A Distributed and Measurement-based Framework Against Free Riding in Peer-to-Peer Networks

Murat Karakaya, Ibrahim Korpeoglu, Ozgur Ulusoy
Department of Computer Engineering
Bilkent University
06800 Ankara, Turkey
{muratk,korpe,oulusoy}@cs.bilkent.edu.tr

Abstract

Peer-to-peer networks have attracted a significant amount of interest as a popular and successful alternative to traditional client-server networks for resource sharing and content distribution. However, the existence of high degrees of free riding may be an important threat against P2P networks. In this paper [1], we propose a distributed and measurement-based method to reduce the degree of free riding in P2P networks. We primarily focus on developing schemes to locate free riders and on determining policies that can be used to take actions against them. We propose a model in which each peer monitors its neighboring peers, makes decisions if they exhibit any kind of free-riding, and takes appropriate actions if required. We specify three types of free riding and their symptoms observable from the activities of the neighboring peers. We employ simple formulas to determine if a peer exhibits any kind of free riding. The counter actions to be applied to the free riders are defined. We combine the mechanisms proposed to detect free riders and to take appropriate actions in an ECA rule and a state diagram.

1 Introduction

Peer-to-peer (P2P) networks have attracted a significant amount of interest both in the Internet community and in the academic world as a popular and successful alternative to traditional client-server networks for resource sharing and content distribution. Although there are different architectural designs and applications for P2P file sharing, in nearly all P2P systems files are stored at peers, searched through the P2P network mechanisms, and exchanged directly between peers using the underlying network mechanisms. In an ideal case, a file that is downloaded by a peer is automatically opened for sharing with other peers. However, peers can, and frequently do, turn off this property and stop sharing a downloaded file to economize on their own resources such as bandwidth. 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 at desirable levels. Researchers have observed the existence of high degrees of free riding in P2P networks and they suggest that free riding may be an important threat against the existence and efficient operation of P2P networks [2].

There may be various reasons and motivations behind free riding. For example, peers with a Network Address Translation (NAT) address may act as a free rider. Bandwidth limitation would be another cause. Another reason would be the peers' concern of sharing "bad" or "illegal" data on their own computers. Some peers may concern about security if they share something.

Free riding may cause several negative side effects on P2P networks. In a free riding environment, a small number of peers will serve for all other peers. Therefore, many download requests will be directed towards these peers which may lead to scalability problems [7]. Renewal of content or presenting interesting content may decrease in time, thus the number of shared files may become limited or may grow very slowly. Fault-tolerance property of P2P networks may be adversely affected due to the fact that a very small portion of the peers provides most of the content[2]. This also leads to a client-server like paradigm [8, 10] and decreases P2P network advantages. Quality of search process may degrade due to increasing number of free riders in the search horizon. As the peers age in the network, they begin not to find interesting files and may leave the system for good with all the files and resources that they have shared earlier [7, 4]. Moreover, a large number of free riders and their queries will generate a great amount of P2P network traffic, which may lead

[1]This work is supported in part by The Scientific and Research Council of Turkey (TUBITAK), Grant Number: EEEAG-103E014

[2]1% of the peers provides 37% of the content [2].

to degradation of P2P services. Furthermore, underlying available network capacity and resources will be decreased by free riders, which will cause extra delay and congestion to non-P2P traffic.

In this paper, we propose two mechanisms to cope with free riding. The first mechanism primarily focuses on locating and detecting free riders, whereas the second one deals with taking actions against them. We propose a model in which each peer monitors its neighbors, makes decisions, and take actions accordingly. As the first step of our work, we propose several design criteria which should be met by any P2P system aiming to prevent free riding. Then, we specify three free riding types and their symptoms observable from the activites of neighboring peers. We present simple methods and formulas to determine if a peer exhibits any kind of free riding activity. The counter actions which will be applied against free riders are also specified. We than integrate tother the hints that suggest a peer to be free rider and the counter-actions that can be applied to such a peer using a finite state diagram that shows the possible states and the transitions between them. We also represent the transitions using an ECA rule that enables automatic execution of conter-actions upon updates and depending on the current conditions. We identify three possible counter-actions that can be applied against a peer that is exhibiting free-riding behavior.

The organization of the paper is as follows. Section 2 briefly describes the related work. In Section 3, following the discussion of the design criteria and performance metrics, the mechanisms for locating free riders and taking actions against them are described. In the last part of this section, integration of the proposed mechanisms in the form of a finite state machine and an ECA rule is presented. Section 4 discusses the possible attacks against the proposed mechanisms by free riders. Finally, the conclusions are presented in section 5.

2 Related Work

User traffic on Gnutella network is extensively analyzed in [2] and it is observed that 70% of peers do not share any file at all. Furthermore, 63% of the peers who share some files do not respond to any queries. Another interesting observation is that 25% of the peers provide 99% of the whole content in the network.

In a more recent work, Saroui et. al. confirm that there is a large amount of free riding in Gnutella network as well as in Napster [10]. Another interesting observation is that 7% of the peers together provide more files than all of the other remaining peers.

In [7], Ramaswamy and Liu concentrate on how to prevent free riding. They propose to calculate a utility function for each peer in order to estimate its usefulness to all community. According to the result of the function, P2P network will permit a peer to search and download a file or just reject its request. The function is based on two parameters; the total size of the files downloaded, and total size of the files uploaded by the peer. The difference of these two values determines the utility of the user to the system. If the user requests a file to download with a size less than the utility value, then it is permitted to download. Otherwise, it is refused. There are two ways to increase the utility value for a peer: either the peer can upload new files, or the peer can wait for some time for a bonus utility value. When a peer downloads a file, its utility is decreased by the amount of the size of the downloaded file.

With the proposed method, a free rider can not download a file from the system if its utility value is lower than the size of the requested file. However, there can be some ways to walk around the utility values. For example, a user can share some small files with fake names resembling popular file names. Other users can download these files and in this way the peer can get utility values for them. Moreover, the proposed method depends on accurate information about the peers which is provided by the peers themselves. A P2P network depending on such a protocol can be misreported and cheated by writing some malicious client programs.

In a recent work [11], Vishnumurthy et.al. suggest using a single scalar value, called Karma, to evaluate a peer's utility to a system like in [7]. Each peer has an account consisting of Karma. When a peer uploads a file to a requesting peer, it gets some amount of Karma from the requesting peer. On the other hand, if the peer downloads a file, it gives some amount of its Karma to the peer serving the file. The account of a peer is replicated by a group of peers, called the bank-set, in order to ensure Karma against loose and tampering. The transfer of Karma between peers is executed through bank-set of each peer. The main difference from the work in [7] is that the utility value of a peer is not stored at the peer itself but at some other peers.

However, 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. Thus, the prototype of the proposed scheme was implemented on top of Pastry, which is a P2P network using a distributed hash table (DHT), in other words a structured P2P network.

3 Mechanisms Against Free Riding

We think that in order to reduce the amount of free riding and increase cooperation among peers, availability of a set of mechanisms is required.

We try to create an environment in which peers will be monitored about their contributions to a P2P network and enforced to act in more cooperation and to

contribute to continue using the services and resources of the P2P network.

3.1 Design Criteria

While developing some mechanisms to prevent or diminish free riding, we need to consider several design issues some of which are listed below:

- **Simplicity:** The actions observed and reactions to them should be simple to implement and manage.
- **Decentralism:** Making decisions and taking actions should be executed in a decentralized way.
- **Low overhead:** Methods should not cause much overhead. Non-free riders should not devote much resources to prevent free riding.
- **Abuse-proof:** Peers may try to walk around mechanisms by misreporting their status or implementing their own client programs. Mechanisms must not depend on information provided by peers solely. Instead, mechanisms should depend on P2P paradigm to collect information about peers.

3.2 Performance Metrics

We propose the use of the following metrics to evaluate the performance of a P2P system that applies mechanisms to prevent and reduce free riding.

- **Quality of Service**: If possible, quality of service (QoS) received from a P2P network by non-free riders should be increased, where the QoS received by free riders should be decreased. QoS parameters may include search time spent in resolving queries, hit quality and quantity, and download time.
- **Availability**: We expect that by increasing the cooperation between peers and by forcing free riders to contribute, availability of content and services in a P2P network can be increased. For example, a scheme that could replicate popular items on free riders would increase hit ratio for those items, even though the original content providers would leave the system.
- **Load Sharing and Scalability**: Content provider peers can be bottleneck due to excessive search and download operations they are involved in. Via increased cooperation in a P2P system, the load on those peers can be also shared by other peers that would otherwise be free riders. This will help the system to be more scalable where larger number of search queries and download operations can be executed on the system successfully.

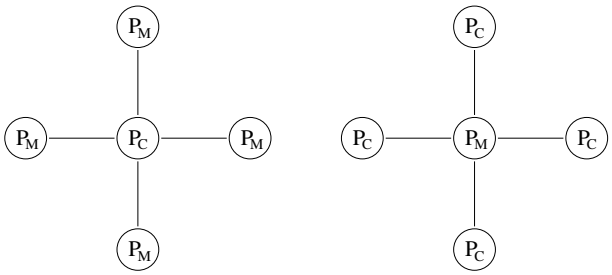

Figure 1. Peers can be in two roles: monitoring and controlled.

- **Robustness**: Mechanisms against free riding can make a P2P network more robust against disconnections and legal attacks, which will increase network population in terms of available content and also in terms of number of nodes that are reachable. This will expand the search horizon and will increase the hit ratio in search operations.

3.3 Locating and taking actions against free riders

We propose a system in which every peer passively monitors the activities of its neigboring peers. In the proposed system, peers can be classified into two different roles as shown in Figure 1. In the first type of role, a peer functions as a monitoring peer, P_M, which monitors and records the number of messages coming from and going to neighboring peers. The messages are implemented with descriptors in Gnutella Protocol [3] (see Table 1), the protocol upon which our solution is based. At the same time, each peer is a controlled peer, P_C, which means that its messages are monitored and counted by its neighboring peers.

3.3.1 Locating Free Riders

In order to determine if a peer P_C is a free rider or not, we may exploit several clues that may be derived from its observed behaviours. To derive the clues about the neigboring peers, we need to maintain several information about the neighbors and their behaviors. Due to the small-world phenomena, average number of neighbors of a peer is expected to be about 3-4 [5]. Therefore, this process does not impose high overhead on peers.

The information that is maintained about neighbors of a peer constists of some statistical counters which are presented in Table 2. These counters are updated when messages are received from the neigbors and when messages are sent towards the negbors. The clues about the neigboring peers (if they are free riders or not, and the types of free riding they exhibit if they are free riders) are derived from the values of these counters.

One issue to consider is whether there exists enough time to collect statistical information from neigboring peers and make decisions. It is known that in a P2P

Table 1. Gnutella Protocol Descriptors

Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a `Query` descriptor will respond with a `QueryHit` if a match is found against its local data set.
QueryHit	The response to a `Query`. This descriptor provides the recipient with enough information to obtain the data matching the corresponding `Query`.
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.

network peers can join and leave the system at any time. We can find some related work in the literature about the network topology dynamics and peer characteristics of P2P applications. In [8], it is stated that about 40% of the peers leave the Gunetella network in less than 4 hours, while only 25% of the peers are alive for more than 24 hours. In another work [10], the median session duration of both Napster and Gnutella clients is about 60 minutes. In a similar work [4], 90% of average session lengths of Kazaa clients is found to be about 30 minutes. As a result, it can be assumed that peers stay connected long enough to collect statistical information about them and take necessary actions.

Another issue is whether a monitoring peer can snoop and monitor enough number of messages that are coming from or going towards the neighboring peers. In [6], it is reported that the average number of queries per second for three peers located at different geographic locations is about 50. Also, about 30 query responses per second are recorded. This shows that a peer will have enough number of messages forwarded over itself to judge if a neighboring peer is a free rider or not.

3.4 Free Riding Types

In this section, we identify some possible free riding types that a peer may exhibit. We also formulize how the identified free riding types can be detected by using the statistical information gathered about a free riding peer.

Table 2. Observed Counters

Symbol	Description
Q_P	Number of `Query` descriptors submitted by P_C.
RQ_P	Number of `Query` descriptors routed by P_C.
TQ_P	Number of `Query` descriptors routed towards P_C.
QH_P	Number of `QueryHit` descriptors submitted by P_C.
RQH_P	Number of `QueryHit` descriptors routed by P_C.
SQH_P	Number of `QueryHit` descriptors satisfying queries of P_C.
N_P	Number of `Notify` descriptors submitted for P_C.

- **A peer does not share anything at all or shares uninteresting files.** It may be observed that a neighboring peer does not return any `QueryHit` messages to the queries that it receives. There may be two reasons for that: either the peer does not have any files matching the queries, or the peer does not share any files at all. To decide if a peer is a zero-content (or an uninteresting content) contributor, whenever the monitoring peer initiates a search or routes a search on behalf of another peers by sending a `Query` message to its neighbors, the monitoring peer also increases the value of the respective TQ counters (maintained in a log table) for its neighbors. The monitoring peer also observes and counts the `QueryHit` messages received from the neighboring peers. If the monitoring peer receives a `QueryHit` message that has the IP address of one of its neighbors in it, the monitoring peer increases the value of the QH counter maintained for that peer in the log table. Receiving a `QueryHit` message originating from a neigboring peer indicates that the neigboring peer is sharing an interesting file that is requested.

  The monitoring peer then compares the values of TQ and QH counters maintained for a neighboring peer, to decide if that peer is a free rider that is not sharing any files (a *non-contributor*). More specifically, for this decision to be made, the monitoring peer may compare the QH/TQ ratio against a theshold value and decide that the neigbor is a free rider of type non-contributor if the ratio is smaller than the threshold. Several different approaches for setting up a threshold value may be proposed[3]. Below, we formulize the condition that is required to judge if a neigboring peer is a free

[3] We may set up a constant value for unsatisfied query number, $(TQ_P - QH_P)$, e.g. 100. Or we may use a time based threshold, e.g. 10 minutes. If there is no `QueryHit` message from the peer in that period of time, we may treat this peer as non-contributor.

rider or not.

Furthermore, to remove the warm-up period and to obtain valid statistical information we propose to use a threshold value for the number of forwarded `Query` messages to the observed peer, τ_{TQ}. A monitoring peer start deciding about a neighboring peer after this threshold.

if ($\text{TQ}_P > \tau_{TQ}$) and ($\frac{QH_P}{TQ_P} < \tau_{non\_contributor}$) **then**
peer P is considered as a `non-contributor`
endif

- **A peer consumes more resources than that it shares.** A monitoring peer counts the `QueryHit` responses (QH) originated from its neighbors and successful `QueryHit` messages (SQH) destined to and received by its neighbors. The comparison of these two numbers reveals if any of the neighboring peers consumes more than it shares. More specifically, a threshold value, $\tau_{consumer}$, can be compared against the ratio of these two numbers. If the ratio QH/SQH is smaller than the threshold, a decision that the neigboring peer is a free-rider of type *consumer* can be made.

  if ($\text{TQ}_P > \tau_{TQ}$) and ($\frac{QH_P}{SQH_P} < \tau_{consumer}$) **then**
  peer P is considered as a `consumer`
  endif

- **A peer drops others' queries.**

  A monitoring peer counts `Query` and `QueryHit` messages forwarded by each of its neighbors. If these two values are very low for a neighboring peer, it can be assumed that the neighboring peer does not have enough connections or it drops queries and/or query hits. $\tau_{dropper}$ is used as a threshold value. We call this type of free rider as a *dropper*.

  if ($\text{TQ}_P > \tau_{TQ}$) and ($\frac{RQ_P + RQH_P}{TQ_P} < \tau_{dropper}$) **then**
  peer P is considered as a `dropper`
  endif

3.4.1 Actions against Free Riders

If a peer identifies another peer as a free rider, it can take some counter-actions against it. We specify three level of actions. Level 1 action is the least restrictive for the free rider. Level 3 action is the most restrictive for the free rider.

- **Level 1 Action: Decrementing TTL value:** Normally, when a peer receives a `Query` message from a neighboring peer, it first executes the search on local files for a match, then the `Query` is forwarded to the other neighboring peers. Before the `Query` message is forwarded, its TLL value is decremented by one. To act against a suspected free rider, the monitoring peer can play with the TTL value for `Query` messages that are received from the suspected peer, i.e. it can decrement the TTL value by more than one before forwarding. In this way, the search horizon of the free riding peer is narrowed down. This also reduces the overhead that `Query` messages are imposing on the network.This counter-action is applied to a peer that exhibits only one type of free riding, i.e. it is either a non-contributor, or a dropper, or a consumer.

- **Level 2 Action: Ignoring requests:** A free rider can be punished by the monitoring peer by ignoring the searches (i.e. the `Query` messages) originating from that free riding peer. The `Query` messages originating from the free rider peer can be partially or totally ignored. Ignoring a `Query` message means not searching the local files for a match and not forwarding the `Query` any more. In other words, the `Query` is message is simply dropped. We can do this action parametric, so that the probability of ignoring (dropping) the `Query` messages can be adaptive and tunable. However, a monitoring peer should be careful about the origin of the `Query` messages while dropping them. It has to drop only the messages that are originated from a free riding peer. This counter-action is applied to a peer that is exactly exhibiting two types of free-riding (for example a peer that is both a consumer and a dropper).

  We expect that ignoring the requests of free riders (fully or partially) does not only punish the free riders, but also improves the overall system performance. If not controlled, Query messages may become a significant fraction of overall network traffic. For example, as it is pointed out in [9], an 18 bytes of search string in a Query message may cause 90 megabytes of data to be forwarded by the P2P network peers. As another example, [1] states that total number of messages including the responses triggered by a single Query message can be as large as (assuming 4 connections per peer):

$$2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26240 \qquad (1)$$

- **Level 3 Action: Disconnecting from network:** If a peer is sure that a neighboring peer is a free rider that is exhibiting all types of free riding, the peer may drop the connection with that peer. In that way, the peer saves its resources which can later be allocated to another peer. The difference between ignoring (all or partial) search request and disconnection is that, in the preceding method, if any change in behavior of the peer is observed, the punishment can be canceled. However, when disconnection is executed, the disconnected

peer should reconnect to the system through a new peer.

3.5 Putting all together

In the previous sections we have discussed the clues to detect free riding types and possible counter actions that can be taken against free riders. We now would like to integrate them together using an ECA (event-condition-action) rule and a finite state machine (FSM). As described in section 3.4, a free-riding peer can be a non-contributor, or a dropper, or a consumer, or a combination of these. A neighboring peer can be also a good behaving peer, in which case it is not a free-rider and it will not show any of the mentioned free riding types.

As the first step towards a formal description, we will use three Boolean variables to denote if a neighboring peer is non-contributor, or a dropper, or a consumer or not. We call these three Boolean variables as N (for non-contributor), D (for dropper), and C (for consumer). If the counter values maintained at the log table of the monitoring peer indicate that the neighboring peer is a non-contributor, then N has the value 1, otherwise it has the value 0. If counter values at the log table indicate that the peer is a dropper, then D has the value 1, otherwise D has the value 0. If the counter values indicate that the peer is a consumer, then C has the value 1, otherwise it has the value 0.

When the counters maintained for the neighboring peer P at the log table of the monitoring peer change, the values of these variables (N, D, and C) may also change. For example, if (QH_P/TQ_P) ratio was first smaller than the respective threshold (i.e. N = 1), and later becomes greater than that threshold, peer P becomes no longer a non-contributor and the value of N changes from 1 to 0.

At any moment in time, depending on the counter values maintained in the log table of a monitoring peer (and hence depending on the values of the above mentioned three Boolean variables) we may have one of the following eight conditions shown in Table 3 holding for a neighboring peer P.

Table 3. Conditions

N	D	C	Condition
0	0	0	C0
0	0	1	C1
0	1	0	C2
0	1	1	C3
1	0	0	C4
1	0	1	C5
1	1	0	C6
1	1	1	C7

If, for example, condition C0 holds at a given time, that means there is no free-riding at all at that time. If one of the conditions C1, C2, or C4 holds at a given time, that means only one type of free riding is exhibited by the neighboring peer P. This means, peer P is either a non-contributor, or a dropper, or a consumer. In other words, either N, or D, or C has the value of 1, and the other two variables have the value of 0. If one of the conditions C3, C5, or C6 holds at a given moment, that means the peer is exhibiting exactly two types of free riding. In other words, both N and C, or both N and D, or both D and C are 1. If condition C7 holds at a given moment, that means the peer is showing all types of free riding, i.e. the peer is a consumer, a dropper, and at the same time a non-contributor.

A monitoring peer may apply the appropriate counter-action policy against a neighboring peer depending on the values of N, D, and C (i.e. depending on the current condition defined by these three variables). Table 4 shows what action is to be taken against the neighboring peer under which condition. If, for example, condition C0 holds, there is no free-riding and therefore no counter-action is applied against the peer. If one of the conditions C1, C2, or C4 holds, then the level 1 counter-action is applied. If one of the conditions C3, C5, or C6 holds, then the level 2 counter actions is applied. If condition C7 holds, the level 3 counter action is applied and the peer is disconnected.

Table 4. Conditions and Counter-Actions

Conditions	Action Level	Action Description
C0	Level 0	No counter-action
C1, or C2, or C4	Level 1	Reduce TTL by more than 1
C3, or C5, or C6	Level 2	Ignore Requests Partially
C7	Level 3	Disconnect

We will represent each row at the above table with a state in the monitoring peer. When there are updates on the log table counters, the state of the monitoring peer may change, since the condition may change. In each state (i.e. each row of table above) we will apply a different counter-action to a peer. We have four states: S0, S1, S2, and S3 (as shown in Table 5). When the monitoring peer is in state S0 for a neighboring peer P, only the condition C0 may hold and no counter action is applied. When the monitoring is in state S1, one of the conditions C1, C2, or C4 may hold, and the level 1 counter action is applied.

If we state briefly, the level of counter action to be applied depends on the current state. At state S1 the level 1 counter action is applied, at state 2 the level 2 counter action is applied, and at state 3 the level 3 counter action is applied. We may have a transition between two states when the condition (values of N, D, and C) changes upon updates on the log table. The

Table 5. States, Conditions, and Counter-Actions

State	Conditions	Action Level
S0	C0	Level 0
S1	C1, or C2, or C4	Level 1
S2	C3, or C5, or C6	Level 2
S3	C7	Level 3

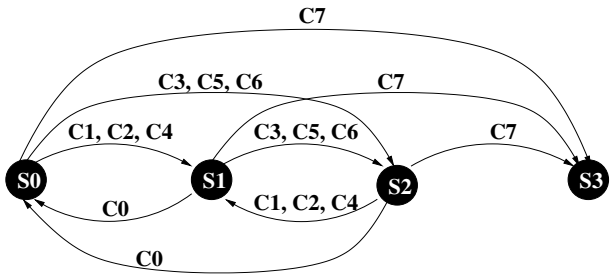

Figure 2. State diagram

Figure 2 shows the whole state diagram that shows all possible transitions. If, for example, the monitoring peer is in state S1 for a neighboring peer and an update occurs on the log table that causes a new condition to appear, the monitoring peer can make a transition to either the state S0, or the state S2, or the state S3 depending on the new condition. If the new condition is C5, the new state becomes S2; if the new condition is C7, then the new state becomes S3; if the new condition is C0, then the new state becomes S0.

On the above state diagram, if the monitoring peer makes a transition towards right, it means it is increasing the level of counter-action that is to be applied to the neighboring peer. If the monitoring peer makes a transition towards left, it means that it is decreasing the level of counter action that is to be applied to the neighboring peer.

We can express the transitions that a monitoring peer has to make upon an update on the log table and a change on the condition using an ECA rule (Figures 3, 4, 5, 6). Upon a transition, the monitoring peer moves to a new state where it applies a different counter action policy towards the neighboring peer.

4 Possible Attacks and Counter Measures

As pointed out in section 3.1, an important design issue for mechanisms against free riding is having abuse-proof property. Aiming to investigate this issue, in this section we discuss a list of possible counter attacks against free riding prevention mechanisms. We also discuss how we can defend against those kind of attacks.

Figure 3. ECA rule that governs the state transitions

define rule *StateTransition*
on update log_table($QH_P, TQ_P, ...$)
if (*State == S0*) **then**
    Execute ConditionalAction-0
elseif (*State == S1*) **then**
    Execute ConditionalAction-1
elseif (*State == S2*) **then**
    Execute ConditionalAction-2
endif
endrule

Figure 4. Contitional Action 0

ConditionalAction-0:

if ($Condition == C7$) **then**
    State = S3;
elseif ($Condition == C1$
    $\vee\ Condition == C2$
    $\vee\ Condition == C4$) **then**
    State = S1;
elseif ($Condition == C3$
    $\vee\ Condition == C5$
    $\vee\ Condition == C6$) **then**
    State = S2;
else
    Do not change state
endif

- **Fake `QueryHit` Messages:** A free rider can cheat its neighbors by replying to some queries with `QueryHit` messages fraudulently as if it has the requested file. But when the requesting peer asks for the file, it may just refuse the connection. In this way it may seem to the network as it is serving well, since neighboring peers may not be aware of unsuccessful download and cheating (download path may be different than the Query and QueryHit paths). In the log tables of the neighbors, the malicious provider may seem to be a non-free rider because of its `QueryHit` replies.

  Given the descriptors in Table 1, it may not be possible for a neighboring peer to observe and per-

Figure 5. Conditional Action 1

ConditionalAction-1:

if ($Condition == C0$) **then**
    State = S0;
elseif ($Condition == C3$
    $\vee$ $Condition == C5$
    $\vee$ $Condition == C6$) **then**
    State = S2;
elseif ($Condition == C7$) **then**
    State = S3;
else
    Do not change state
endif

Figure 6. Conditional Action 2

ConditionalAction-2:

if ($Condition == C0$) **then**
    State = S0;
elseif ($Condition == C1$
    $\vee$ $Condition == C2$
    $\vee$ $Condition == C4$) **then**
    State = S1;
elseif ($Condition == C7$) **then**
    State = S3;
else
    Do not change state
endif

ceive this kind of fake messages. Because, download occurs between two peers outside the P2P network and there is no feedback mechanism or reputation concept in unstructured P2P networks. To prevent this kind of fake `QueryHit` messages, we propose to use a new descriptor (see Table 6). The descriptor is used to report a malicious peer to its neighbor to inform that the peer is believed to be a cheater. When a querying peer is refused by a responding malicious peer during the attempt of the download, the querying peer may send a `Notify` descriptor through the P2P network to reach the 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 to only one neighbor which has delivered the `QueryHit` message, containing the IP of denying peer. Any intermediate peer on the way to the last peer, forwards the `Notify` message to only one neighboring peer based on the `Query` Descriptor Id. The last peer is the neighbor of the suspected peer. It checks and logs the `Notify` message and takes necessary actions against the suspected malicious peer. The `Query` descriptors are stored in the network for some time to route `QueryHit` descriptor in the same backward path. Therefore, we do not enforce to store new information on the peers. However, the time for deleting records from routing tables should be extended such that an unsuccessful download attempt can be executed.

Table 6. New Protocol Descriptor

Descriptor	Description
`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.

There could be some side effects of the proposed `Notify` descriptor. As stated before, small number of peers provide large amounts of files in the system and they are posed to heavy download traffic. Furthermore, some peers have very limited upload bandwidth. These peers may refuse more connections when they reach the maximum number of connections. If a download from such a peer can not be initiated, submitting a `Notify` descriptor for this peer would be unfair and incorrect. To hinder these kinds of false notifications, we propose to use a ratio of the `Notify` messages to `QueryHit` messages for a given peer. If it exceeds a predefined parameter, e.g. 80%, then proper actions can be taken against the peer.

- **Fake Files**

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

5 Conclusion

In this work, we have proposed a distributed and measurement based method to reduce the degree of free riding in unstructured P2P networks. We have first specified possible free riding types and counter actions that can be taken against free riders. Then, we have proposed mechanisms which can detect free riders and employ counter actions against them. Furthermore, we have combined these mechanisms into

a formal framework by using an RCA rule and finite state machine showing what kind of counter action is to be applied under which condition. The mechanisms proposed for reducing the amount of free riding meet the essential requirements of P2P paradigm, such as distributed computing, anonymous connections, unreliable connections, and so on. We have also proposed a new descriptor (a Notify message) to be used against the counter attacks by malicious peers to the proposed mechanisms.

By reducing the amount of free riding in a P2P network, we expect also 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 network peers and elements, and the scalability of the network.

We are currently developing a simulation program to implement and evaluate the proposed system and different mechanisms proposed to handle free riding problem in P2P networks.

References

[1] K. Aberer and M. Hauswirth. An overview of peer-to-peer information systems. *WDAS*, 2002.

[2] E. Adar and B. A. Huberman. Free riding on gnutella. *http://www.firstmonday.dk/issues/issue5_10/adar/*, 2000.

[3] Clip2. The gnutella protocol specification v0.4 (document revision 1.2). *http://www9.limewire.com/developer/gnutella protocol0.4.pdf*, Jun. 2001.

[4] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. *Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload*. In the Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003.

[5] M. Jovanovic, F. Annexstein, and K. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. *Technical Report, University of Cincinnati*, 2001.

[6] E. P. Markatos. *Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella*, pages 65–74. In the Proceedings of the second IEEE International Symposium on Cluster Computing and the Grid, May 2002.

[7] L. Ramaswamy and L. Liu. Free riding: A new challenge to peer-to-peer file sharing systems. *36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track7,Big Island, Hawaii*, January 2003.

[8] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal special issue on peer-to-peer networking*, 6, 2002.

[9] J. Ritter. Why gnutella can't scale. no, really. *http://www.darkridge.com/ jpr5/doc/gnutella.html*, February 2001.

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

[11] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A secure economic framework for p2p resource sharing. *In Proceedings of the Workshop on the Economics of Peer-to-Peer Systems*, June 2003.