor (QH_P) and the `QueryHit` messages that are destined to the neighbor (QHS_P). By comparing the ratio of these two values against a threshold value τ_{consumer}, the monitoring peer can decide if the neighbor is a consumer or not.

In identifying consumers, the number of actual downloads, instead of QHS_P, could have been used. However, in unstructured P2P networks, the download process is executed directly between two peers [17]. Therefore, the intermediate nodes are not aware of the download process. This means that, the monitoring peers are not able to use actual download numbers to identify the consumers. Therefore, we propose using the `QueryHit` messages as an indication of possible downloads. We assume that if a query gets one or more `QueryHits`, the owner of the query would download at least one file. Furthermore, we only count once for all the `QueryHit` messages received for the same query. All `QueryHits` that is received for the same `Query` message will have the same unique message ID value.

The following condition is checked to decide if a neighbor is a consumer or not whenever a respective counter maintained for the neighbor and used in the formula is modified. Again thresholds for QT_P and QH_P counters are used to eliminate the warm-up

[2] Peers who are cooperative but share unpopular files would be affected by false positives. From the perspective of the performance measures we have investigated, it seems that punishing such kind of users has a small impact on the overall performance of the network. A bias against these peers is one unintended consequence of emphasizing performance in an incentive mechanism. We acknowledge that the solution of this issue is beyond the scope of this paper.

period before starting to make decisions about the behavior of a neighbor.

```
if $(QT_P > \tau_{QT})$ and $(QH_P > 0)$ and
    $\left(\frac{QH_P}{QHS_P} < \tau_{\text{consumer}}\right)$ then
    peer $P$ is considered as a consumer
endif
```

3.4.3. Dropper

A peer is identified as a dropper if the peer drops others' queries. Some peers might not forward protocol messages (Query, QueryHit, etc.) in order to save their connection bandwidth.

In order to detect a dropper peer P, a monitoring peer can count Query (QR_P) and QueryHit messages (QHR_P) forwarded by this neighbor. If the sum of these two values is very low compared to the number of Query messages sent toward the neighbor (QT_P), it can be assumed that either the neighbor does not have enough connections (to receive Query or QueryHit messages and forward them), or it drops Query and/or QueryHit messages. Again we can use a threshold value, τ_{dropper}, for the ratio.

```
if $(QT_P > \tau_{QT})$ and $\left(\frac{QR_P+QHR_P}{QT_P} < \tau_{\text{dropper}}\right)$ then
    peer $P$ is considered as a dropper
endif
```

3.5. Counter-actions against free riders

When a peer identifies a controlled peer as a free rider, the peer can start taking some actions against it. Here, we will focus on some sample counter-actions that can be implemented simply by modifying the existing P2P protocols.

Our counter-actions are based on ignoring Query messages submitted by free riders or reducing the scope of these queries. In this way, we reduce the amount of service that free riders get from the network. There are two main services that a peer can get from a P2P network: (1) *searching* for files by issuing Query messages; (2) *downloading* files after getting answers to the queries. If we reduce the amount of searching service that a free rider gets, we also cause a reduction in the amount of downloading service that it gets. Therefore, our counter-actions aim to reduce the propagation of Query messages submitted by free riders; then the free riders will have less chance of getting QueryHit messages and will perform fewer downloads.

We propose two types of counter-action schemes: (1) *single counter-action* schemes and (2) *mixed counter-action schemes*. A single counter-action applies the same action to all types of free riders. A mixed counter-action scheme applies a different counter-action for each type of free riding.

The proposed single counter-actions are described in more detail below.

3.5.1. Modifying TTL value

When a peer receives a Query message from a controlled peer, it first executes a search on local files for a match, and then forwards the Query to its other neighbors. Before the Query message is forwarded, its TTL value is normally decreased by one. However, the monitoring peer can play with this TTL value, i.e. the monitoring peer can decrement the TTL value by more than one before forwarding it further. In this way, the search horizon of the free riding peer is narrowed. This also reduces the overhead imposed by Query messages on the network. To observe the effect of this counter-action at a finer granularity for different values of TTL reduction, we propose to employ two different values, i.e., 2 and 4, for decreasing TTL.[3] We call the corresponding counter-actions TTL-2 and TTL-4, respectively.

3.5.2. Dropping requests

As a sharper counter-action, the monitoring peer can simply ignore all the search requests coming from a neighbor identified as a free rider. Dropping a Query message means not searching the local files for a match and not forwarding the Query any further; this is totally different from what happens in the Modifying TTL counter-action. We call this counter-action DROP.

Dropping the search requests of free riders or narrowing down their search horizon by modifying TTL not only punishes the free riders, but it also significantly decreases the overhead of P2P control messages over the underlying infrastructure. Uncontrolled query messages in a flooding-based P2P network can become a significant portion of overall

[3] Actually, we implemented and observed the effects of different values between 2 and 6 in the simulation experiments. To give some insight about the effect of the Modifying TTL Action with different reduction amounts on the system performance, we select TTL-2 and TTL-4 as representative values in this paper.

network traffic.[4] We believe that decreasing the number of queries submitted by free riders may help improve the performance and scalability of both P2P networks and the underlying Internet.

A monitoring peer that would like to execute a *mixed counter-action scheme* can apply an appropriate counter-action depending on the type of free riding. As mentioned earlier, a free riding peer can be either a non-contributor, a dropper, or a consumer. Thus, a possible mixed counter-action scheme may dictate that counter-action TTL-2 is applied if the free rider is a consumer, counter-action TTL-4 is applied if the free rider is a non-contributor, and counter-action DROP is applied if the free rider is a dropper. In these settings, we aim to apply more severe counter-actions to free riding types that will cause more severe damage to the P2P network. A neighbor that is not identified as a free rider will not invite any counter-action.

The type of free riding is determined according to the values of statistical counters maintained for a neighbor in the log table of a monitoring peer. When the values of the counters change, it may indicate that the type of free riding practiced by the neighbor has changed. For example, if the (QH_P/QT_P) ratio for a neighbor P is smaller than the respective threshold (i.e., the neighbor is a non-contributor), and later becomes greater than that threshold, the neighbor is no longer a non-contributor.

4. Performance evaluation

In this section, we first present our simulation model and performance metrics. Then we provide and discuss the results of the simulation experiments.

4.1. Overview of the simulation model

We used a simulation-based approach to study the model of a typical P2P network with free riding and our framework incorporated; we used this method because the model is very complex to study analytically. We implemented our simulation model including our framework on the GnuSim P2P network simulation tool that we had developed earlier [23]. GnuSim was implemented as an event-driven simulator using the CSIM 18 simulation library [16] and C++ programming language on a Windows platform. Interactions between peers and the P2P network, such as searching, downloading, pinging, etc. were implemented according to the Gnutella protocol specification given in [17].

Our model simulates a P2P network of 4900 peer nodes. The peers are inter-connected to form a mesh topology at the beginning of a simulation run. Additionally, we use a Power-Law topology for experiments reported in Section 4.3.3. We assume that all peers stay connected in the same way until the end of a simulation run.

We assume that there are three types of peers in the simulated network: type A, type B, and type C. Type A and type B peers are contributors. Type C peers are free riders. Type C peers can be further classified as a non-contributor, a dropper, or a consumer. A type C peer is randomly and uniformly assigned to one of these 3 types of free riding. The properties of peer types are summarized in Table 3. The properties of each peer type include the population ratio, shared file ratio, maximum number of simultaneous uploads possible, query generation mean, and whether peers replicate the downloaded files or not. The default values of each of these properties are set similar to the values reported in [1,9,37–39].

At the beginning of each simulation run, peers are created according to the setup explained above and assigned to one of three main types (A, B, or C). During simulation, peers interact with other peers according to their assigned types. For example, a dropper drops all the messages, a non-contributor does not share any file, etc.

There are 49 000 distinct files, with four copies of each, distributed to the peer nodes at the beginning of each simulation run. These 196 000 files are uniformly distributed to peer groups according to the

Table 3
Properties of peer types

Property	Type A	Type B	Type C
Free riding type of the peers in the peer type	None	None	Mixed
Population ratios of each peer type	10%	20%	70%
Ratio of shared files of each peer type to total files	87%	12%	1%
Peers replicate the files they downloaded or not	True	True	False

[4] For example, as it is pointed out in [7], 18 bytes of search string in a `Query` message may cause 90 megabytes of data to be forwarded by the peers of a P2P network. As another example [5], states that the total number of messages including the responses triggered by a single `Query` message can be as large as 26 240 (assuming four connections per peer).

type of the groups and the file sharing ratios presented in Table 3. We do not distribute any files to peers that are free riders of the non-contributor or dropper type. We assume that each file is the same size and can be downloaded in 60 units of simulation time. During a simulation run, peers randomly select files to search and download, and they submit search queries for them. The inter-arrival time between search requests generated by a peer follows an exponential distribution with a mean of 60 time units. We assume that the query generation rate of consumer peers is twice that of other free riding peers.

Each peer's upload capacity (the number of simultaneous uploads the peer can perform) is limited to 10. If a peer reaches upload capacity, a new upload request is rejected by the peer. The requesting peer can then try to download the file from another peer, selected from a list of peers obtained from the `QueryHit` message. We assume that the requesting peer repeats the same request a maximum of three times. After that, the peer gives up the downloading attempt and records this as an unsuccessful download. Then, it can initiate a request for another file.

Each simulation experiment is run for 4000 units of simulated time. A simulation experiment is repeated 10 times and the results for that experiment are an average of the 10 individual results.

4.2. Performance metrics

In order to measure the performance improvement of our schemes, we first determined a number of performance metrics. Below, we describe our metrics in detail.

- *Number of downloaded files*: This is an important metric indicating the number of downloads that can be performed in a P2P system during a fixed time interval. If peers can download more files from the P2P network, then level of satisfaction with the network will be higher.
- *Number of unsuccessful downloads*: The availability of content and services in a P2P network is an important issue. A network that is providing good service should not reject many of the contributing peers' requests. Since the network resources are limited, the upload capacity of peers contributing to the network will also be limited. If this limit is exceeded, the peers will start refusing download requests.
- *Number of uploads by contributors*: This metric indicates the load imposed on a peer. Contributors can become overloaded due to the excessive number of search and download operations they are involved in. Adapting free riding mechanisms in a P2P system, decreases the load on contributor peers by reducing requests from free riders.
- *Download cost*: We define the download cost for a peer as the ratio between the number of uploads and the number of downloads (i.e. #uploads/#downloads) performed by the peer. This ratio indicates the load imposed on a peer compared to the service the peer gets from the network. The smaller the ratio is, the better it is from the perspective of the peer.
- *Number of P2P network protocol messages*: This metric shows the messaging overhead in the P2P network and the underlying infrastructure. Messaging overhead affects the scalability of a system. In unstructured P2P networks particularly, the messaging overhead may be high due to the flooding approach used in querying. High numbers of protocol messages sent over the network also increase the level of congestion in the network; congestion affects the performance of several network services [15].
- *Fairness*: Fairness metric shows that the level of service that can be used by a peer is proportional to the level of contribution that is provided by that peer. In other words, a peer contributing more than what is needed to overcome the thresholds is fairly compensated with more services. Thus, the solution encourages peers to contribute more and rewards peers based on the extent of their contributions.

4.3. Simulation results and analysis

In simulation experiments, we first tested the effectiveness of our detection mechanism. Afterwards, we conducted experiments to observe changes in the performance of a P2P network when counter-action schemes are applied.

4.3.1. Evaluation of detection mechanism

The detection mechanism is a crucial part of the framework. Therefore, we did extensive simulation experiments to measure the performance of our framework in detecting free riders and free riding types. We used the following performance metrics to evaluate our detection mechanism:

- *Success ratio*: Ratio of peers correctly detected as free riders to peers designated as free riders in the beginning of each simulation run.
- *Sensitivity ratio*: Ratio of free riders whose free riding type is correctly detected to the number of peers who have been detected correctly as free riders.
- *False alarm ratio*: Ratio of peers incorrectly detected as free riders to the number of peers detected as free riders.

A good detection mechanism should provide high values for success and sensitivity ratios and low value for false alarm ratio. The success ratio is an important metric for both single and mixed counter-action schemes; sensitivity ratio on the other hand is an important metric for mixed counter-action schemes, since in those schemes the type of free riding determines the counter-action to be applied. The false alarm ratio is a metric that indicates how many peers are incorrectly detected as free riders. If the false alarm ratio is high, it means that the framework applies counter-actions to contributors; thus some contributors are negatively affected by the incorporation of the framework into the P2P network.

An important restriction on the success of the detection mechanism is the behavior and ratio of droppers. This is because free riders of the dropper type usually cannot use our detection mechanism, and hence can not apply any counter-action to their neighbors. As they do not route other peers' queries to their neighbors, they may not satisfy the detection mechanism's "routed query threshold (τ_{QT})" condition only by the count of their own queries. Therefore, in the overall detection results, droppers may play a negative role and limit the detection mechanism's success.[5] When the τ_{QT} threshold is decreased, however, droppers have more chance of satisfying the threshold value by recording only their queries, and they may then detect free riders. Thus, lowering the value of τ_{QT} increases the success ratio in the presence of droppers, as shown in Table 5.

Fig. 1 shows the Success Ratio of the detection mechanism for default values of simulation parameters. The overall success ratio is about 76%. This means that our detection mechanism is able to detect 76% of peers designated as free riders at the start of a simulation run. The false alarm ratio is about 9%. That is, 9% of the detected free riders were not really free riders. Their interactions with their neighbors during the simulations led the detection mechanism to identify them as free riders.[6]

In Section 3.4, we use some threshold values for identifying each free riding type. Table 4 shows the default values of the thresholds. Default values are based on the P2P network traffic observations reported in [1,2,8,10]. As part of our simulations we tried to observe the effect of different threshold values.

In Table 5, we observe that when the τ_{QT} threshold is set to lower values, the detection mechanism begins to detect earlier and the success ratio increases. However, the false alarm ratio also worsens with low values of τ_{QT} because the system tries to decide about a peer with less information available. There is therefore a trade-off between success and false alarm ratios and this trade-off is effected by the τ_{QT} threshold. Sensitivity is not greatly affected by the value of the τ_{QT} threshold.

Another threshold used in the detection mechanism is $\tau_{\text{non-contributor}}$, which is used to decide if a peer is a non-contributor. Table 6 shows the effect of this threshold. Interestingly, for some large values such as 0.1 and 0.01 the success ratio does not change much, but the false alarm ratio changes and becomes too high. This result suggests that high values not be used for this threshold. The success ratio does not change much for different high values of the threshold, because even the precision of the ratio is different; the number of detected peers with 0.01 is almost the same as with the value 0.1. That is, most of the non-contributor peers have a $\frac{QH_P}{QT_P}$ ratio less than 0.01. Therefore, the comparison leads to a similar success ratio. In Table 6, we again observe that the success ratio is (negatively) correlated with the false alarm ratio.

4.3.2. Evaluation of counter-actions

In Section 3.5 we proposed two types of counter-action schemes: single and mixed. We implemented

[5] For example, in our simulations we observed that the peers about which droppers cannot make any decision constitute around 20% of all the peers. This implies that our framework cannot reach a success ratio better than 80% with the current settings of the simulation parameters.

[6] This level of false alarm ratio causes 9% of the peers detected as free riders to face counter-actions; false alarms are a side effect of the detection mechanism. However, the performance metrics show that the performance is improved for contributors despite the false alarms (see Section 4.3.2).

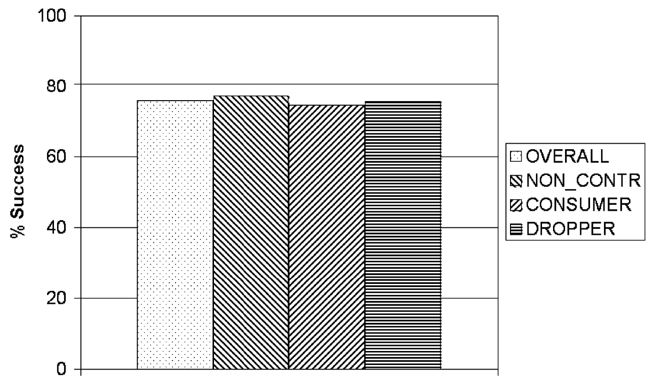

Fig. 1. Success Ratio of detection mechanism in detecting free riders and identifying their free riding types.

Table 4
Threshold values for detection mechanism

Threshold	Description	Default	Range
τ_{QT}	Threshold value for the number of routed queries toward a controlled peer to begin the detection	50	25–100
$\tau_{\text{non-contributor}}$	Threshold value for formula $\frac{QH_P}{QT_P}$ to decide if peer P is a non-contributor	0.001	0.1–0.0001
τ_{consumer}	Threshold value for formula $\frac{QH_P}{QHS_P}$ to decide if peer P is a consumer	0.1	0.05–0.5
τ_{dropper}	Threshold value for formula $\frac{QR_P+QHR_P}{QT_P}$ to decide if peer P is a dropper	0.1	0.05–0.5

Table 5
Effect of τ_{QT} threshold values on the detection mechanism

τ_{QT}	Success (%)	Sensitivity (%)	False alarm (%)
25	95.39	66.98	13.82
50	75.73	66.84	9.73
100	75.38	66.82	9.70

Table 6
Effect of $\tau_{\text{non-contributor}}$ threshold values on the detection mechanism

$\tau_{\text{non-contributor}}$	Success (%)	Sensitivity (%)	False alarm (%)
0.1	76.54	66.12	42.87
0.01	76.54	66.12	29.45
0.001	75.73	66.84	9.73
0.0001	73.03	69.27	5.24

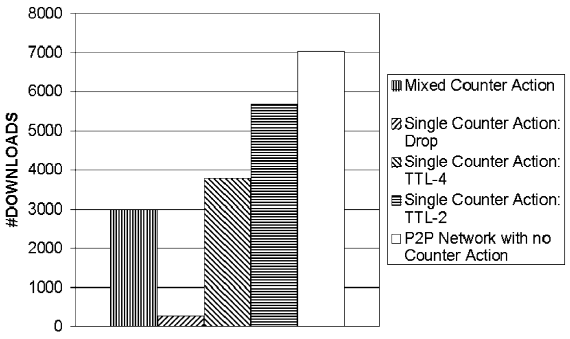

Fig. 2. Decrease in free riding peers' downloads when different counter-actions are applied.

three different single counter-action schemes: DROP, TTL-4, and TTL-2. We also implemented a mixed counter-action. This section evaluates the effectiveness of these schemes. The metrics we used in our evaluation are described in Section 4.2.

- *Downloads of free riders*: As Fig. 2 shows, the number of downloads by free riders drops when mechanisms against free riding are applied. Counter-actions against free riders decrease the reach of the `Query` messages sent by peers detected as free riders; this reduces the chance of getting a hit to one of these queries. In this way, the average number of downloads by free riders is reduced. For example, the DROP counter-action causes a 95% reduction in the number of downloads by free riders. The least successful counter-action is the TTL-2 single counter-action, which achieves a 20% reduction. But even the least successful counter-action scheme leads to fewer free rider downloads than not using any counter-action at all.

The success of the DROP counter-action is expected, since when all the queries submitted by free riders are dropped, those peers cannot get `QueryHit` messages back, and therefore they cannot download files. They can only download until they are detected. The other schemes are able to reduce the search horizon of the queries submitted by free riders, but the free riders still have the chance to get `QueryHit` messages and perform downloads. The mixed counter-action scheme yields the second-best result. We believe that this approach has important consequences compared to single action schemes. Considering the potential false alarms that can be given by the detection mechanism, applying a different counter-action depending on the severity of free riding helps us to better deal with false alarms as discussed below.

- *Downloads of contributors*: It is desirable to increase the number of downloads for contributors. Since peers' upload capacity is limited, the download requests of contributors can sometimes be rejected. The rate of rejection is higher when there are many free riders in the system. Hence eliminating the effects of free riders on the P2P network will help to increase the number of downloads that contributors can make. This is indeed shown by Fig. 3; applying our schemes achieves an increase in downloads done by contributors as much as 10%.

  Fig. 3 shows an important point; improvement in downloads is greater with a mixed counter-action than with any single counter-action. While the mixed counter-action scheme produces about a 10% improvement, the two single counter-actions, TTL-2 and TTL-4, can produce about 8% and 5% improvements, respectively. The DROP counter-action scheme actually reduces the number of downloads by contributors. We think this is due to false alarms in detection mechanisms. When we apply strict counter-actions such as DROP, the number of misdetected peers that are negatively affected is significant. On the other hand, a mixed scheme handles false alarms better by applying different counter-actions to different types of free riders, and therefore can provide different levels of punishment, from light to severe, to peers suspected as free riders.
- *Amount of P2P protocol messages*: The number of P2P protocol messages transmitted in the network is an important factor affecting the scalability of the P2P network. Counter-actions against free riders result in a reduction of up to 83% in the number of transmitted P2P protocol messages (`Query` and `QueryHit`) originating from and destined for the free riders (Fig. 4).

  When we compare the reductions in transmitted P2P control messages for different counter-actions, we see that the DROP single counter-action again gives the best results (83%). The mixed counter-action scheme, on the other hand, reduces the control traffic due to free riders by about 73%.

  If we evaluate the counter-actions with respect to their effect on reducing the total P2P control traffic in the network (i.e., the control traffic due to the free riders plus the contributors), we see that the DROP single counter-action scheme leads to a reduction of about 77%, whereas the mixed counter-action scheme leads to a reduction of about 65% (Fig. 5). The least successful counter-action is TTL-2; it leads to a reduction of 40%. All these results show that applying the proposed framework helps a P2P network handle more peers with less control messaging overhead and the system becomes more scalable with respect to the peer population.

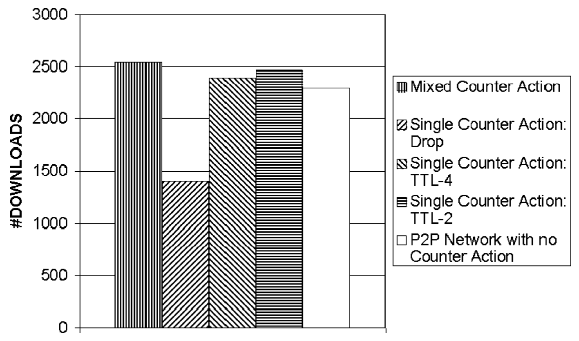

Fig. 3. Increase in contributors' downloads when different counter-actions are applied.

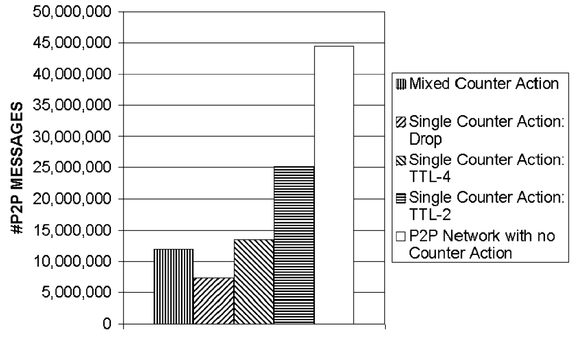

Fig. 4. Decrease in P2P messages of free riding peers when different counter-actions are applied.

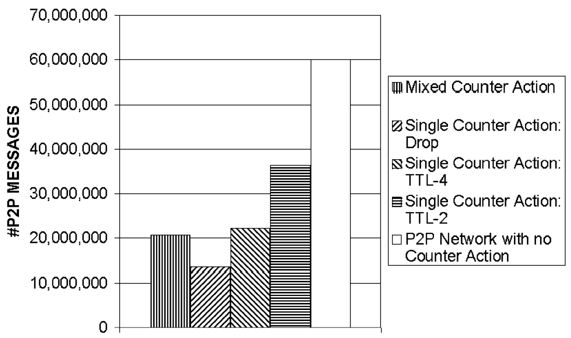

Fig. 5. Decrease in P2P messages of all peers when different counter-actions are applied.

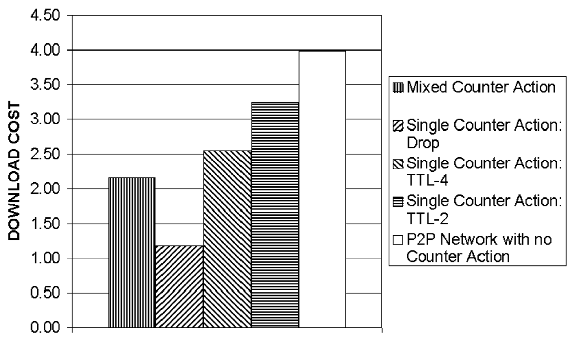

Fig. 7. Decrease in contributors' download cost when counter-actions are applied.

- *Uploads of contributors*: A metric that can indicate the load on a peer is the number of uploads done by the peer in a given time period. With our framework we want to achieve a reduction of the load on contributors. We expect that if we reduce the downloads of free riders, we can also reduce uploads, since a large portion of these uploads are done to free riders. In simulation experiments, we observed a significant reduction in the number of uploads done by contributors when a counter-action scheme is applied. As Fig. 6 shows, the scheme that gives the best result is again the DROP single counter-action scheme, causing a reduction of about 81%. The mixed counter-action scheme causes a reduction of about 40%.
- *Download cost*: The load on a contributor can also be defined as a normalized load, i.e. as the ratio of uploads to downloads. The results of our experiments show that our framework also causes a reduction in the download cost of contributors. As it can be derived from Fig. 7, the framework achieves a 70% reduction in the contributors' download cost when the DROP single counter-action is applied. The framework achieves a 46% reduction when a mixed counter-action scheme is applied.
- *Unsuccessful downloads*: We also looked at the improvement achieved in the number of unsuccessful downloads when the proposed counter-action schemes are used. As Fig. 8 shows, the DROP single counter-action achieves the best improvement; the number of unsuccessful downloads is reduced by 98%. The mixed counter-action scheme, on the other hand, reduces the number by about 91%. The decrease in the number of unsuccessful downloads means that contributors can better access the network resources when the proposed mechanisms are used. Free riders' requests and downloads may prevent non-free rider peers from accessing files and other resources. When the traffic due to free riders is reduced, the contributors start reaching to the resources more easily and get better satisfied with P2P network services.

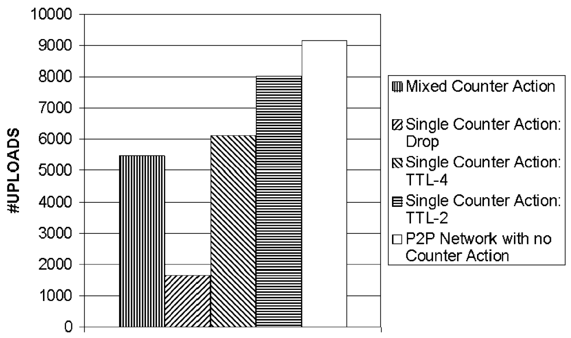

Fig. 6. Decrease in contributors' uploads when counter-actions are applied.

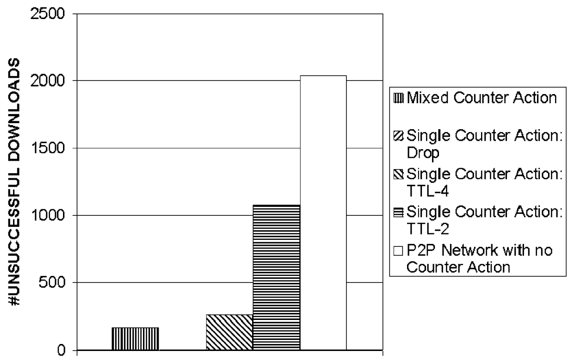

Fig. 8. Decrease in contributors' unsuccessful downloads when counter-actions are applied.

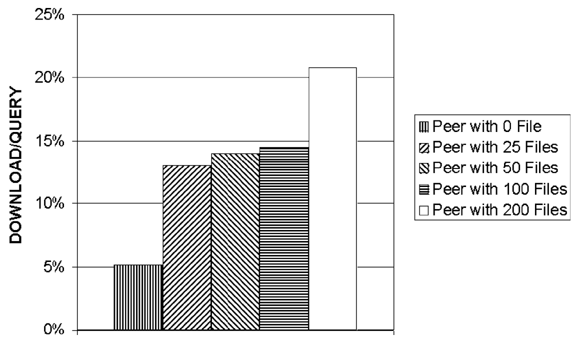

Fig. 9. Increasing utility values for increasing number of files shared by a probe node.

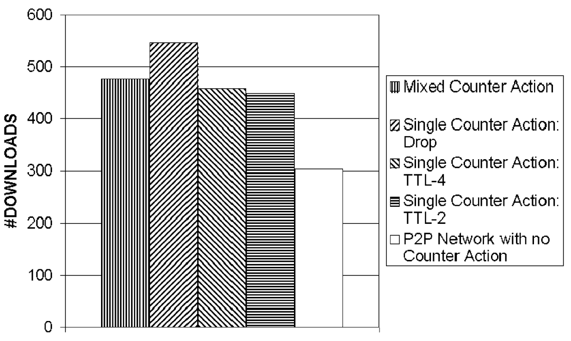

Fig. 10. Increase in contributors' downloads when a Power-Law Random Graph is simulated.

- *Fairness*: To observe the fairness of our mechanisms, we conducted several simulation experiments. In one of these experiments, we randomly chose a probe peer and assigned to it different number of files to share. As seen in Fig. 9, we assigned to the probe peer none (0), 25, 50, 100, and 200 files, and observed the Download/Query ratio (number of downloads/ number of submitted queries) as an indication of peer's utility from the system.

  As the figure shows, although the probe peer submits nearly the same number of queries, it can download different number of files depending on how much files it shares. Because, when it shares less files while requesting the same amount of service, it will face counter-actions, and this will limit the number of downloads it will be able to get. On the other hand, when it shares more files, monitoring peers will not apply any counter-action, thus it will be able to reach more peers and download more files. Therefore, if two peers have similar query patterns but provide different levels of service to the system, they will get different levels of utility from the system as well. Thus, the proposed mechanisms are fair. In other words, a peer contributing more than what is needed to overcome the threshold is fairly compensated. Hence, the proposed mechanisms not only encourage peers to provide enough services to overcome the threshold barrier, but also encourage them to contribute more to get better service.

4.3.3. Effect of different parameter values

We also executed sensitivity experiments to observe how our framework performs for different values of some important parameters: number of peers, number of shared files, level of free riding, and different network topologies. We observed that the performance results for different parameter settings are consistent and similar with the ones reporte d in this paper. Therefore, due to similarity and also space limitation, we do not provide all the results of these additional experiments here. We just report the results for a different network topology.

Recent studies investigating P2P topologies show that P2P networks exhibit small-world properties and a power-law distribution of node degrees [5,6]. To simulate such network topologies we implemented a Power-Law Random Graph of 1600 peers as suggested in [40]. Power-Law Random Graph is a topology in which the node degrees follow a power-law distribution. That is, if all peers from the most connected to the least connected is ordered, then the ith most connected peer would have $\frac{\omega}{i^{\alpha}}$ neighbors, where ω is a constant. After determining the peer degrees, we connected the peers randomly.[7]

We have executed several simulations for different values of ω and α. The results given here are obtained when ω is set 100 and α is set 0.7. We observed that our framework performs well in the Power-Law Random Graph too. This is due to the fact that the detection and counter-action mechanisms require only local interactions between neighbors. For example, Fig. 10 displays the performance in terms of the number of downloads by contributors. As shown in the figure, the increase in the number of downloads of contributors is around 50–60%

[7] In our P2P network model, links between peers are bi-directional connections which constitute both the in-degree and out-degree connections of a peer. Hence, modelling peer connections with a power-law distribution implies that both the number of in-degree and out-degree connections of the peers follow a power-law distribution.

for all counter-actions. As another example, Fig. 11 displays the decrease of the total P2P control traffic in the network (i.e., the control traffic due to the free riders plus the contributors). We observe that the DROP single counter-action scheme leads to a reduction of about 76%, whereas the mixed counter-action scheme leads to a reduction of about 72%.

5. Possible attacks to the framework and counter-measures

In this section, we describe a list of possible counter attacks against our free riding prevention mechanisms. We also discuss how our framework would react and how we can defend against those kinds of attacks.

5.1. Fake QueryHit messages

A free rider can cheat its neighbors (monitoring peers) by replying to some queries with `QueryHit` messages fraudulently as if it has the requested file. When the requesting peer asks for the file, it may just refuse uploading it. In this way it may pretend as it is serving well, since controlled peers may not be aware of unsuccessful download and cheating. In the log tables of its neighbors, the malicious peer may seem to be a non-free rider because of its `QueryHit` replies.

Given the descriptors in Gnutella protocol [17], it may not be possible for a controlled peer to observe and perceive this kind of fake messages. Because, download occurs between two peers outside the P2P network and there is no feedback mechanism for downloads in unstructured P2P networks. To handle this kind of fake `QueryHit` messages, we propose to use a new descriptor: `Notify` (see Table 7). This descriptor is used to report about a malicious peer to its neighbor. When a querying peer is refused by a responding malicious peer during a download attempt, the querying peer may send a `Notify` descriptor through the P2P network to reach the monitoring neighbor of the malicious peer. To avoid an increase in the network traffic, the querying peer does not broadcast the descriptor message. Instead, it forwards the descriptor only to the neighbor which has delivered the `QueryHit` message, containing the IP address of the denying peer. Any intermediate peer on the way to the denying peer forwards the `Notify` message to only one of its neighbors based on the message ID (GUID) of the `Query` message stored in its query routing table.[8] The monitoring peer on the path to the denying peer is the neighbor of the denying peer. After processing the `Notify` message, the last peer logs this message, and takes the necessary action against the malicious peer.

Table 7
New protocol descriptor

Descriptor	Description	Content
`Notify`	Used to report a suspected peer that refused to upload the file it provided in `QueryHit` descriptor in respond to a given `Query` descriptor	`Query` Descriptor Id; Suspected peer IP; File index

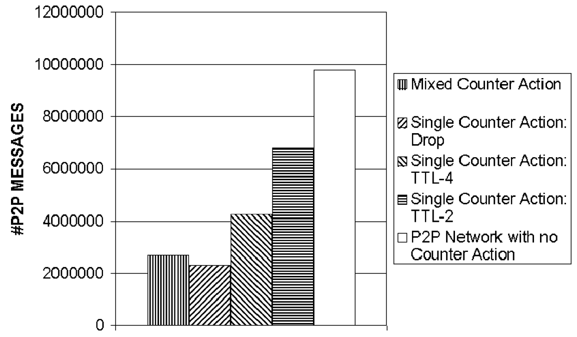

Fig. 11. Decrease in P2P messages of all peers when a Power-Law Random Graph is simulated.

There could be some side effects of the proposed `Notify` descriptor. A malicious peer can initiate an application-layer Denial of Service (DoS) attack using the Notify messages. However, almost every message type in P2P protocol (Query, QueryHit, Push, Ping, and Pong) can be exploited in order to launch denial of service attacks [27–30]. Some proposals exist in the literature aiming to counter the application-layer DoS attacks [30–32]. We think that we can also use some schemes to deal with DoS attacks using the Notify message. One scheme can be based on the comparison of the number of Notify messages routed by each controlled peer. If a monitoring peer detects a big difference among the number of Notify messages routed by its controlled peers, it can begin to filter (delete/drop) Notify messages coming from that controlled peers

[8] Since, as a requirement of Gnutella P2P Protocol, the `Query` messages are stored in the routing table of each peer for some time to route back the possible `QueryHit` messages, we do not need to store extra state information that can be used to route the `Notify` message on intermediate peers.

(similar to what is proposed in [30]). Since the danger of DoS attack exists for all P2P protocol messages, we think that the precautions taken for other P2P messages can be applied for Notify message as well. Prevention of DoS attacks is out of the scope of our current work, however, it can be interesting to investigate the applicability and effectiveness of the two simple schemes described above as a future work.

5.2. Fake files

Free riders could also share dummy files with popular names in order to cheat querying peers. These files can be very small in size to reduce upload overhead. In that way, free rider peers can conceal themselves. This situation however, can also be prevented by using the `Notify` descriptor proposed above.

5.3. Hiding query ownership

In our free riding detection mechanism, the monitoring peers exploit the TTL field value of the incoming `Query` messages to decide if the controlled peer is the owner of the message or not. If the TTL value of the `Query` message is equal to the max TTL value, then the `Query` message is assumed to be originated at the neighbor.

If a free rider wants to prevent monitoring peers applying the counter-actions against its queries, it may try to hide its ownership of the queries by setting the TTL field to a value different than the standard maximum TTL value. Then the originator of the `Query` will not be identified correctly by a monitoring peer. If the free rider sets the TTL to a value greater than the allowed maximum value, this can easily be detected by the monitoring peer. If the free rider sets the TTL to a value less than the allowed maximum, then the free rider harms itself by reducing the search horizon of the `Query`. In this case we think that there is no need to take an extra action, since we expect that a free rider will not decrease its search horizon voluntarily.[9]

[9] This is because using an initial TTL value even one less than the allowed maximum decreases the search horizon dramatically. For example, if a free rider submits a `Query` message with an initial TTL value of 6 in a network where the maximum allowed value is 7, then the free rider loses about 67% of its reach (search horizon) compared to submitting the `Query` with a TTL value of 7.

Table 8
Results of free rider malicious TTL attack

Metric	Standard TTL	Malicious TTL	Change (%)
# Downloads of FRs	2992	2019	−32.50
# Downloads of non-FRs	2543	2489	−2.12
# P2P Messages of FRs	11974199	9013018	−24.73
# P2P Messages of all peers	20613217	17361929	−15.77
# Uploads of non-FRs	5473	4471	−18.31
# Unsuccessful Downloads of non-FRs	168	58	−65.48

Mixed counter-action applied.

We observed the effects of this kind of malicious action in our simulations, and Table 8 provides the results.[10] During the experiments, we assumed that all free riders act maliciously with regard to the initial TTL value setting in `Query` messages. This is the worst case for our framework. We argue that although free riders may prevent the monitoring peers from applying counter-actions by using malicious TTL values, their level of benefits from the system and their negative effects on the system will also decrease considerably if they set the TTL value maliciously. When they cheat on the TTL, they actually reduce the reach of their own queries, and hence the quality of the results they get. As Table 8 shows, when a malicious TTL value is used, the amount of downloads of free riders decreases. The number of P2P messages observed in the network due to free riders also decreases. Hence, acting maliciously on the TTL value does not help to the free riders. Therefore, we do not see an urgent need to develop a solution against this kind of TTL attack.

5.4. Insufficient cooperation against free riding

Some peers may be reluctant to use the proposed mechanisms against free riding or malicious peers may collude with their neighbors to hide each other's "free riding status". Thus, free riders may attack the system by disabling the proposed framework. As a result, we may observe low level of cooperation against free riding due to the high population of free riders. We have simulated such an environment by applying the worst scenario (all free riders collude) and observed the results. We have compared the case when our framework is

[10] We have used the mixed counter scheme while performing simulation experiments for evaluating the effects of attacks to the framework.

Table 9
Results of insufficient cooperation attack

Metric	None of the Peers	Only Non-FRs	Change (%)
# Downloads of FRs	7041	6243	−11.3
# Downloads of non-FRs	2293	2332	1.7
# P2P Messages of FRs	44403153	32773645	−26.2
# P2P Messages of all peers	60076872	45494519	−24.3
# Uploads of non-FRs	9148	8411	−8.1
# Unsuccessful Downloads of non-FRs	2040	1467	−28.1

Mixed counter-action applied.

applied by only contributors with the case when our framework is not applied. In Table 9, we provide the results for both cases.

As Table 9 shows, even though only 30% of peers apply the mechanisms (they are contributors), the number of downloads of free riders is decreased, the messaging overhead is reduced, and the load on contributors is decreased compared to the case when our framework is not applied. This implies that our mechanisms are quite robust against the type of attack where some peers disable the proposed mechanisms by either collusion or modifying their client software.

5.5. Constantly changing neighbors

A free riding peer may attack the framework by constantly changing its neighbors, and thus it may keep utilizing the services without ever being identified as a free rider.

As discussed in Section 3.2, the P2P network traffic observations [8–10] show that peers tend to stay connected quite long periods of time. One of the reasons for that is the practical difficulty of disconnecting and re-connecting again. Another reason is that a peer does not get query hit messages immediately after it has submitted a query. A peer should not change its neighbors for the time period between submission of a query and the arrival of the respective query hits (name it *search-QueryHit cycle duration*). If the peer breaks the existing links too fast, it will not get a reply. Therefore, the peer should stay connected for at least a certain time interval which should be longer than the search-QueryHit cycle duration.

Hence, if our scheme can detect a free rider and apply a counter-action against it in a time interval that is less than the search-QueryHit cycle duration, then the attack will not work and it will not make

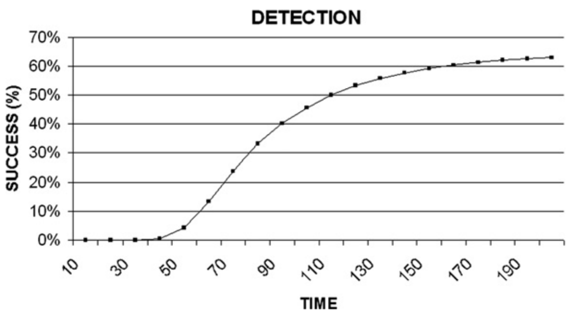

Fig. 12. The Success of the detection mechanism in the first 200 simulation time.

much sense for a free rider to try this. Therefore it is important to know how long it takes to get query hits back and how long it takes to detect the free riders. These concerns depend on several factors. The success ratio (the ratio of free riders that are detected correctly) can give us a clue about the speed of our detection mechanism. Fig. 12 plots the success ratio versus simulation time. At the beginning of a simulation run, the success ratio will be zero since there is no free rider detected yet. Towards the end of the simulation run, however, the success ratio will have a value that can be close to 1 in ideal case.

In Fig. 12, we observe that, with the default settings of simulation parameters, at time 90, 40% of free riders are detected successfully. At time 150, 60% of free riders are detected successfully.[11] From the figure we can see that free riders start becoming detected after 50 time units. Therefore, if a free rider peer would like to avoid detection, it should change its neighbors every 50 time units, with the default parameter settings. If it changes its neighbors at a rate slower than this, let us say every 100 time units, the chance to be detected and to face counter-actions becomes increased. The probability of detection becomes around 45% for 100 time units.

To investigate the effectiveness of the potential attack, we modified our simulation code to simulate this attack, and conducted several sets of new experiments. In these experiments, we first randomly selected a probe peer to act as a free rider applying the attack. During a simulation run, the probe peer changes its neighbors periodically using a fixed time period between changes. We measured the utility the probe peer gets from the P2P network at the

[11] If the P2P network traffic becomes higher (i.e. more queries are forwarded), the time required to exceed the τ_{QT} threshold will be sooner and free riders will be detected faster.

end of a simulation run. The utility is expressed as the ratio of the number of downloads the probe peer performs to the number of queries it submits. We obtained results for two different time intervals between changes of neighbors: 50 and 100 time units. The results are displayed in Fig. 13. In the figure, we also included two other utility values. One is the utility value that a contributor peer can get and the other is the utility value that a free rider who is not trying the attack (i.e., not changing connections) can get.

As can be seen in Fig. 13, the probe peer succeeded to increase its utility by changing its neighbors constantly. We can observe that the length of the time period between changes has an effect on the service the probe peer receives, as we have discussed above. If this period is longer, the probability of detection gets increased and the probe peer will more likely face counter-actions; and this will reduce the service it will get.

However, the first experiment we describe above cannot reflect a real-life scenario where lots of peers would like to apply the attack at the same time. Therefore we also conducted experiments for the scenario where all free riders in the network apply the attack expecting to increase the utility they get. The results are displayed in Fig. 14. As seen in the figure, the probe node acting as a free rider and applying the attack is negatively affected in this case when all free riders in the network apply the attack. This is because, one of the side effects of the suggested attack is that when all free rider peers change their neighbors, their previous neighbors lose the connection via these peers and they would lose the possible incoming `QueryHit` messages as well. Since the `QueryHit` messages in unstructured P2P networks are routed back through the same route of the received `Query` messages, when an intermediate peer tries to route a `QueryHit` which is routed by a free rider peer, it could not route it anymore, due to the changed neighbors. So, some of the `QueryHit` messages would be dropped without reaching to the destined peers. As observed in the figure, this side effect is not negligible. The probe peer loses its advantage considerably when all other free riders also apply the same attack.

Therefore, we can conclude that although the attack seems to increase the utility of an individual free rider, in a more general and real situation, when all or most of the free riders apply the attack, the utility that a free rider gets is not increased to a level to justify the practical difficulties of applying the attack. The free rider will not reach to a level of utility comparable to that of a contributor peer.

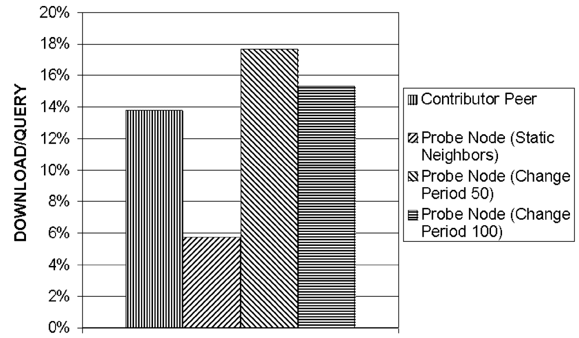

Fig. 13. The results for the Probe peer, when the attack is only applied by the probe FR peer.

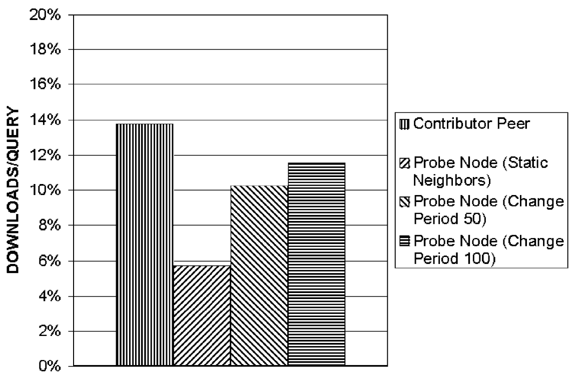

Fig. 14. The results for the Probe peer, when the attack is applied by all the FR peers.

6. Conclusion

In this work we have proposed a distributed framework to reduce the degree of free riding in unstructured P2P networks. The framework is simple to implement, has low-overhead to run, fully complies with the concepts and protocols of unstructured P2P networks, and is decentralized to operate efficiently.

We first specified possible free riding types that could be encountered in a P2P network. We then proposed some mechanisms to detect free riders of these types. We also proposed some possible counter-actions to apply against peers detected as free riders. By reducing the amount of free riding in a P2P network, we aim to increase the quality of service that peers can get from the network, the availability of content and services, the robustness of the system, the balance of the load on peers, and the scalability of the network. As the performance results of simulation experiments indicate, the mechanisms do reduce the level of free riding and its adverse effects on P2P networks; the performance of the P2P network is considerably improved.

In general, the DROP single counter-action against all kind of detected free riders results in the largest improvement for all performance metrics except the number of downloads by contributors; latter result arises from false detection in determining free riders. To increase performance for contributors, we suggest using a mixed counter-action scheme as it is the best counter-action if we also include increasing downloads.

References

[1] Eytan Adar, Bernardo A. Huberman, Free riding on Gnutella, 2000. <http://www.firstmonday.dk/issues/issue5_10/adar>.

[2] Evangelos P. Markatos, Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella, in: IEEE International Symposium on Cluster Computing and the Grid, 2002, pp. 65–74.

[3] Lakshmish Ramaswamy, Ling Liu, Free riding: a new challenge to peer-to-peer file sharing systems, in: Annual Hawaii International Conference on System Sciences, 2003.

[4] Karl Aberer, Manfred Hauswirth, Peer-to-Peer information systems: concepts and models, state-of-the-art, and future systems, in: International Conference on Data Engineering, 2002.

[5] Karl Aberer, Manfred Hauswirth, An overview of peer-to-peer information systems, in: Workshop on Distributed Data and Structures, 2002.

[6] M. Jovanovic, F.S. Annexstein, K.A. Berman, Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella, Technical Report, University of Cincinnati, 2001.

[7] Jordan Ritter, Why Gnutella Cannot Scale. No, Really, 2001. <http://www.darkridge.com/jpr5/doc/gnutella.html>.

[8] Matei Ripeanu, Ian Foster, Adriana Iamnitchi, Mapping the Gnutella network: properties of large-scale peer-to-peer systems and implications for system design, IEEE Internet Computing, Journal 6 (1) (2002) (Special issue on peer-to-peer networking).

[9] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble, A measurement study of peer-to-peer file sharing systems Multimedia Computing and Networking, 2002.

[10] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, John Zahorjan, Measurement, modeling, and analysis of a peer-to-peer file-sharing workload, in: ACM Symposium on Operating Systems Principles, 2003.

[11] Ramayya Krishnan, Michael D. Smith, Zhulei Tang, Rahul Telang, The impact of free-riding on peer-to-peer networks, in: Annual Hawaii International Conference on System Sciences, 2004.

[12] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, Incentives for sharing in peer-to-peer networks, Electronic Commerce, 2001.

[13] Vivek Vishnumurthy, Sangeeth Chandrakumar, Emin Gun Sirer, KARMA: a secure economic framework for P2P resource sharing, in: Workshop on the Economics of Peer-to-Peer Systems, 2003.

[14] Sepandar D. Kamvar, Mario T. Schlosser, Hector Garcia-Molina, Addressing the non-cooperation problem in competitive P2P networks, in: Workshop on Economics of P2P Systems, 2003.

[15] Atip Asvanund, Karen Clay, Ramayya Krishnan, Michael Smith, An empirical analysis of network externalities in Peer-to-Peer music sharing networks, in: International Conference on Information Systems, 2002.

[16] Herb Schwetman, CSIM: a C-based, process oriented simulation language, in: Winter Simulation Conference, 1991.

[17] Clip2, The Gnutella Protocol Specification v0.4 (Document revision 1.2), 2001. <http://www9.limewire.com/developer/gnutellaprotocol0.4.pdf>.

[18] Kazaa Lite Web Site, 2004. <http://www.k-lite.tk>.

[19] Leander Kahney, Cheaters bow to peer pressure, 2001. <http://www9.wired.com/news/tecnology/0,1282,41838,00.html>.

[20] M. Karakaya, I. Korpeoglu, O. Ulusoy, A distributed and measurement-based framework against free riding in Peer-to-Peer networks, in: IEEE International Conference on Peer-to-Peer Computing, 2004.

[21] Nazareno Andrade, Francisco Brasileiro, Walfredo Cirne, Miranda Mowbray, Discouraging free-riding in a Peer-to-Peer grid, in: IEEE International Symposium on High-Performance Distributed Computing, 2004.

[22] Nazareno Andrade, Miranda Mowbray, Walfredo Cirne, Francisco Brasileiro, When can an autonomous reputation scheme discourage free-riding in a Peer-to-Peer system, in: Workshop on Global and Peer-to-Peer Computing, 2004.

[23] M. Karakaya, I. Korpeoglu, O. Ulusoy, GnuSim: a Gnutella network simulator, Technical Report BU-CE-0505, Department of Computer Engineering, Bilkent University, 2005. <http://www.cs.bilkent.edu.tr/tech-reports/2005/BU-CE-0505.pdf>.

[24] George H.L. Fletcher, Hardik A. Sheth, Katy Börner, Unstructured Peer-to-Peer networks: topological properties and search performance, in: International Workshop on Agents and Peer-to-Peer Computing, 2004.

[25] D. Schoder, K. Fischbach, C. Schmitt, Core concepts in Peer-to-Peer (P2P) networking, in: R. Subramanian, B. Goodman (Eds.), P2P Computing: The Evolution of a Disruptive Technology, Idea Group Inc., Hershey, 2005.

[26] Qixiang Sun, Hector Garcia-Molina, SLIC: a selfish link-based incentive mechanism for unstructured Peer-to-Peer networks, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2004), 2004.

[27] A. Kuzmanovic, D. Dumitriu, E. Knightly, I. Stoica, W. Zwaenepoel, Denial-of-service resilience in Peer-to-Peer file sharing systems, in: The ACM SIGMETRICS'05, 2005.

[28] Neil Daswani, Hector Garcia-Molina, Beverly Yang, Open problems in data-sharing Peer-to-peer systems, in: ICDT, 2003.

[29] F. Dabek, E. Brunskill, M.F. Kaashoek, D. Karger, Building peer-to-peer systems with Chord, a distributed lookup service, in: Hot Topics in Operating Systems Workshop, May 2001.
[30] N. Daswani, H. Garcia-Molina, Query-flood DoS attacks in Gnutella, in: ACM Conference on Computer and Communications Security, Washington, DC, November 2002.
[31] S. Osokine, Flow control algorithm for distributed 'broadcast-route' networks with reliable transport links, 2001. <http://www.grouter.net/gnutella/flowcntl.htm>.
[32] Christopher Rohrs, Sachrifc: simple flow control for Gnutella, 2002. <http://www.limewire.com/developer/sachrifc.html>.
[33] Minaxi Gupta, Paul Judge, Mostafa Ammar, A reputation system for Peer-to-Peer networks, in: ACM 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), 2003.
[34] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, F. Violante, A reputation-based approach for choosing reliable resources in peer-to-peer networks, in: 9th ACM Conference on Computer and Communications Security, November 2002.
[35] S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina, Eigenrep: reputation management in P2P networks, in: 12th International WWW Conference, 2003.
[36] Michal Feldman, John Chuang, Overcoming free-riding behavior in Peer-to-Peer systems, ACM SIGecom Exchanges 5 (4) (2005).
[37] D. Hughes, G. Coulson, J. Walkerdine, Free riding on Gnutella revisited: the bell tolls? IEEE Distributed Systems Online 6 (6) (2005).
[38] M. Yang, Z. Zhang, X. Li, Y. Dai, An empirical study of free-riding behavior in the maze P2P file-sharing system, in: IPTPS, 2005.
[39] Sidath Handurukande, Anne-Marie Kermarrec, Fabrice Le Fessant, Laurent Massoulié, Simon Patarin, Peer sharing behaviour in the edonkey network and implications for the design of serverless file sharing systems, in: the First EuroSys Conference (Eurosys'2006), Leuven (Belgium), April 2006.
[40] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured Peer-to-Peer networks, in: ICS'02, USA, June 2002.

Murat Karakaya is currently a Ph.D. candidate in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His current research interests include Peer-to-Peer networks and mobile database systems.

İbrahim Körpeoğlu received his Ph.D. and M.S. degrees from University of Maryland at College Park, both in Computer Science. He is currently an Assistant Professor in the Computer Engineering Department of Bilkent University, Ankara, Turkey. Prior to joining Bilkent University, he worked in Ericsson, IBM T.J. Watson Research Center, Bell Labs, and Telcordia Technologies, in USA. His research interests include computer networks, wireless ad hoc and sensor networks, mobile computing, and P2P networks.

Özgür Ulusoy received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. He is currently a Professor in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His current research interests include Peer-to-Peer and mobile systems, web querying, and multimedia database systems. He has published over 80 articles in archived journals and conference proceedings.