

ARA: A Robust Audit to Prevent Free-Riding in P2P Networks *

MyungJoo Ham and Gul Agha
Open System Laboratory, Department of Computer Science
University of Illinois at Urbana-Champaign
{ham1, agha}@cs.uiuc.edu

Abstract

A number of solutions have been proposed to address the free-riding problem in peer-to-peer file sharing systems. The solutions are either imperfect—they allow some users to cheat the system with malicious behavior, or expensive—they require human intervention, require servers, or incur high mental transaction costs. We propose a method to address these weaknesses. Specifically, we introduce a utility function to capture contributions made by a user and an auditing scheme to ensure the integrity of a utility function's values. Our method enables us to reduce cheating by a malicious peer: we show that our approach can efficiently detect malicious peers with a probability over 98%.

1. Introduction

Since the introduction of Napster, many peer-to-peer (P2P) file sharing systems have emerged. It is estimated that there are about 60 million users of P2P systems in the US [11] and that such systems consume up to 80% of the network traffic [18]. In this paper, we focus on *pure P2P systems*, i.e., systems which do not have any central nodes [14]. In pure P2P systems, unlike those with servers, we can not dictate the behavior of a peer.

Free-riding is a serious problem in P2P [12, 19]. Free-riding may be motivated in part by the asymmetric nature of network connections provided by many broadband ISPs: these networks provide relatively low bandwidth for outgoing traffic. However, one case study [8] shows that nearly 90% of peers in Gnutella were either sharing nothing or were sharing files that were never wanted by other users. In other words, P2P traffic is entirely concentrated on a few contributors. This suggests that free-riding is not simply

the result of a lower outgoing bandwidth. If free riders open their resources, we may increase throughput significantly [19].

Several methods have been proposed to address free-riding. Unfortunately, our experiments suggest that these methods are vulnerable to free riders who are determined to exploit the system. A key difficulty in pure P2P systems is that we do not have centralized servers. Even in P2P systems with servers, unlike the case of a conventional client-server model, these servers usually do not dictate the behavior of the peers. Moreover, the solutions proposed thus far are vulnerable to one or more of the following malicious actions: modification of a peer, throttling flows over a peer, and cracking locally saved values. We consider these types of behaviors as cases of *cheating*.

We tested eMule with servers [2] and eDonkey2000 both with and without optional servers [1]. Both systems impose a rate limit on downloading that is proportional to the user defined rate limit on uploading. Without modifying the client, we could substantially deceive both systems by throttling down outgoing traffic while setting an outgoing rate limit very high in order to get faster download. Such throttling is easily accomplished by network control software such as NetLimiter [4]. Some servers blacklisted us if we throttled excessively (e.g. 1 kBps for upload, 128 kBps for download). However, this method requires servers and the servers cannot decide exactly whether the client is throttled or there are other problems because it relies on the client (peer).

The aim of our approach is to encourage resources sharing by forcing each peer to contribute in order to be served. We achieve this by embedding an economic system into a pure P2P system. Our approach is similar to that of micropayments with servers [13]. However, we try to avoid the mental transaction costs problem [13, 23] by making the payment (*credit*) volatile and our approach does not rely on servers. Peers interested in a peer observe its behavior to detect any misbehavior. Briefly, interested peers are the peers that can be affected badly if the target peer misbehaves. Because interested peers can also misbehave, we

*The work has been supported in part by the NCASSR ONR grant N00014-04-1-0562 and the ONR MURI grant N0014-02-1-0715. We thank Myeong-Wuk Jang, YoungMin Kwon, and Sameer Sundresh at UIUC and the anonymous reviewers of the paper for their invaluable comments and advice.

utilize signatures with a private and public key system such as MD5 checksum [21] to promote integrity of information travelling between interested peers. Our approach detects misbehavior without any intervention from central servers, without side effects such as the mental transaction costs incurred by users found in other systems.

This paper is organized as follows: Section 2 describes our assumptions. Section 3 discusses the design of the credit system and the design of integrity checking methods. Section 4 shows the performance of our approach including detection rate and overhead. Section 5 briefly compares our work with other approaches. The last section concludes this paper with a discussion of future work.

2. Assumptions

Our target system is a file-sharing peer-to-peer system that does not require central nodes. It is assumed to be open so that there may be malicious but compatible peers. We also assume that we already have an appropriate P2P structure: i.e. to locate resources, a peer can simply query.

Each peer is assumed to have an *id*-like an IP address. Because information kept in peers in order to count contribution and detect misbehavior is volatile (stored for limited time) peers with DHCP do not need to suffer too much after renewing their address.

Lastly, we assume that every peer can find out the public key of any peer and that peers cannot crack encryption. We might need a public key infrastructure (PKI), but we will not discuss this issue further. This assumption does not require centralization because PKI can be distributed [24].

2.1. Peer behavior

Although there may be some peers who are giving away their resources while not demanding any resources from others, we do not take into account such generous peers because it is not harmful. Instead, we assume that each peer is selfish, which means that it would contribute only if doing so is directly beneficial to the peer itself, and not necessarily to the community (as discussed in [15]). However, we also assume that a peer does not tolerate malicious behavior on the part of other peers. In other words, each peer demands resources all the time, and in order to keep its own benefits, detects and reports misbehavior of others.

We define being *fair* as a peer being served maximally while contributing as much as it is served. More specifically, we say a system is *fairer* than another if the average ratio of contribution to consumption for a peer is closer to 1, where the average is weighted by the relative contribution of each peer.

In order to discourage cheating, we assume that blocking malicious peers with considerable probability is sufficient.

Moreover, we assume that there may be many peers which prefer to free-ride but there are not so many peers that would cheat if the cheaters are punished: specifically, the numbers immune to punishment is less than say 1% of the population (or on the order of log of the population).

In P2P systems with an economic model, mental transaction costs are a barrier to the throughput of the systems [13, 23]. It is assumed that if credit is volatile, users do not need to suffer mental transaction costs because they will try to spend credit as soon as possible before their credit values disappear.

2.2. Popularity of a stricter system

We assume that a system's strictness on sharing policy does not degrade its attractiveness to users. Thus modifying a P2P system so that it is stricter in enforcing contribution would not scale down the system. This assumption is justified by the following argument.

Suppose there are two types of peers; each peer of type *A* has a lot of resources to share and is highly motivated to contribute, and each peer of type *B* has little resources to share or little motivation to contribute. Now consider two P2P systems that are compatible with every peer: system S_1 has a naive policy while system S_2 has a stricter policy on free-riding so that peers need to themselves contribute to get access to resources. Once we get enough peers on S_2 so that S_2 starts working, group *A* in S_2 would start to be compensated by being allowed to consume more resources than *B*, which would attract more *A* peers from S_1 . Because *A* provides more resources, S_1 gets fewer resources traded even if it has many *B* peers. Thus peers in S_1 will starve, which means that S_1 becomes useless, and the remaining peers in S_1 will eventually move to S_2 to get resources. Thus, a stricter system does not need to be less popular than a naive system even if peers of type *B* would not prefer it.

This suggests that a strict system would be competitive and may be embedded in current systems without decreasing the number of peers.

2.3. Types of cheating

We target four types of cheating. Cheating is dealt with if it gives benefits to the cheaters; we assume that peers cheat to increase their benefits, not to decrease them. Note that *credit* represents the capability to consume resources. Specifically, credit values, transaction records, and lists of interested peers are the information that is shared in order to count and verify contributions. (We will define these concepts rigorously in the next section).

Exaggerated Credit : An individual peer may exaggerate credit. This requires sending incorrect information justifying the exaggerated value in order to evade detection.

Conspiracy : A cheating peer may try evade detection using collaborators. However, note that if a collaborator simply gives away its credit to another through bogus transactions, we do not regard this as a case of cheating; we simply consider it as transferring credit.

Blame Transfer : A cheater may try to blame an innocent peer in order to hide malicious peers' misbehavior. An innocent peer would announce any misbehavior that it detects. However, cheaters may try to deny this by announcing the innocent announcer is a cheater. Cheaters may also announce potential announcers as cheaters to evade detection.

Omitting Interested Peers : The last type is to send malicious lists of interested peers. The *interested peers* of peer i , discussed in the Section 3, are the peers observing peer i . A malicious peer may omit some interested peers from the malicious peer's lists of interested peers sent to other peers in order to hide inconsistency created by sending malicious information. (It turns out that adding peers which should not be in the list does not yield a benefit to the cheater and can be easily detected. Therefore, we do not consider adding non-interested peers as a form of cheating.)

2.4. Why not a qualitative approach

We are not concerned about providing fake files (content cheating). It is not feasible to measure the quality or authenticity of files without human intervention (until the development of suitable A.I. techniques). There are approaches that utilize the quality of resources – specifically, determine if it is fake – which rely on human intervention [17]. We do not try this type of approach because it is not autonomous and it requires frequent human intervention which is subjective, and users, especially in P2P, do not want to participate if it might break their anonymity. In eMule [2], eDonkey [1], and Pruna [5], they try to verify the quality through user interactions. However, even for files with many sharers, there is little participation on determining the quality of the files. Another problem is that the providers may not be responsible for bad resources; e.g. a provider might be one of the 'victims' of the fake file it provides. Often, misunderstandings and cultural difference may also introduce fake files.

3. Design

3.1. Credit system

A credit value C_i as a utility function is assigned to each peer i . With a virtual clock of each peer, $C_i(t)$ is the credit value during *period* t where period t is from $t - 1$ to t . Each peer will determine the accessibility and the priority of an incoming request based on the sender's credit value.

Let $T_{I_i}(t)$ be the amount of incoming bytes to peer i during period t and $T_{O_i}(t)$ be the amount of outgoing bytes from peer i during period t . If we let the credit value to be the uploaded bytes (*contribution*) minus the downloaded bytes (*consumption*),

$$C_i(t) = \sum_{u=0}^{t-1} (T_{O_i}(u) - T_{I_i}(u))$$

Note that we added up to $t - 1$ not t because a credit value at a certain time t depends on previous transactions, not on the transactions currently being processed.

In order to verify $C_i(t)$, we need to verify all $T_{I_i}(t)$ and $T_{O_i}(t)$ from the beginning; 0 to $t - 1$ with corresponding transaction records. As time passes, we may need an indefinitely large space to store records necessary for verifying credit. In order to limit the amount of space and time needed, we use a system-wide constant m which specifies how many recent periods are counted. The constant m makes the stored information volatile.

$$C_i(t) = \sum_{u=t-m}^{t-1} (T_{O_i}(u) - T_{I_i}(u))$$

Credit value may be interpreted as the ability to pay for consumption if providers deny resources to peers without some required amount of credit. A system-wide constant LL is defined as the lower limit for a service. If a requesting peer has credit less than LL , then it will be rejected. LL is a positive constant. Therefore, a newcomer should contribute first in order to be served. This restriction discourages *white-washing* [16] by giving a disadvantage to white-washers. LL also requires peers without proper credit to contribute so that providers can earn credit.

However, the specification of a lower limit restriction creates another problem: the system may deadlock, especially in the beginning when every peer in the system has zero credit, which is smaller than LL . Therefore, no peer can consume another's resources which in turns means that peers cannot contribute to earn credit and deadlock occurs. Even assuming that we somehow managed to start up the system, if there are some peers with enough credits, the system may suffer starvation until rich peers request resources from the poor. To remedy this, we add a condition to evaluate *effective* $LL(LL_e)$ based on actual use and set it as the lower limit for each peer. If a provider p has a credit value $C_p < LL$, then it permits incoming requester i if

$$C_i \geq LL_e$$

$$LL_e = \min(LL, k_{LL} \cdot \max(C_p, 0)) \quad , 0 < k_{LL} \leq 1$$

LL_e solves the start-up deadlock and the starvation.

The sum total of credits in a system is zero. This restricts the total throughput because many peers' credit may be under LL_e even if they have traded for a long time. If there

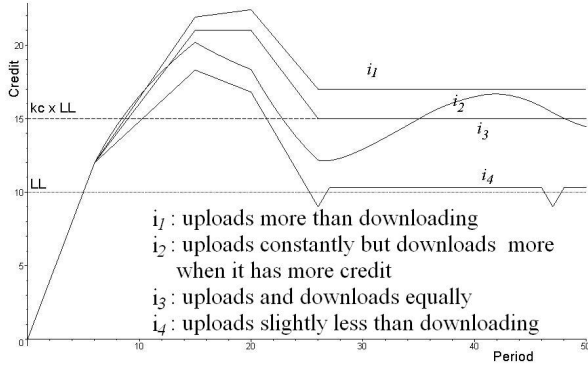


Figure 1. Credit function behavior (drawn with Maple 8 [3])

are many peers with large credits, poor peers may starve because the average of the poor peers' credit is negative. Therefore, a peer with some contribution needs to have a positive credit value larger than the LL_e even if the peer downloaded a little bit more than the peer contributed.

$$C_t(t) = \sum_{u=t-m}^{t-1} (T_{O_i}(u) - T_{I_i}(u)) + \min \left(k_C \cdot LL, k_O \cdot \sum_{u=t-m}^{t-1} T_{O_i}(u) \right) \quad (1)$$

$, k_C > 1, k_O > 0$

This constraint enables a credit value in the start-up phase to increase faster by k_O and to stabilize near $k_C \cdot LL$ if the consumption and contribution are balanced. This ensures relatively long participating peers overcome LL unless they download too fast while discouraging white-washing. The constraint also enables the total sum of the credit to be positive and increasing as the system grows.

3.2. Interested peers

The interested peers of target peer i are those who had uploading and downloading transactions with i recently, say the m most recent time intervals ($t-m \sim t-1$). These peers are interested in the behavior of i because misbehavior of i and its justification for that – modifying transaction records – may harm their interest directly. For example, if malicious peer i served 10GB to peer j during period t but peer i modifies the value to 20GB to justify exaggerated credit, then other peers checking consistency with i may think that j received 20GB from i . Therefore, those other peers may underestimate peer j 's credit or they may claim that j is malicious. This cheating harms j 's benefits so that j would not tolerate it. Thus, based on the selfishness assumption, we

can rely on the interested peers' vigilance against misbehavior. The interested peers of peer i have i as an interested peer of themselves because when peer i traded with j , j also knows that j traded with i .

As in case of transaction records and $C_i(t)$, the list of interested peers needs to be kept for the recent time m . On receipt of a request for the list of interested peers, any peer that has the information can reply; at least the target peer i has it. The list of interested peers is updated to all the interested peers at the end of each period. Each peer keeps lists of interested peers of itself and its interested peers.

We can use other ready-to-use P2P structures to locate peers with the list of the interested peers of target peer i directly, i.e., without querying the target peer or known interested peers to double check the list. If we utilize P2P structures such as CAN [20] or CHORD [22], we can verify i 's list whether i omitted some peers from the list or not; at least i 's interested peers know that they should be on i 's list and they reply to queries asking for i 's interested peers.

Peers utilize PKI to verify transaction records. A malicious peer might want to deny its transaction records stored on its interested peers, or it may alter transaction records written by other peers to blame them. However, if the records include the issuers' signature (e.g., MD5 checksum [21]), peers can ensure that the records are authentic and not deniable. To prevent replay, we include a timestamp.

3.3. Integrity Check

In order to perform an integrity check based on the information provided by the interested peers, we need to let each peer *disseminate* its information to its interested peers during operation. After each period, for each connection with peer j , peer i creates a transaction record including the ids of i and j , the amount and the direction of the transmission, and the period signed by i . This record is disseminated to all the interested peers by i . Also, peer i updates all the interested peers about its credit and its list of interested peers signed by i . They are stored for m periods. Any peer can query information about i to any of the interested peers of i . Each peer allows the lists of interested peers of itself and its interested peers to be located and queried through underlying P2P structures. Note that all p_X on the following paragraphs are probability constants used to select peers or records as samples or to execute corresponding methods.

Credit Audit : The basic credit verification method is to directly ask $C_i(t)$ from some (p_C) of the interested peers of the target i and to compare the values with the values for $C_i(t)$ announced by i . Each peer also executes this method with p_C for peer k when contacting the target peer i 's interested peer k when Transaction Record Audit is executed.

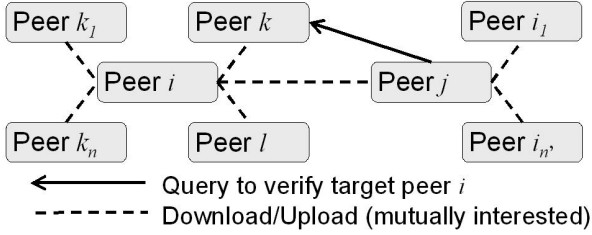


Figure 2. Relation between peers

To evade *Credit Audit*, a malicious peer i would need to modify all transaction records sent to its interested peers so that all incorrect $C_i(t)$ announced are identical and the records are locally consistent in each interested peer. However, there would be inter-peer inconsistency in the records between some of the interested peers. Because each interested peer k can instantly detect if i gives malicious transaction records between i and k , if i modified records regarding k for some of its interested peers, i should modify records related to another peer l for k . This creates inter-peer inconsistency between l and k .

Transaction Record Audit : Transaction Record Audit is executed by each interested peer of target peer i . This audit is executed by an interested peer j which verifies with other interested peers the transaction records that it received from i . The verification does not need to be an exhaustive: j needs verify the records with some sample peers. Specifically, within a period, j initiates this audit with probability p_T . Then it queries other interested peers k of i , sampled with a probability p_{TS} , about the transaction records between i and j and between i and k .

Interested Peers List Audit : The list of interested peers of i is also to be audited because i may provide malicious (omitted) lists to evade Transaction Record Audit. An investigating peer j contacts other interested peers to verify the list. It picks $n_i p_{IS}$ peers as samples where n_i is the number of interested peers of i ; query which peers have the list of interested peers of i to the P2P structure (querying who are the interested peers of i). To each chosen peer k , peer j queries whether j is in k 's copy of the list of interested peers of i . If any inconsistency is found, j receives the entire list of interested peers of i signed by i from k to verify the previous query result. This audit is initiated by any of the interested peers by a probability p_I per period. Note that if a peer denies being on the list, the target peer can show the counterexample—a transaction record signed by the denier, and if a peer still insists on the authenticity of the list, it should show a transaction record signed by its counterpart. This rule prevents evasion.

Table 1. The probability constants

p_C	<i>Credit Audit</i> is run p_C per interested peer
p_T	<i>Transaction Record Audit</i> is run p_T
p_{TS}	per interested peer picking samples with p_{TS} probability for each interested peer of the target.
p_I	<i>Interested Peers List Audit</i> is run p_I
p_{IS}	per interested peer picking samples with p_{IS} probability for each interested peer of the target.

3.4. How cheating is mitigated

We describe here how the four types of cheating described in Section 2.3 are mitigated. For the purposes of our analysis, we assume that malicious peers try to evade detection as much as possible.

Exaggerated Credit : When a peer i exaggerates one's own credit by modifying a transaction record related to a peer k , it is detected by *Credit Audit* and *Transaction Record Audit*. If the malicious peer i did not equally modify the credit value for every peer, it would be detected by *Credit Audit* with probability p_C per contacting peer and per period. If a record related to k is modified and sent to the peer k or transaction records are not modified accordingly to the modified credit value, it is detected directly. So i should modify another record for k while modifying records related to k for others. Then, any peer l who got the modified record regarding k will detect this when it contacted k with *Transaction Record Audit*, and others will detect this when they contacted l . Because each peer initiates *Transaction Record Audit* with probability p_T and the peer will pick a peer from the other group of modified records with probability p_{TS} , even if the size of the other group is only one, each interested peer has at least $p_T p_{TS}$ per period chance (some of the peers have a greater chance) of detecting this type of cheating even if the modification is minimal.

Conspiracy : Conspiracy is mitigated by *Transaction Record Audit*. Assume that there are two collaborators i_1 and i_2 . Then, i_1 will ignore i_2 's malicious behavior and i_2 will ignore i_1 's. If i_1 exaggerated its credit by modifying transaction records regarding i_2 , i_2 will report the exaggerated value if i_2 is requested for the records about i_1 , but it will report the exact value when the record is required to verify i_2 . If i_2 replies with records for the exaggerated value all the time, then it is equivalent to transferring i_2 's credit to i_1 (which we do not regard as cheating). To report differently, i_2 needs to make i_2 's interested peers disjoint from i_1 's and report the exaggerated credits to i_1 's and the exact credits to i_2 's. Otherwise, once a peer in the intersection initiates either a *Credit Audit* or a *Transaction Record Audit*, the cheating may be detected. When the two sets of

interested peers are disjoint, if a peer in the set of interested peers of i_1 or i_2 initiates Credit Audit during Transaction Record Audit, picking i_2 or i_1 as a sample respectively, then the cheating is detected; this happens because i_2 is an interested peer of i_1 and vice versa. The chance for a peer to detect this cheating during one period is at least same as the following equation 2.

$$1 - (1 - p_T \cdot p_{TS} \cdot p_C) \quad (2)$$

Note that the number of co-conspirators should be minimized to evade detection because the greater the number of collaborators, the higher the probability that one of them will be picked for audit.

Blame Transfer : A malicious action report should have at least two contradicting records signed by the malicious peer. A malicious report without proper information is trivial. This makes it even easier to detect.

Omitting Interested Peers : The last type of cheating is mitigated by an Interested Peers Audit. When a malicious peer i excluded an interested peer k – the modification is minimal when it excluded only one peer – and i reported the list of the interested peers to others without k , it is detected by initiating Interested Peers Audit by k or by others. When Interested Peers Audit is initiated by k – the chance is p_I per period - the cheating is always detected. Otherwise, the chance is $p_I p_{IS}$ per peer and per period.

3.5. Reward and punishment

Generally, peers with lots of resources are not so abundant [12] and thus many peers wait in the rich peers' queues. This long queue waiting makes higher priority in a queue very precious, which enables priority according to credit to encourage contribution. Giving higher priority in queues to 'better' peers is seen in some systems such as eMule [2] and Pruna [5].

A misbehavior report containing contradictory reports signed by the malicious peer i is sent to all interested peers of i by the peer that detected the malicious action. Upon receipt of a report, peers verify the report and put i in their local blacklists. If any query regarding i (based on one of the auditing methods) arrives at a peer j that blacklisted i , j replies with the report. An upload request from blacklisted peer i is denied because its credit income can no longer be trusted. In response to such requests, the receiver informs the recently interested peers of i which may be unaware that i is blacklisted. Therefore, i remains starved; this mechanism results in a permanent outcasting of malicious peers.

4. Performance Analysis

We have studied the performance by execution by Maple 8 [3] running on Windows XP. The following parameters were used. Each period was 10 minutes long. Recent time m was 12. The number of interested peers, n_i , is 50. Bandwidth for a peer is 1 Mbps/128 kbps for down/up streams which is similar with common broadband services. The size of a signature is 20 bytes, and that for a data element is 4 bytes.

4.1. Detection probability

We assume that a cheater is smart enough to minimize the detection rate, so it minimizes the number of modified records as discussed in the Section 3.4. We describe the detection rate of one period in this section. However, because m is larger than one, integrity checking is executed by repeatedly increasing the rate.

Exaggerated Credit : At first, for this type of cheating, we have two disjoint groups of interested peers for the malicious peer i ; group A with a modified record related to a peer k_B in B and group B with a modified record related to a peer k_A in A. A is the number of peers in group A. Not to be detected during a period by all the interested peers by Transaction Record Audit, each peer in A should not check B and vice versa. The probability to do so is

$$\begin{aligned} Evade_1 &= (1 - P_B)^{n_i - A} \cdot (1 - P_A)^A \\ P_B &= p_T \cdot \left(1 - \frac{(n_i - A)! \cdot (n_i - n_i \cdot p_{TS})!}{n_i! \cdot (n_i - A - n_i \cdot p_{TS})!} \right) \\ P_A &= p_T \cdot \left(1 - \frac{A! \cdot (n_i - n_i \cdot p_{TS})!}{n_i! \cdot (A - n_i \cdot p_{TS})!} \right) \\ \forall x < 0, \quad x! &= 1 \end{aligned}$$

where P_A is the probability to be detected by a peer in A during one period and P_B is same for B. We do not consider the Credit Audit here because this auditing method cannot detect an exaggerated credit if the exaggerated credit values are identical. A should be set to 1 or $n_i - 1$ in order to maximize $Evade_1$. Therefore, we assume that the size of a cheaters set A as 1. Note that P_A and P_B are symmetric in terms of A and $n_i - A$.

Conspiracy : To fail detection during one period, every peer initiated Transaction Record Audit to a collaborating peer should not have initiated a Credit Audit during the same period. The probably of such initiation is p_C ; thus the probability for such failure per one period is:

$$Evade_2 = (1 - p_T \cdot p_{TS} \cdot p_C)^{n_i}$$

Blame Transfer : This type of cheating is skipped because it is easily detected at once (as discussed in Section 3.4).

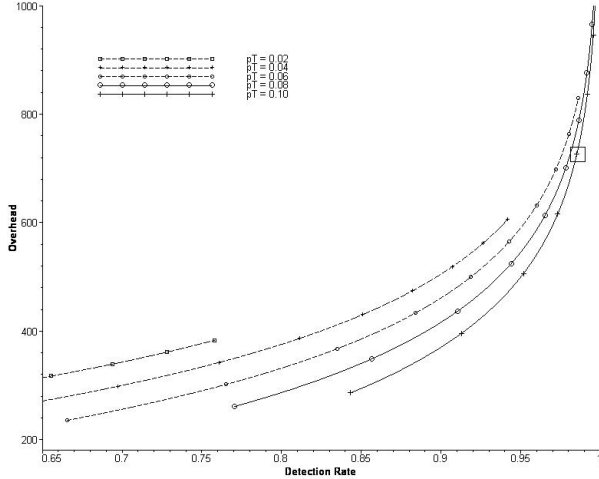


Figure 3. Detection rate for Exaggerated Credit with Credit Audit and Transaction Credit Audit. Overhead in bytes per period.

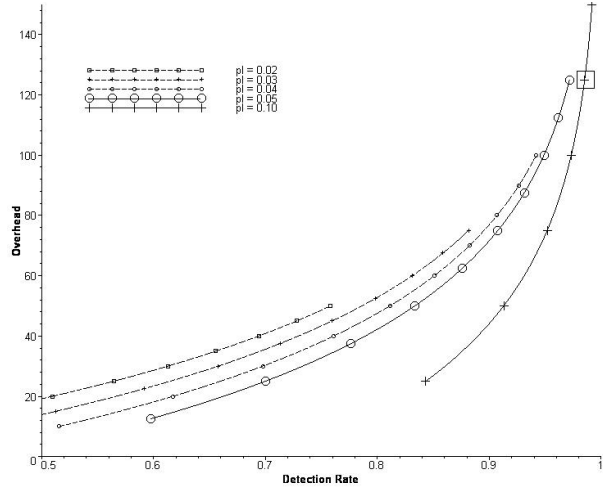


Figure 4. Detection rate for Omitting Interested Peers with Interested Peers Audit. Overhead in bytes per period.

Omitting Interested Peers : This type of cheating is undetected if every peer fails to detect and the omitted peers do not execute Interested Peers Audit. When a malicious peer i omits a peer from the list to minimize the detection rate, the probability that this omitting is undetected during a period is

$$Evade_4 = (1 - p_I \cdot p_{IS})^{n-1} (1 - p_I)$$

4.2. Overhead

We count communication overheads caused by two sources. The first overhead is created by information dissemination. The second overhead is required by the three auditing methods. Overhead is discussed in terms of how many bytes of data sent and received for a peer per period. However, we ignore the overhead resulting from other network specific factors, e.g. from the headers of each packet.

The first kind of overhead consists of credit value announcements (28 bytes), transaction records ($32n_i$ bytes), and the list of interested peers ($24 + 4n_i$ bytes) for each interested peer. This takes 1852 bytes per period per an interested peer, and adds up to 92600 bytes, which is amortized to about 154 bytes per seconds or $<1\%$ of outgoing bandwidth.

The second overhead occurs during integrity checking and is determined by the probability constants of the four methods. Credit Audit needs 28 bytes per reply and 4 bytes per request, and is executed $n_i \cdot (p_C + p_T \cdot p_C)$ times for per period. Transaction Record Audit needs $40n_i \cdot p_{TS}$ bytes for reply and $4n_i \cdot p_{TS}$ bytes for request, and is executed $n_i \cdot p_T$ times per period. Interested Peers Audit needs $2n_i \cdot$

p_{IS} bytes for reply and $8n_i \cdot p_{IS}$ bytes for request, and is executed $n_i \cdot p_I$ times per period.

We can see the trade-off relation between the overhead and the detection rate in Figures 3 and 4. Figure 3 shows the relation between the detection rate for Exaggerated Credit and the overhead from Credit Audit and Transaction Record Audit. p_C is 0.1, p_{TS} is in the range of 0.01 to 0.10 increases by 0.01. p_T is in the range of 0.02 to 0.10 increases by 0.02. Each line in Figure 3 shows overhead-detection rate relation with constant p_T . On each line, the bottom point has $p_{TS} = 0.01$ and increases by 0.01. Note that some points are out of the graph. In Figure 3, the point chosen with a box shows that the overhead per period per peer is 726 bytes (1.3 bytes/sec or $<0.01\%$ of outgoing bandwidth) for detection rate per fault of 98.5% (29.4% per period). This is done by settings of $p_{TS} = 0.05$ and $p_T = 0.10$. With this setting, the detection rate for Conspiracy is 97.5% per period and approximately 100% total.

For Omitting Interested Peers, Figure 4 shows the same metrics for an Interested Peers Audit. Each line shows settings with the same p_I value. Parameter setting for p_I is 0.02, 0.03, 0.04, 0.05, and 0.10 from the leftmost line. p_{IS} is 0.01 to 0.10 from the bottom step by 0.01. Some points are out of the graph. The chosen point with a box has settings of $p_I = 0.10$ and $p_{IS} = 0.5$ and shows overhead of 125 bytes per period and a detection rate of 98.5% in total (25.7% per period).

Therefore, the detection rate is 98.5% or more with the second overhead of 851 bytes/period (1.42 bytes/sec) with the example settings. For a broadband user, this overhead is negligible and it does not increase as the bandwidth in-

creases. Further research is needed to reduce the relatively high overhead for information dissemination (the first kind of overhead) which takes 154 bytes/sec and has much redundancy.

5. Related Work

Several approaches against free-riding and cheating in P2P systems have been established in both existing systems and research. eDonkey [1], eMule [2], and Pruna [5] have download and upload rate limits which basically throttle the bandwidth for peers. A peer needs to allocate more outgoing bandwidth to get more incoming bandwidth. However, a user might alter the application or simply throttle the outgoing bandwidth using another application [4] making it easy to get more resources than one deserves. When servers are used, we can blacklist peers that seem to be suspicious, but this is a limited solution. Also when information about a peer is stored at that peer, such information can easily be exploited by users, as in Pruna [5]. Another approach is to count all contributions and consumption via micropayments with servers [16]. This strategy is used in the V-share P2P system [6] with servers, but the mental transaction costs problem may limit the throughput realized [23]. In [17], nodes rely on the authenticity of resources to determine a reputation value, which needs human intervention and might not work in some cases, as mentioned in Section 2.5. There are some interesting approaches to encourage contribution by providing users with mental reward according to their actions [10], but we do not want the system to rely on human interactions.

6. Conclusion

To address free-riding problems, we have introduced a credit system to encourage contribution while minimizing side-effects due to mental transaction costs. Moreover, we have described methods to keep the credit values consistent with malicious peers and without servers and human intervention. Our method is unique in the sense that it does not require human intervention or servers to prevent free-riding and it is not prone to malicious peers.

In future research, we plan to implement an agent or actor simulation based on evolutionary prisoners dilemma (EPD) [9]-like game theory to see how potentially malicious peers will react in the system and how the performance-overhead, detection rate, and throughput-changes. This will also enable us to compare the fairness of our system against others. We expect to decrease overhead by reducing redundancy in information dissemination. Moreover, we intend to remove dependency on PKI to make our scheme feasible on less powerful nodes such as MOTES [7]. Evaluating the performance under the dynamic environment of P2P

and improving our approach accordingly would be following this work.

References

- [1] eDonkey. <http://www.edonkey2000.com>.
- [2] eMule. <http://www.emule-project.net>.
- [3] Maple. <http://www.maplesoft.com>.
- [4] NetLimiter. <http://www.netlimiter.com>.
- [5] Pruna. <http://www.pruna.com>.
- [6] V-Share. http://www.gample.net/V6/download_share.htm.
- [7] Wireless Sensor Network (MOTES). http://www.xbow.com/Products/Wireless_Sensor_Networks.htm.
- [8] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.
- [9] R. M. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, NY, 1984.
- [10] H. Bretzke and J. Vassileva. Motivating cooperation on peer to peer networks. In *User Modeling*, pages 218–227, 2003.
- [11] N. Coleman. The riaa and the music piracy debate. *Washingtonpost.com*, Oct 2003.
- [12] M. Fischmann and O. Gunther. Free riders: Fact or fiction? Sep 2003.
- [13] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *ACM Conference on Electronic Commerce*, pages 264–267, 2001.
- [14] R. L. Graham. Toward a Definition for Pure Peer-to-Peer. P2P2001, Apr 2001. <http://www.ida.liu.se/conferences/p2p/p2p2001/pure.html>.
- [15] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, Dec 1968.
- [16] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for Cooperation in Peer-to-Peer Networks. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, California, USA, June 2003.
- [17] S. Marti and H. Garcia-Molina. Limited reputation sharing in p2p systems. In *ACM Conference on Electronic Commerce*, pages 91–101, 2004.
- [18] A. Parker. The true picture of peer-to-peer filesharing. Technical report, CacheLogic, Jul 2004.
- [19] L. Ramaswamy and L. Liu. Free riding: A new challenge to peer-to-peer file sharing systems. In *HICSS*, page 220, 2003.
- [20] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.
- [21] R. L. Rivest. *RFC1321: The MD5 Message-Digest Algorithm*. MIT Laboratory for Computer Science and RSA Data Security, Inc, Apr 1992.
- [22] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.
- [23] N. Szabo. Micropayment and mental transaction costs. *The 2nd Berlin Internet Economics Workshop*, May 1999.
- [24] S. Yi and R. Kravets. Moca: Mobile certificate authority for wireless ad hoc networks. *The 2nd Annual PKI Research Workshop*, 2003.