HAL
open science

# Incentive for P2P Fair Resource Sharing

Emmanuelle Anceaume, Maria Gradinariu, Aina Ravoaja

## To cite this version:

Emmanuelle Anceaume, Maria Gradinariu, Aina Ravoaja. Incentive for P2P Fair Resource Sharing. The fifth IEEE International Conference on Peer-to-Peer Computing, Sep 2005, Konstanz/Germany, Germany. inria-00000312

**HAL Id: inria-00000312**

**https://inria.hal.science/inria-00000312**

Submitted on 23 Sep 2005

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Incentive for P2P Fair Resource Sharing

Emmanuelle Anceaume
*IRISA, Campus Universitaire de Beaulieu, Rennes, France*
*anceaume@irisa.fr*

Maria Gradinariu
*IRISA, Campus Universitaire de Beaulieu, Rennes, France*
*mgradina@irisa.fr*

Aina Ravoaja
*IRISA, Campus Universitaire de Beaulieu, Rennes, France*
*aravoaja@irisa.fr*

***Abstract*— We consider the problem of Fair Resource Sharing to optimize the performance of resource sharing in peer to peer systems. Resource sharing systems currently face rational peers which may exhibit a variety of strategies including : *no participation*, also referred as free-riding, and *greedy behavior*. The first aspect has been extensively studied in the late years, while the second one has not received much attention. The broad class of proposed solutions focus on designing incentives to reward cooperative peers. The side effect of these incentives is twofold : the system load is not balanced and the resource potential of the system is not fully exploited. The P2P fair resource sharing aims at both balancing the load and maximizing the use of system resources.**

**The contribution of our work is twofold. First, we specify the P2P Fair Resource Sharing problem and propose a mechanism to solve it in large scale dynamic networks with rational users. Our mechanism is composed of a novel incentive (i.e. fair cooperation) and an algorithmic part encapsulated in a middleware layer. Second, we propose an architecture for our mechanism middleware layer including four distributed services that bring together several research area : aggregation, semantic group membership and tracking. Finally, we implement our mechanism using a peer-to-peer unstructured model and evaluate it through simulations.**

## I. Introduction

In peer-to-peer (P2P) systems resources are highly distributed and stored by individual peers. In such environments where each peer may present a different self-interested entity, designing algorithms for sharing resources becomes a difficult task. Traditional distributed algorithms design faces only two types of behaviors : *obedient* (i.e. to follow the algorithm) or *malicious* (i.e. to "play" against the others). In contrast, in P2P systems users may have a strategic behavior which may be neither obedient nor malicious, just rational. It was established in [1] for example that in the absence of incentives a large fraction of peers have *a priori* no motivation to share while they are naturally incited to use system resources. The effects of "free-riding" (consuming resources without contributing) was extensively studied in the last years [1], [13]. This behavior can cause severe degradation of the system performances or even may lead to the system collapse [14]. As emphasized by Shneidman and al. [14], different approaches may be adopted to face rationality : *i)* to ignore it and to expect that system will do its best despite self-interested peers, *ii)* to limit the effect that a rational peer can have on the system by using trusted mechanisms, or *iii)* to adopt the fault tolerance techniques. However, none of these approaches benefits from resources that may be potentially offered by these rational and self-interested peers. Thus the system must motivate each peer to behave rationally in a system efficient way. Solutions come from economics and more precisely from the mechanism design theory that always stressed for algorithms that work correctly in the presence of predictable selfish behavior.

In this work we address the Fair Resource Sharing problem. Essentially, this problem aims at optimizing the resource sharing in P2P systems where peers exhibit rational behavior. The mechanism we propose to solve this problem comprises a fair differential service incentive for motivating peers to cooperate and an algorithmic ingredient encapsulated into a distributed middleware layer. The incentive ingredient eventually harnesses the resources of all the peers by motivating free-riders to change their strategy (peers that contribute more get better quality of services) while encouraging greedy peers to share their requests with less solicited ones. The middleware layer is composed of four services : *i)* The Registration service, which is used by each new peer to advertise the system of its resource potential ; *ii)* the Semantic Group Membership service, which self-organizes peers into semantics clusters to maximize a fair distribution of requests among peers having similar resources but dissimilar popularity ; *iii)* the Tracking service whose role is to track free-riders within a given semantic group, and *iv)*, the Aggregation service which combines the actions of each peer to evaluate his behavioral functions. All these services, bringing together for the first time distinct research areas (i.e. semantic group membership, aggregation and tracking) are distributed

and implemented using peer-to-peer tools. The engineering novelty of our mechanism design is twofold. First, services are implemented within a middleware layer, limiting the impact of malicious peers, and second, the modular conception of our mechanism, that abstracts the incentives from the services needed to implement these incentives can be further exploited in the design of a generic middleware for incentives implementation in P2P systems.

The rest of the paper is organized as follows. Section II discusses some related work. In Section III we present the system model and the specification of the Fair Resource Sharing problem. Section IV presents the fair cooperation mechanism. Section V presents the architecture of fair cooperation mechanism middleware layer. Simulation results are presented in Section VI. Section VII concludes.

## II. Related Work

A broad class of differential service incentives have been recently studied in order to convey the free-riding problem in P2P resource sharing [8], [2], [5]. A first attempt to formally prove the efficiency of such differentiation mechanisms, using a game theoretic framework, has been proposed by Buragohain et al. [6]. Their model has then been revisited and enriched by Feldman et al. [8] in order to study the free riding and the whitewashing problems (peers that change their identities in order to escape from the consequences of their past). The latter takes into account the generosity levels of the users. They then evaluate the efficiency of a penalty mechanism under these settings. These models, however make implicitly the assumption that each peer scrupulously follows the specification of these mechanisms and, especially, report truthfully their own contribution. Ngan et al. [10] propose an auditing mechanism based on top of a Chord overlay to ensure that peer report truthfully their real contribution. In our work, we don't expect nodes to report their own contribution but, instead, consider the near past interactions of nodes and evaluate their contribution through this. Banerjee et al. [4] consider a reciprocation strategy and use expected utility, based also on past interactions, as a decision function. However, the collection and maintenance of the history of the past interactions between peers is not precised.

A very close work to ours is that of Triantafillou et al. [11], [12]. Their system is build on top of a DHT overlay. The key idea of their solution is that each time a peer accesses a resource shared by another one, the former owes the latter a *favor*. Each peer then redirects the requests for the same resource to those from which he has get favors. These favors are also used to compute the generosity level of the peers and to differentiate the service in proportion with the generosity of each node. However, their work lacks a clear formulation of the fairness problem.

## III. System Model and Problem Specification

### A. Communication Model

We assume a large finite, yet unbounded dynamic set of users, also referred to as *peers*. The set is dynamic in the sense that peers can join or leave at any arbitrary time.That is, peers can leave or join the system arbitrarily often, and they can fail temporarily (transient faults) or permanently (crash failures). Each peer is associated with an unique identifier. Peers can communicate with each other over a *network* providing a best-effort datagram service, similar to the Internet. In other words, most messages are delivered unless either the sender or receiver fails beforehand. Yet, neither the message delivery nor ordered delivery of messages is guaranteed.

### B. P2P Resource Sharing Model

We assume that each peer in the system owns a set of resources declared as shared within the system. A resource is characterized by a type and a quantity available at some time $t$. Peers ask shared resources via *request messages*. A request message includes the requester identifier, the type and the quantity of the resource. We say that a peer *matches a request* if it owns a resource matching the type of the requested resource. A peer *satisfies a request* if it matches the request and if it provides this resource to the requester or forward it to a less demanded peer. Note that, even when the necessary quantity of resource is only partly available on a given peer, any request that matches this resource is taken into account.

Each peer in a resource sharing system gains a certain benefit $b$ from using the system and pays a certain cost $c$ participating to it. More precisely, $b(p, r(t), t)$ is the benefit that peer $p$ gains for each remote resource $r(t)$ used at time $t$ [1], and $c(p, r)$ represents the cost induced to $p$ for each unit of its accessed resource of type $r$. These two parameters have a strong influence on the utility of a peer, that is the benefit derived from its interaction with the other peers in the system. This utility depends on two additional parameters : The participation level of a peer $p$, $part(p, r, t)$, which represents the quantity of resource of type $r \in \mathcal{R}$ that $p$ has offered to the system until time $t$. The access level of $p$, $al(p, t)$, which is the fraction of resource of the system that $p$ has access at

[1] $p$ connects at time $t = 0$.

time $t$. Formally, the utility $u(p,t)$ of peer $p$ at time $t$ is defined as :

$$u(p,t) = \int_{t'=0}^{t} b(p, r(t'), t')al(p,t') - \sum_{r \in \mathcal{R}} c(p,r) * part(p,r,t). \quad (1)$$

### C. Peers Behavior

A peer-to-peer resource sharing system is made of autonomous, rational and strategic peers that interact among each others. These peers are rational because they wish to maximize their utility (by accessing limited but sharable resources), and they are strategic because they can choose the actions that minimize their participation (*free-riding* strategy) or maximize their access level (*greedy strategy*). Both strategies have dramatic consequences on the system welfare. An emergent property of a resource sharing system where nodes are neither free-riders nor greedy is *fairness*.

### D. P2P Fair Resource Sharing Problem

Formally, the fair resource sharing problem is characterized by the following properties :

**Requests Sovereignty** Eventually, any request sent by a non-free rider is satisfied if the requested resource is available in the system.

**Peer Sovereignty** Any peer is allowed to request a resource infinitely often.

**Peer Fairness** If a request is sent infinitely often then it is received infinitely often by any peer that matches it.

**Peer Cooperation** If a non-free rider peer receives infinitely often a matching request then he satisfies it infinitely often.

## IV. Principles of our Solution

To motivate players to cooperate in a fair resource sharing system, we propose to extend the classical differential service incentive model to the fair differential service incentive.

A first attempt in designing differential service incentive mechanism is the following : to gain full access to the system resources, a peer has to provide a fixed proportion of its own resources, otherwise he obtains only a small fraction of these resources. Courcoubetis et al. [3] have shown that this fixed fee-based mechanism is sufficient to ensure a social welfare in large scale social networks, however it is adapted to the so-called one shot games only. That is, games in which each peer chooses an action to take (for instance, to provide a resource or not), once for all. In our context, a peer may decide to unilaterally switch his strategy in the course of the game if this improves his utility. An example of such behavior is the whitewashing phenomenon [7] characterized by the fact that free-riders repeatedly masquerade as newcomers by obtaining new identities at low or no cost prior to each request. Clearly, whitewashing enables a peer to escape the consequences of his past. Hence, the cooperation incentive has to adapt to these changes. That is, the system should be able to detect such strategy changes and act accordingly.

The differential service incentive basically increases the access level of a peer according to the participation level. However an adequate relationship between these two parameters is important since by imposing a too high restriction on the access level peers may be discouraged to join the system, while relaxed participation fees would encourage free riding. The side effect of this mechanism is to encourage selfishness by tempting peers to increase their participation level in order to increase their access level. Clearly, this may considerably augment the popularity of some peers at the expense of unpopular or newcomers peers that will be bound to remain isolated because requests are preferably sent to popular peers. To circumvent this islanding process, we extend the differential service mechanism with fairness incentives. Essentially, this mechanism motivates peers to forward some of the received requests to less solicited peers whenever he considers that his current level of participation is high enough with respect to his neighbors.

The fair differential service incentive we propose, referred in the following as *fair cooperation incentive*, aims at ensuring that whenever a peer behaves as required (formally defined below) then he may access all the resources he wishes in the system, while whenever a peer changes his strategy, his access level is modified accordingly. To ensure such properties, the behavior of a peer is evaluated when he joins the system, and each time its suspicion level reaches an upper bound *susp_max*, that is, each time a peer is detected as free-rider. As will be described later on in the implementation part, this evaluation consists in sending several requests in the raw to the tested peer, and to compute his access level according to his behavior upon receipt of these requests. Senders of these requests are chosen within the semantic neighborhood of the tested peer which guarantees that the requests they send match the profile of the tested peer. Neither the tested peer nor the testers are aware of the fact that a test is running. This is socially beneficial and prevents free-riders from changing their strategy during the test period. Peers keep their access level until the next test (if any). Clearly, this is a good argument to motivate peers to quickly change their strategy whenever they realize that their access level have dropped. More

precisely, let $al_0$ be the initial access level of peer $p$ when it enters the system, $R$ be the number of matching requests $r$ sent to the evaluated peer $p$, and *part(p,r,R)* be $p$ participation level during the test, then the access level of $p$ at time $t$, $al(p,t)$, is given by the following equation :

$$al(p,t) = \begin{cases} al_0 + (1 - al_0)\frac{2*part(p,r,R)-R}{R} & \text{if } part(p,r,R) \neq 0, \\ \epsilon & \text{otherwise.} \end{cases} \quad (2)$$

Then, if $p$ accepts the $R$ matching requets, that is $part(p,r,R) = R$ his access level will increase to 1. Similarly, if he rejects all his incoming requests $part(p,r,R) = 0$, this will be set to $\epsilon$.

Recall, that another problem faced by the resource sharing systems is *whitewashing*. In networks where identity is free, peers may change their identity over time to escape the consequence of their past by disconnecting and reconnecting. To prevent such behavior, an *entry fee* has to be imposed to newcomers. Thus, peers have no incentive to change their identity since they would have to pass this entry fee at each reconnection. This is the reason why, the access level of each newcomer is initialized to $al_0$. The value of $al_0$ is set by the system designer, and has to be low enough to avoid whitewashing but sufficient high to not discourage peers to participate to the network. This value is set to $al_0 = \frac{1}{2}$ [2]. The access level of a free rider drops to a positive value $\epsilon$ to incentivize him to remain in the network, and eventually, change his strategy.

We now describe the strategies that increase peers utility :

**Cooperative Strategy.** The strategy adopted by a peer upon receipt of a matching request consists in satisfying the request according to the requester strategy and his participation level with respect to his neighborhood. This leads to the following acceptance rule :

**Rule 1** (**$\nu$-request acceptance**) Let $r \in \mathcal{R}$ be an available resource at peer $q$ at time $t$. Upon receipt a matching request for $r$ sent by peer $p$, $q$ satisfies it with probability $f_q(p,t)$ if $\forall s \in \mathcal{N}_q, part(q,r,t) - part(s,r,t) \leq \nu$, where $\nu$ is an application dependent constant.

By setting $f_q(p,t) = al(p,t)$, each peer has access to a fraction $al(p,t)$ of the system resources.

**Fair Strategy.** The strategy adopted by the recipient of a request that considers himself as busy enough consists in sharing his load with one of his neighbors. This leads to the following fairness rule :

**Rule 2** (**$\nu$-forwarding rule**) Upon receipt of a matching request at time $t$, peer $p$ forwards it to peer $q \in \mathcal{N}_p$ if $part(p,r,t) - part(q,r,t) \geq \nu$, where $\nu$ is an application dependent constant.

This rule decreases the charge imposed on the currently too solicited peers by allowing newcomers or unpopular peers to benefit from it.

We can now precisely define a fair cooperative peer : *Definition :* A peer $p$ is a **fair cooperative peer** if, upon receipt of a matching request, $p$ executes either Rule 1 or Rule 2.

Our incentive mechanism is characterized by the following properties [3] :

- **Cooperative Peers Rewarding Property :** Eventually, the expected access level of a fair cooperative peer equals 1.
- **Non-Cooperative Peers Discrimination Property :** Eventually, the exptected access level of a non-cooperative peer equals $\epsilon$.
- **Fairness Property :** Eventually, the participation level of any two fair cooperative neighbors are $\delta$ apart, where $\delta$ is some fixed constant.

Thus, a free-rider refers to a non fair cooperative peer. Hence, we extend the classical notion of free-riding to non cooperation and greedy behavior.

## V. Architecture to Implement Fair Resource Sharing Incentives

In this section we propose an architecture that implements our fair cooperative incentive mechanism. The architecture is built on top of a supervising overlay. Members of this overlay, *Supervisors*, are trusted players whose role is to verify/test the "good behavior" of each peer. Supervisors self-organize to form an overlay that "cover" all the peers of the system (see Figure 1). The choice of the trusted peers and their organization is not the focus of the current work.

This supervision overlay is the back-bone of the following distributed services (Figure 2) :

The *Registration Service* assigns to each resource a supervisor.

The *Semantic Group Membership Service* self-organizes peers owning similar resources. Each group includes a supervisor.

[2] Friedman et al [9], use a periodic-based model. If, after each period, there is a fraction $\alpha$ of identity changes (turnover rate), for high number of peers, and with a low probability of messages loss, the entry fee has to be set to $\frac{1}{(1-\alpha)(2-\alpha)}$, if the turnover rate satisfies $\alpha \leq \frac{3-\sqrt{(5)}}{2}$, otherwise, whitewashing problem cannot be solved. Here we choose the best case where the whitewashing probability is close to 0.

[3] The two first properties depict the service-differentiating nature of our system. According to the rationality principle which is the main assumption on the behavior of the users, non-cooperative peers, eventually, either change their strategy to a cooperative one or at worst disconnect from the network.

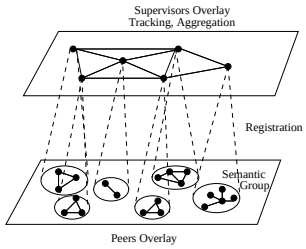


Fig. 1: Middleware Overview.

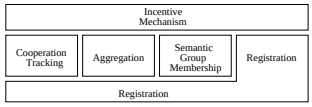


Fig. 2: Middleware Architecture.

The *Cooperation Tracking Service* monitors peer cooperation level.
The *Aggregation Service* combines information related to peers participation and their access levels.

### A. Orchestrating the Services to Fair Cooperation

In this section, we provide a generic orchestrating protocol for implementing the fair cooperation mechanism (the two Rules described in Section IV). The pseudocode of this protocol partially appears in Figure 3.

When a site $p$ enters the system, it first invokes the `initiate-join` method with the list of resources it owns and it wishes to share with the system. This function first registers $p$ and provides for each resource $p$ owns, the supervisor in charge of it. Note that if the newcomer $p$ is a supervisor, then the list of resources he is in charge of is forwarded to his application. Then, for each local resource $r$ at $p$, this function instantiates a semantic group object and settles a semantic group of similar sites $q$ (that is, the list of sites that locally own resource $r$) using the semantic group primitive `Join`. This neighborhood list is forwarded to the application. Note that the neighborhood list is continuously updated. Then an aggregate and a tracking objects are instantiated.

Once registered, sites can ask for resources by a call to `send-request` with the type of resource they wish to get[4], and the recipient of the request. The recipient is the one to which the request is sent unless another site is under test, in which case it is privileged to receive the request. This makes the incentive test transparent to both the tested site (a tested site, upon receipt of a request, cannot distinguish whether the received request is part of a test period or not) and the testing sites (the sites, chosen

[4]Note that for readability reasons, the attributes of the requested resources are omitted.

**Upon** `send-request`($r$,$p_n$) from application **do**
  $idreq := idreq + 1$;
  timer :=$T0$;
  **if** ($\exists p$ such that *tested*$^{p}_{r.type}$=TRUE) **then**
    $p_n := p$; (* *p is priviledged as recipient of the request**)
  send (`Request`,$idreq$,$r$,$myId$) msg to $p_n$
  **while** (TRUE) **do**
    **wait until** (`response` or `fwrd` is received) or timeout
    **if** receive (`Fwd`,$idreq$,$r$,$p_j$) msg from $p_n$ **then**
      *aggregate*$_{r.type}$.`GetPart`($idreq$,$p_j$,$p_n$,$part_j$,$part_n$)
      **if** ($part_n - part_j \leq \nu$) **then**
        *aggregation*$_{r.type}$.`GetAcc`($idreq$,$myId$,*prob*);
        *tracking*$_{r.type}$.`IndStatus`($idreq$,$p_n$,FALSE,*prob*);
        **deliver** (RESPONSE,$r$,$\perp$,$p_n$) to application;
        **return**;
      **else**
        *tracking*$_{r.type}$.`IndStatus`($idreq$,$p_n$,TRUE,$\perp$);
        *aggregation*$_{r.type}$.`CompPart`($idreq$,$p_n$);
        timer:= $T0$;
        $p_n := p_j$;
        **exit** from **wait until** loop
    **else**
      **if** receive (`Response`,$idreq$,$res$) msg from $p_n$ **then**
        *tracking*$_{r.type}$.`IndStatus`($idreq$,$p_n$,TRUE,$\perp$);
        *aggregation*$_{r.type}$.`CompPart`($idreq$,$p_n$);
        **deliver** (RESPONSE,$r$,$res$,$p_n$) to application;
        **return**;
      **else** (* timeout *)
        *aggregation*$_{r.type}$.`GetAcc`($idreq$,$myId$,*prob*);
        *tracking*$_{r.type}$.`IndStatus`($idreq$,$p_n$,FALSE,*prob*);
        **deliver** (RESPONSE,$r$,$\perp$,$p_n$) to application;
        **return**;

**Upon** receiving (`Request`,$idreq$,$r$,$s$) msg from $p_j$ **do**
  *frwd* $= nil$;
  *aggregate*$_{r.type}$.`GetPart`($idreq$,*fwrd*,$myId$,$\perp$,$\perp$);
  **if** (*frwd* $\neq nil$) **then**
    send (`Fwd`,$idreq$,$r$,*fwrd*) msg to $p_j$;
    send (`Request`,$idreq$,$r$,$s$) msg to *fwrd*;
  **else**
    **if** (Matching($r$)) **then**
      **deliver** (REQUEST,$idreq$,$r$,$p_j$) to application;
    **else**
      (*find some random peer $p$ in neighborhood*)
      $fwrd$ :=*group*$_{r.type}$.GetPeer();
      send (`Fwd`,$idreq$,$r$,$fwrd$) msg to $p_j$;
      send (`Request`,$idreq$,$r$,$s$) msg to *fwrd*;
  **return**;

Fig. 3: Interaction Between the Application and the Incentive Middleware

by the supervisor to send a request to the tested site, are not burden with additional testing requests). Once the request has been sent, the requester site $p$ waits until the request is satisfied or a timer times out. In the latter case, $p$ provides a non participation indication to the tracking service. Note that this indication is accepted by the tracking service with a probability equal to $p$'s access level. This decreases the weight of non-cooperative sites. In the former case, if $p$ receives the requested resource then both the participation and the suspicion levels of

the recipient $q$ are updated accordingly. The resource is forwarded to the application. Finally, if $p$ receives a forwarding indication, $p$ checks whether this forwarding is well-founded or not by calling the aggregation service. In the event of an affirmative, the forwarding is rewarded as such by increasing $q$'s participation level and decreasing $q$'s suspicion level. In the other case, $q$'s suspicion level is increased because $q$ was not allow to forward the request (i.e., the $\nu$-forwarding rule did not apply). As previously, the scope of this indication depends on the access level of $p$. The application layer of the requester is informed that the recipient did not act correctly. This ends the request transaction.

### *B. The services in Detail*

In the following we provide for each service the interface it exports, and their implementation. Note that for space reasons, trivial implementations are omitted.

*1) Registration Service:* By invoking this service with the list of his resources, a peer obtains the set of supervisors in charge of them. The interface exported by this module consists in the following method :

`Join`(**in** RESOURCE *res_list*, **out** (PeerId,RESOURCE) *sup_res*)
Indicate that the invoking peer wishes to get his supervisors.
The *sup_res* output parameter contains a list of tuples (*sup*,*res*) for each *res* in *res_list* list. An unique supervisor is associated to a given resource.

*2) Semantic Group Service:* The system self-organizes into semantic groups. This service guarantees fairness participation for any peer by *i)* allowing peers sharing the same interests to group together in order to augment the requests successful rate, *ii)* homogenizing the requests flow (in P2P, some peers are "attractive" than others which contradicts the symmetry principles of these systems), and *iii)* enabling new peers to shortly participate to augment their level of participation (initially, peers have no reputation which does not motivate other peers to deal with them).

The interface exported by this module consists of the following methods :

`Group`(**in** RESOURCE *res*,**in** PeerId *sup*)
Constructor that instantiates the module
`Join`(**out** NEIGHBORHOOD *mb*)
Indicate that the invoking peer wishes to join the semantic group.
The *mb* output parameter contains the membership of the group.
UPCALL `NewView`(**in** RESOURCE *res*,**in** PeerId *p*)
A new view has to be disseminated.

The Semantic Group Module Implementation is as follows :

Local variables :
*members* - the membership list of the group
*supervisor* - the supervisor of the group
*resourceType* - the type of the group

SemanticGroup : :Group(*r*,*sup*){
Initializes variables *members* and *supervisor* ;
Allocates memory for the *members* data structure ;}

**Upon** `Join`(*membership*) from application **do**
*members* :={myId} ;
send (MB,*resourceType*,*members*) message to *supervisor* ;
**wait until** (VIEW,*res*,*memb*) msg is received from *supervisor* ;
*members*:=*memb* ;
**return** ;

**Upon** receipt of (MB,*res*,*newmember*) msg from the network **do**
*members* :=*members* ∪ *newmember* ;
send (VIEW,*res*,*members*) to *members* ;
**return** ;

**Upon** receipt of (VIEW,*res*,*memb*) msg from *supervisor* **do**
*members*:=*memb* ;
invoke `NewView`(*res*,*memb*) ;
**return** ;

*3) Aggregation Service:* This module computes the access level and the participation level of a peer as defined Section III-B.

`Aggregation`(**in** RESOURCE $r$,**in** PeerId $s$,**in** NEIGHBORHOOD *mb*)
Constructor that instantiates the module
`NewView`(**in** NEIGHBORHOOD *mb*)
Update of the membership list is needed
`CompPart`(**in** INT #,**in** PeerId $peer$)
The invoking peer indicates that *peer* has satisfi ed a request
`GetPart`(**in** INT #,PeerId $p$,PeerId $q$,INT part1,INT part2)
The invoking peer wants to know the participation level of peers $p$ and $q$
`CompAcc`(**in** INT #,**in** PeerId $peer$,**out** LONG prob)
`GetAcc`(**in** INT #,**in** PeerId $p$,**out** LONG prob)
The invoking peer wants to know the access level of $p$

The Aggregation Module Implementation is as follows :

Local variables :
*supervisor* - the supervisor in charge of the aggregation module update
*resourceType* - the type of the resource associated to *supervisor*
*members* - the list of members involved in this module
*access* - the access level list containing the access level value of each peer involved in this module
*part* - the participation level list containing the participation level value of each peer involved in this module

Aggregation : :Aggregation($r$,$s$,*mb*){
Initializes all the *supervisor*, *resourceType*, and *members* variables ;
Allocates memory for the *access* and *participation* data structures }

**Upon** `GetPart`($idreq$,*fd*,*fr*,$s_1, s_2$) from application **do**
send a (GetPART,$idreq$,$myId$,*fd*,*fr*,$s_1, s_2$) message to *supervisor* ;
**wait until** receipt (PART,$idreq$,$p_1, p_2, val_1, val_2$) from *supervisor*
*fd* :=$p_1$ ;*fr* :=$p_2$ ;$s_1 := val_1$ ;$s_2 := val_2$ ;
**return** ;

**Upon** receiving a (GetPART,$idreq$,$myId$,*fd*,*fr*,$s_{fd}$,$s_{fr}$) message from $p_i$ **do**
**if** (*fd*=nil) **then**
search in the *part* list the peer $p$ with the minimum participation level $part_p$
**if** ($part_p < part_{fr}$) **then**

```
        send (PART,idreq,p,fr,part_p,part_fr) message to p_i ;
      else
        send (PART,idreq,nil,fr,⊥,⊥) message to p_i ;
    else
      send (PART,idreq,fd,fr,part_fd,part_fr) message to p_i ;
    return ;

Upon CompPart(idreq,p) from application do
    send a (IncPART,idreq,p) to supervisor ;
    return ;

Upon receiving a (IncPART,idreq,p) message from p_i do
    part_p++ ;
    return ;

Upon GetAcc(idreq, p,acc) from application do
    send (GetACC,idreq,p) message to supervisor ;
    wait until receipt (ACC,idreq,p,value) from supervisor
    access_p :=value ;acc :=value ;
    return ;

Upon receiving a (GetACC,idreq,p) message from p_i do
    send (ACC,idreq,p,acces_p) message to p_i ;
    return ;

Upon CompAcc(p,req,resp) from application do
    if (resp ≠ 0) then
      access_p := al_0 + (1 − al_0) resp/req ;
    else access_p := al_0 + ε ;
    return ;
```

*4) Cooperation Tracking Service:* This service tracks peer suspicion level. Peers feed this service by providing it the outcome of their interactions with other peers of the group.

This service exports the following methods :

```
tracking(in RESOURCE r,in PeerId s)
  A constructor that instantiates the module.
IndStatus(in INT #,in PeerId p, in Boolean status,in LONG prob)
  The invoking peer indicates in status whether p has responded to
  its request or not. The report is taken into account with probability
  prob, which is the access level of the requesting peer.
UPCALL CompAl(in INT #,in PeerId p,in INT req,in INT resp)
  Computation of the access level is possible
UPCALL Test(in PeerId p);
  Peer p has to be tested
UPCALL EndTest(in PeerId p);
  Test of peer p is over

Local variables :
  supervisor - supervisor in charge of the tracking module update
  resourceType - type of the resource associated to supervisor
  members - list of the members sharing the same resourceType
  suspicion - list containing the suspicion value of each p ∈ members
  tested_p - p's structure with 3 fi elds : status, req and rsp
  tested - list of tested_p structure

Tracking : :Tracking(r,s,mb){
  allocate memory for all the data structures; Note that the status
  of all the peers is initialized to TRUE. This implements
  the initial test of newcomers
  initialize supervisor, resourceType and members}

Upon IndStatus(idreq,p,status,prob) from application do
  send a (TRACK,idreq,p,status,prob,tested) msg to supervisor ;
  return ;

Upon receipt of (TRACK,idreq,p,status,prob) msg from the network
  if (tested_p.status = TRUE) then
    tested_p.req++ ;
    if (status) then tested_p.resp++ ;
    if (tested_p.req ≥ D) then
      send a (ENDTEST,p) msg to all peers in 𝒟 ;
      invoke CompAl(p,tested_p.req,tested_p.resp) ;
      reset tested_p ;
  if (status=TRUE) then
    if (suspicion_p > 0) then
      suspicion_p- - ;
  else
    if (suspicion_p < susp_max) then
      do (suspicion_p++) with probability prob ;
      if (suspicion_p = susp_max ∧ ¬tested_p.status) then
        Select a set 𝒟 of D peers in members such that these
        peers have the lowest suspicion level ;
        tested_p.status:= TRUE ;
        send a (REQTEST,p) message to each peer ∈ 𝒟 ;
  return ;

Upon receipt of a (REQTEST,p) from the supervisor ;
  invoke Test(p) ;
  return ;

Upon receipt of a (ENDTEST,p) from the supervisor ;
  invoke EndTest(p) ;
  return ;
```

## VI. Simulations

We have run simulations over 1000 virtual peers. The number of resources types, and thus, the number of groups and supervisors are chosen randomly from a Gaussian distribution with mean 100. Simulations have been ran during 10000 units of time. Requests arrive randomly following a Poisson distribution such that every peer makes approximately 1 request per 10 units of time.

Peers are classified in three categories according to their type :

- **obedient peers** follow the prescribed protocol,
- **free riders** reject all incoming requests,
- **adaptive peers** adapt to the evolution of their welfare.

To simulate the adaptive changes, we used the following learning algorithm : an adaptive peer has a set of *strategies* $\{accept, reject\}$ from which he chooses randomly one, at each interaction. Then, after a fixed number of interactions called *stage*, each strategy $s$, generates a utility $u(s)$ (see Equation 1). Then, at the next stage, the probability of choosing a strategy $s$ is $\frac{u(s)}{\sum_s u(s)}$.

The ratio of obedient peers, free riders and adaptive peers are, respectively, 0.5, 0.1 and 0.4.

Figure 4 studies the behavior of different peers face to the fair cooperation incentive mechanism. The ac-

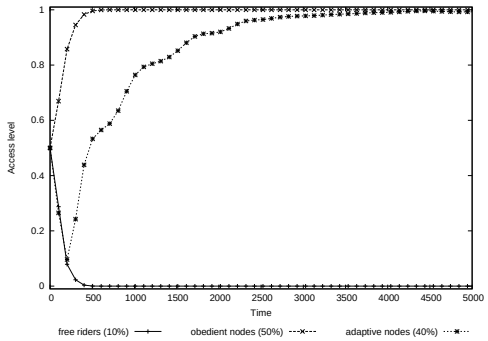


Fig. 4: Access Level Differentiation Mechanism.

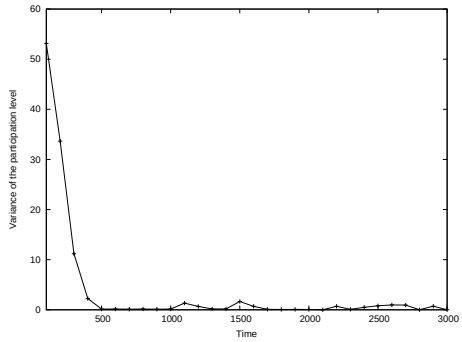


Fig. 5: Fairness Mechanism.

cess level of the obedient peers converge quickly to 1, while the access level of free riders drops to 0. This highlights the two first properties of the mechanism (see Section IV). Furthermore, the access level of the adaptive peers significantly change and converge to the obedient ones after having decreased during a short period. In fact, the decreasing phase corresponds to the detection of the bad behaviours and the increasing phase shows that adaptive peers has changed their strategy to cooperation.

Figure 5 shows the variance of the participation level of peers with the fair cooperation incentive mechanism. This shows that the difference between the participation level of peers and the mean overall participation very quickly converge to nearly 0. That is, peers are almost equally solicited. This corroborates the fairness property of the mechanism, described in Section IV.

## VII. Conclusions and Further Work

In this work we have presented an incentive mechanism for Fair Resource Sharing. Our mechanism is composed of a novel incentive, fair cooperation, and an algorithmic part encapsulated in a middleware layer. Moreover, we have proposed an architecture for the middleware layer of our mechanism that brings together several research areas (aggregation, semantic group membership, and tracking) and have implemented it using peer-to-peer techniques. We have proven the efficiency of our mechanism through simulations.

In our near future work we are interested in formally analyzing our fair cooperation mechanism as well as extending the simulation work to measuring the trade-off between fairness in the load balancing and efficient request serving.

## VIII. Acknowledgments

We wish to convey our thanks to the "Region de Bretagne" for supporting this work.

## References

[1] E. Adar and A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Discouraging free riding in a peer-to-peer cpu sharing grid. *13th IEEE Int. Symposium on High-Performance Distributed Computing*, pages 129–137.

[3] P. Antoniadis, C. Courcoubetis, and R. Weber. An asymptotically optimal scheme for p2p file sharing. In *2nd Workshop on the Economics of Peer-to-Peer Systems*, Harvard University, 2004.

[4] D. Banerjee, S. Saha, S. Sen, and P. Dasgupta. Reciprocal resource sharing in p2p environments. In *4th Int. Joint Conference on Autonomous Agents and Multi Agent Systems (accepted)*, 2005.

[5] E. Buchmann and K. Böhm. Fairnet - how to counter free riding in peer-to-peer data structures. In *Prof of the Int. Conference on Cooperative Information Systems*, pages 337–354, 2004.

[6] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *Proc. of the 3rd Int. Conference on P2P Computing*, page 48, 2003.

[7] M. Feldman, K. Laï, I. Stoica, and J. Chuang. Scalable and robust incentives techniques for peer-to-peer networks. In *Proc. of the ACM Conference on Electronique Commerce*, 2004.

[8] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. In *3rd Annual Workshop on Economics and Information Security*, pages 228–236, 2004.

[9] E. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2) :173–199, 2001.

[10] T-W. Ngan, D. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *2nd Int. Workshop on Peer-to-Peer Systems*, pages 149–159, 2003.

[11] N. Ntarmos and P. Triantafillou. Aesop : Altruism-endowed self organizing peers. In *Proc. of the 2nd Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing*, pages 151–165, 2004.

[12] N. Ntarmos and P. Triantafillou. Seal : Managing accesses and data in peer-to-peer data sharing. In *4th IEEE Int. Conference in Peer-to-Peer Computing*, pages 116–123, 2004.

[13] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking*, pages 156–170, 2002.

[14] J. Shneidman and D. Parkes. Rationality and self-interest in peer to peer networks. In *Proc. 2nd Int. Workshop on Peer-to-Peer Systems*, pages 139–148, 2003.