Coalitions and Incentives for Content Distribution over a Secure Peer-to-Peer Middleware

Maria-Victoria Belmonte, Manuel Díaz and Ana Reyna
Department of Languages and Computer Science
E.T.S.I. Informática. Bulevar Louis Pasteur, N.35
University of Málaga (UMA), 29071, Málaga, Spain
e-mail: {mavi, mdr, reyna}@lcc.uma.es

***Abstract*— Nowadays, Peer-to-Peer is responsible for more than 60% of Internet traffic. These protocols have proved to save bandwidth and computing resources in content distribution system. But, problems related to user behaviour, such as free riding, still persist, and users must be motivated to share content. In previous work, we have designed and simulated a coalition and incentive theoretical mechanism for content distribution that aims to fight against problems in user behaviour. In this paper, we present a real implementation of it. Since developing a peer-to-peer application from scratch is a laborious and error prone task, we use SMEPP, a middleware that aims to ease the development of secure distributed application, to implement it.**

Keywords-coalitions; incentives; peer-to-peer; middleware; overlay.

I. INTRODUCTION

Nowadays, Peer-to-Peer (P2P) protocols are responsible for more than 60% of Internet traffic, in spite of anti-piracy laws [1]. Many Internet applications are taking advantage of P2P architecture, since P2P paradigm abandons central servers to give way to a network where all nodes play the role of server and client simultaneously. This brings new perspectives to application scalability; where an excess of nodes in the client-server paradigm could lead to saturation or even a system crash, in the P2P paradigm it means greater capacity.

P2P protocols enable content distribution in a cost-effective way, as they do not require a centralised provider to handle all the demands. Instead, a P2P protocol can use its clients' bandwidth for content distribution, saving the bandwidth and computing resources of the system. However, the performance and availability of these systems relies on the voluntary participation of their users, which is highly variable and unpredictable. Empirical studies have shown that a large fraction of the participants share little or no files. For instance, in [1], the authors affirm "*in Gnutella 25% of the users do not share any files, Furthermore, about 75% of the clients share 100 files or less*" (including the 25% that do not share) "*and only 7% of the clients share more than 1000 files. This 7% of users together offer more files than all of the other users combined*". More recently, Handurukande et al. [3] also observed the same behaviour in the eDonkey P2P network and concluded that this is common to most P2P file sharing systems. This phenomenon is known as "free-riding", and is still an open issue on content distribution systems [4]. P2P content distribution systems need mechanisms that motivate peers to share their content.

In [5], we presented a new coalition formation scheme based on game theory concepts which formally prove how coalitions improve P2P systems performance, encouraging participants to contribute resources, receiving in return a better quality of service. Empirical results obtained through simulations illustrated how our approach encourages collaborative behaviour, preventing the free-riding problem and improves the overall performance of the system. Until now, this mechanism has been a theoretical proposal, whose features have been demonstrated only through simulations. In this paper, we present a real distributed implementation of the mechanism.

The development of distributed applications in general, and concretely P2P, is a laborious and error prone task, since many issues must be considered, from network protocols, to security. In order to facilitate the software development of this kind of system, new tools and methodologies capable of abstracting all the underlying complexity should be used. A middleware can simplify and reduce the development time of the design, implementation and configuration of applications, thus allowing developers to focus on the requirements of their applications. In [5], we presented SMEPP (Secure Middleware for Embedded Peer to Peer systems), a new middleware especially, but not exclusively, designed for Embedded Peer to Peer (EP2P) systems. This middleware was designed to overcome the main problems of existing domain specific middleware proposals [6]. The middleware is secure, generic and highly customisable, allowing it to be adapted to different devices, from PDAs and new generation mobile phones to embedded sensor actuator systems, and domains, from critical systems to consumer entertainment or communication [8].

In this paper, we take advantage from SMEPP middleware to implement our coalitions and incentives mechanism for distributed content distribution. On the one hand we prove the suitability of using a P2P middleware, and on the other, we demonstrate that our mechanism can be developed as a real distributed application.

 ISBN: 978-1-61208-173-1

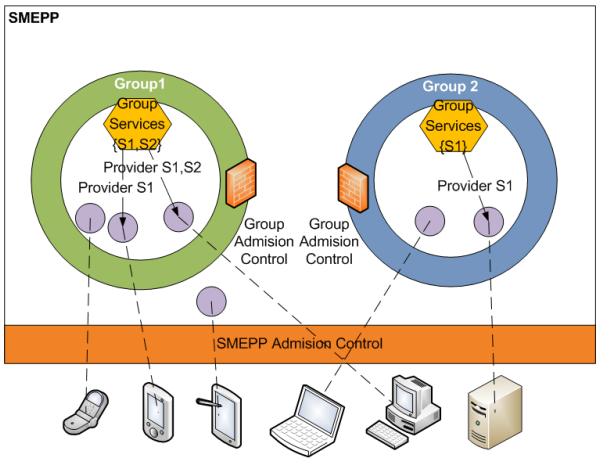

Figure 1. SMEPP abstract model

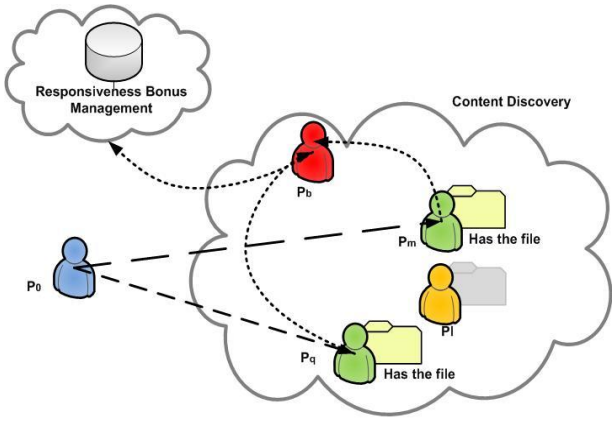

Figure 2. Coalitions and incentives mechanism

The structure of this paper is as follows. The following section presents an overview of SMEPP middleware. In Section III, the main features of our content distribution mechanism are introduced. Section IV focuses on the implementation issues. Finally, conclusion is presented in Section V.

II. SMEPP MIDDLEWARE OVERVIEW

SMEPP middleware is based on three main pillars, its abstract model, its reusable and flexible architecture and its built in security. A detailed description of SMEPP middleware is beyond the scope of this paper, nevertheless we believe it is essential to introduce its main features. In addition, in Section IV, some details, required for the implementation, will be given. More details of SMEPP can be found in [5] .

The abstract model defines the entities involved and how they relate in P2P environments (illustrated in Figure 1). It defines the concepts of peer, group and service. The functionality of the application is offered in the form of services, which can be published or consumed only inside groups of peers. The group definition determines the level of security inside a group. To access a group the peer has to provide the suitable credentials, this is internally managed by the middleware. The service discovery is effectively performed thanks to the underlying structured overlay network that implements CHORD protocol [9]. In addition, the abstract model also defines the API, which is generic and language independent, and defines the functionality exposed by the middleware with a high level of abstraction, this is the way the programmer can interact with the middleware.

The architecture of SMEPP is based on software components. Component-oriented paradigms have proved to be a good approach to designing a middleware. Software components offer several features (reusability, adaptability, etc.) which are particularly suitable for dynamic environments and rapidly changing situations that a middleware has to face. This is especially interesting for our application. A specific component framework has been designed for the implementation of the middleware. The developer has several tools which allow the tuning of the middleware for a specific platform, device or communication protocol.

Security is the most distinctive feature of SMEPP. Since its conception the security aspect was considered, and tackled transversally on the architecture and on the service model definition, this ensures that the middleware is capable of providing a high level of customisable security.

The SMEPP performance results showed that overall resource consumption of the middleware was relatively small, the overall memory consumption peak being 1,83MB and the highest average memory consumption being 670kB. Moreover regarding the usage of CPU, the middleware uses relatively little CPU time (max being 2% of CPU capacity on a 2GHz Intel Core2 Duo). Taking into consideration that the middleware is designed to work on small capability devices, this is a good result. Furthermore, the suitability of SMEPP was demonstrated by the development of two different innovative real-life applications in the domains of Context Aware Mobile Telephony and Environmental Monitoring in Industrial Plants [8]. This was formerly implemented in JXTA [10], where a new version of the application over SMEPP proving the benefits of using middleware was developed. A comparison between JXTA and SMEPP can also be found in [11].

In this paper, we demonstrate how our coalitions and incentives mechanism for content distribution can be easily implemented over SMEPP, taking advantages of its features, such as the built in security and the structured overlay look up mechanism for content discovery.

III. COALITION AND INCENTIVES MECHANISM OVERVIEW

The central idea of our mechanism is sharing the task of downloading a file between a set of peers making up a coalition. On the one hand, the downloader benefits as the total download time is reduced. On the other hand, the burden on the uploader (or provider) peer is also alleviated, since the total task is divided between the members of the coalition. And in addition, providers are rewarded for their participation in the coalition.

More concretely, these rewards aim to encourage participants to contribute resources, receiving in return a better quality of service. In this way, each peer that participates in a coalition is lending "bandwidth" to other coalition peers, in

 ISBN: 978-1-61208-173-1

exchange for profit or *utility*. Each participant or provider receives a reward each time it participates in a coalition, and is penalised each time it downloads. The reward that a provider obtains by performing a task inside a coalition is calculated using the game-theory concept of *core* [12]. The *core* ensures that each coalition participant receives a fair utility in return for the bandwidth that it supplies. In our model, these utilities are used to compute the *Responsiveness bonus* (*Rb*), which represents the overall contribution of the peer to the system. Therefore, this value will determine the quality of service of each peer. The higher *Rb* the better quality of service, this is the key to encouragement.

Each peer can play three different roles: downloader, participant or manager (In Figure 2, P_b is the downloader, P_0 is the manager, and P_q and P_m are the participants). To sum up: the download process starts when a peer decides to download a file. In order to download this file, the downloader has to find file providers in the network (discovery). Once the providers are found, a coalition manager is elected. The manager selection does not imply centralization, because any potential participant can become the manager with equal probability. Next, the manager sends offers to the potential candidates (the rest of the providers). Each provider answers the offer, and once the manager has received all of them (or a timeout is reached), it has to divide up the task between the potential coalition members (those participants who answered). This process, called *Task Assignment*, establishes the coalition itself, and after this, the download itself starts. During the download, the downloader periodically sends acknowledgement information to the manager, who runs a *checking mechanism* to guarantee the quality of service in the coalition, adapting to network traffic and helping to avoid some attacks of malicious peers (such as free-riding). After these checks, the *Rb* of all the members of the coalition is updated using the utilities obtained after the *Coalition Payment Division*.

A. Task Assignment

Given a collection of providers, the task assignment has to determine the task that each provider will be responsible for, this is the input bandwidth that each participant will provide to the coalition. If there are few participants (under a threshold) no selection has to be done, otherwise only some providers will be chosen for the coalition.

To do so, and to determine the input bandwidth of a participant, *the progressive filling algorithm* is used. This algorithm provides the *max-min fairness* [13]. A bandwidth allocation is max-min fair if and only if an increase of the input bandwidth of a peer *x* within its domain of feasible allocation is at the cost of decreasing some other input bandwidth of a peer *y*. So, it gives the peer with the smallest bidding value the largest feasible bandwidth.

B. Checking Mechanism

The checking mechanism makes the system less vulnerable to peer failures, churns and network congestion problems, while it ensures the quality of service of the coalition. The mechanism works as follows, during the download of a file; the downloader sends acknowledgement information to the manager with a predefined frequency. The manager calculates the difference between the bytes sent and the ones which should have been sent (according to the task assigned to each participant). If this difference exceeds a predefined threshold, the coalition is reconfigured in order to provide better quality of service. Moreover, the manager also checks that the downloader *Rb* is high enough to keep downloading. The central idea is that if the coalition is not working as it was expected or the downloader is abusing the system, the coalition is cancelled.

Since the update of *Rb* values are calculated by the manager and are based on the acknowledgement sent by the downloader, the downloader could avoid the penalty if it sends faked acknowledgement. But the checking mechanism performed by the manager will stop the coalition if the acknowledgement is too small, so the downloader will not be penalised, but neither will they receive the file.

C. Coalition Payment Division

The hallmark of our mechanism is that the coalition payment division ensures fairness, thanks to the game theory concept of *core*. This means that peers won't be negatively affected if they have lower capacity. The details of this are explained in the following paragraph.

Let's call coalitional value *V(S)*, to the total utility or profit of a coalition *S*. For every peer in the coalition, $Pi \in S$, we must distribute *V(S)* between the peers, and assign an amount or utility (x_i) to every peer $Pi \in S$. The problem is to distribute *V(S)* in a stable and fair way so the coalition peers have no reason to abandon it.

Firstly, we must calculate *V(S)*. The profit obtained by *S* is calculated as the difference between the time required for the download with just one uploading participant (only P_0, the manager) minus the time it takes with the coalition *S* (all the participants, including the manager). Then the coalitional value is given by the following equation:

$$V(S) = t_0 \frac{\sum_1^n b_i^{in}}{\sum_0^n b_i^{in}} \quad where\ t_0 = \frac{File\ size}{b_0^{in}} \qquad (1)$$

where t_0 is the time that it would take the P_0 to upload the whole file (being P_0 the only uploader or provider), b_0^{in} the upload bandwidth of P_0 and b_i^{in} the upload bandwidth of the remaining participants of *S*.

Secondly, we use the *core* to distribute *V(S)* between the coalition members. A utility distribution belongs to the *core* if there is no other coalition that can improve on utilities of all of its members. The stable utility division (x_i) to every peer $Pi \in S$ is given, then, by the following equation (in detail in [5]), where b_0^{out} is the download bandwidth of P_0 .

 ISBN: 978-1-61208-173-1

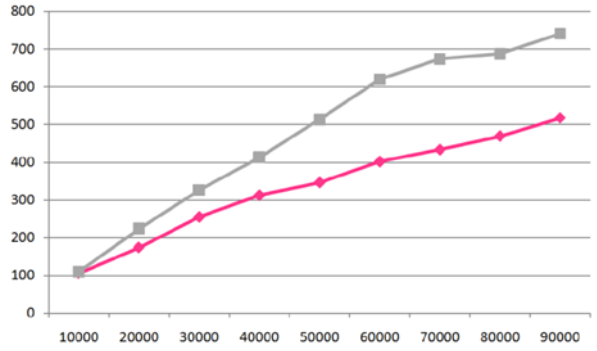

Figure 3. Average download time using coalitions (diamonds) and not using them (squares) (simulation time vs bytes)

$$x_i = \begin{cases} t_0 \dfrac{\left(\sum_1^n b_i^{in}\right)^2}{\left(\sum_0^n b_i^{in}\right)^2} & if\ i = 0 \\ t_0 \dfrac{b_0^{out}\ b_i^{in}}{\left(\sum_0^n b_i^{in}\right)^2} & if\ i \neq 0 \end{cases} \quad (2)$$

D. Responsiveness Bonus Computation

As it has been said, peers with higher utility will get a better quality of service. In our approach the utility accumulated by each peer (Rbi) is proportional to the resources that it supplies, and it is calculated as a heuristic function of x_i. The value of Rb_iwill be reduced when P_i acts as a downloading peer, and incremented when it is a provider or uploading peer. The heuristic uses the x_i values obtained by Pi by means of U_{pi} (Upload points) and D_{pi} (Download points). U_{pi} and D_{pi} accumulate the utility obtained by each coalition formation process in which P_i participates.

Let us call Fsi to the number of files shared (the total size in bytes) by a peer Pi. The Rb_i value of the peer is calculated using the following equation:

$$Rb_i = \begin{cases} 1 & if\ (U_{pi} - D_{pi}) \geq 0 \\ 0 & if\ (U_{pi} - D_{pi}) < 0 \wedge U_{pi} = 0 \wedge F_{si} = 0 \\ 1 & if\ (U_{pi} - D_{pi}) < 0 \wedge U_{pi} = 0 \wedge F_{si} > 0 \\ \dfrac{U_{pi} \cdot \gamma}{D_{pi}} & if\ (U_{pi} - D_{pi}) < 0 \wedge U_{pi} > 0 \end{cases} \quad (3)$$

The Rb_i values are between zero and one. The interpretation of this formula is that if the peer uploads more than downloads, it gets the maximum value, also true when, it is not uploading but sharing. If neither uploading nor sharing; its Rb_i is set to zero. In any other case, it is calculated as the ratio between the upload and the download points (the γ parameter allows us to regulate the relation to increase/decrease the penalty/reward).

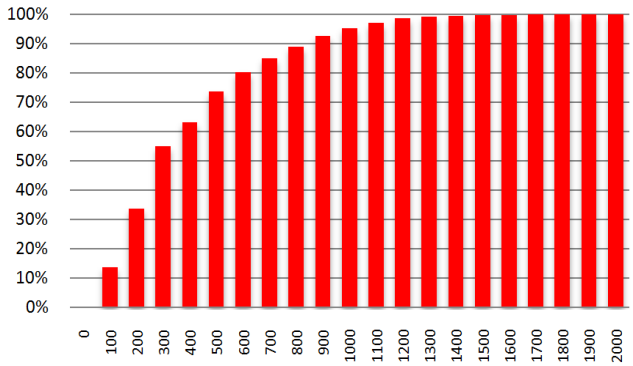

Figure 4. Free tider detection (% detection vs simulation time)

Therefore, we use this value to decrease the download bandwidth $Rb_i \cdot b_b^{out}$ (using it as a multiplier of the download bandwidth of the peer Pb when it wants to download a file). Initially, the Rb_i of the peers is 1, a higher responsiveness bonus (Rb_i closer to 1) will mean that P_i will be able to use most of its bandwidth capacity. Otherwise, a Rb_i closer to 0 will reduce its bandwidth capacity, (in fact, it could even avoid creating the coalition for the download when it is 0). Thus, our incentive mechanism penalises the selfish behaviour of the peers, and provides incentives for collaborative behaviour.

E. Experimental Results

In [5], we presented some simulation results. These experiments confirmed the benefits of using our mechanism. On the one hand download times are improved, and on the other hand, free riders are stopped, this lead to an improvement of the system's effectiveness.

Our own simulator was used to run the experiments. It was configured to simulate a P2P network of 1000 peers during 2000 units of simulated time (steps). All peers had the same bandwidth capabilities. The collection of files shared in the network was defined with different sizes (from 10000KB to 90000 KB), and a random number of copies (between 5 and 500) of these were delivered through the network at the start of the simulation. Each peer had a random number of initially stored files, and the objective of the simulation was that every peer download the files that were not initially stored. Our simulations, considered three types of users (or behaviours): free riders (FR), collaborative (C) and adaptive (A). The first, do not share at all, the second share as much as possible, and the last, only share if they want to download. Depending on the behaviour of each peer (that is randomly assigned in each simulation) it will face its downloads in different ways.

 ISBN: 978-1-61208-173-1

TABLE I. BYTES DOWNLOADED IN SIMULATIONS

	No coalitions		*Coalitions*	
	Pop. 1	***Pop. 2***	***Pop. 1***	***Pop. 2***
FR	157 Gb	123 Gb	24 Gb	20 Gb
C	156 Gb	92 Gb	130 Gb	82 Gb
A		91 Gb		84 Gb
Total	313 Gb	306 Gb	154 Gb	186 Gb

To analyse the impact of the different behaviours on the system the experiments were run with two different populations. The first one without adaptive users: 50% FR, 50% C and 0% A, called Population 1. And the second with adaptive users: 40% FR, 30% C and 30% A, called Population 2. In addition, to analyse the impact of the use of coalitions, simulations were run with and without incentive policies: No Coalitions (NC), where no incentive mechanism was considered and Coalitions (C), which implemented our proposal. After repeating the simulation experiments 100 times we took the average to give the results. Two main metrics were considered: downloaded bytes and average download time.

In Table I the bytes downloaded per populations and per behaviour for both scenarios, with and without coalitions, is shown. In Population 1, when coalitions were used the total amount of bytes downloaded was reduced to 50% with respect to NC, but 84% of this reduction was due to the free riders detection. This showed how the algorithm prevents free riders from abusing, avoiding the overhead of the system resources. In Figure 4, the free rider detection effectiveness of our approach is shown (Population 2). More than 50% were detected at step 300 and the 100% were stopped at step 1500.

In Population 2, when adaptive users were introduced, the benefit of using of coalitions was higher (than in Population 1). The total amount of bytes was reduced by 39% with respect to NC, where 83% was due to the free rider's detection. In addition, comparing coalitions in both populations, the total amount of downloaded bytes were increased by 20% using Population 2, proving that adaptive users benefit the system. Note that in Population 2 there were fewer free riders and collaborative users, therefore, less shared files in the network, this justifies the smaller amount of total bytes downloaded with respect to Population 1.

In addition to the analysis of the downloaded bytes, the average download time offered even better results. In Figure 3, the average download time using and not using coalitions is shown for Population 2. Experiments showed that using coalitions the average download time was smaller. As expected, the benefit of using coalitions is increased as the file size grows. When adaptive users were introduced the download times were improved compared with NC, what demonstrated the effectiveness of our incentive mechanism. More details about the configuration and results of the experiments can be found in [5].

F. Related Works

The incentive mechanisms in P2P networks for content distribution [4] have been classified in different categories. Our approach belongs to reciprocity based mechanisms: peers that contribute more get a better quality of service. Other publications also included in this category are [14][15][16][17][18][19].

From the approaches above, those based on mutual reciprocity, like Bit Torrent [16], Emule [15] or [14], do not fit the asymmetric nature of a collaborative relationship, since the peer's decision to upload to another peer is based on the direct exchange of data/services/credits between two peers that have mutual interests (same content). However, our approach, unlike the ones above, encourages cooperative behaviour by forming coalitions of peers that help each other to download files. So any peer can participate in a coalition increasing its *Rb*, and this will lead to a higher download bandwidth for further downloads from any other peer in the system.

The indirect reciprocity-based approaches, like [17][18][19] or our approach, consider peers' overall contribution to the network, and so they encourage cooperation.

2Fast [17] is also based on creating groups of peers that collaborate in order to download files. However, the system does not enforce fairness and does not specify how the helper may reclaim its contributed bandwidth in the future. Again in [19], where the peer contribution is based on the number of peer uploads and downloads, the computed peer contribution does not guaranty that the peers receive fair utility in return for the bandwidth that they supply. Finally, Karakaya et al. [14] propose a distributed framework in which each peer monitors its neighbours (recording the number of messages coming or going), and the free-riders are located and isolated. However, and unlike this approach, stopping free riders is not the only goal of our approach, indeed we also wish to increase the effectiveness of the download mechanism.

IV. IMPLEMENTATION

When faced with the implementation of our content distri-

TABLE II. SMEPP API

Group management	***Service management***
createGroup	*publish*
joinGroup	*unpublish*
leaveGroup	*getServices*
getGroups	*getServiceContract*
getGroupDescription	*startSession*
getPeers	***Peer management***
getIncludingGroups	*newPeer*
getPublishingGroup	*getPeerId*
Message handling	***Event handling***
invoke	*event (raise)*
receiveMessage	*receiveEvent*
reply	*subscribe*
receiveResponse	*unsubscribe*

 ISBN: 978-1-61208-173-1

bution mechanism, first, the functionality has to be modelled as a service, since SMEPP is service oriented. Then, two main implementation issues must be tackled: first, the content discovery, and second the query and update of the Responsiveness bonus. Both issues will take advantage of the structured overlay network offered by SMEPP.

As we have already stated, SMEPP is based on the concept of peer, group and service. Its API offers primitives to abstract the peer management, group management, and the service management, but also for events and message handling. The SMEPP API is summarised in Table II.

A. Content Discovery

The content discovery process is responsible for finding the peers that provide a specific file in the network. This task is tackled differently in popular P2P systems. In Napster, this responsibility was delegated to central servers which were also responsible of storing the files; In Emule, the task is delegated to many different servers that only store provider's references, not the files. Gnutella implements a pure distributed algorithm to find providers, forwarding query messages through the neighbours. BitTorrent or One-Click Hosting (like megaupload, rapidshare, etc.), do not provide any mechanism for content discovery. In BitTorrent, torrent files containing the description of the shared file are published through webs, mail, forums, etc so, the user must find this torrent file in order to join or to start a download. The same goes for one-click hosting, where search engines or webs specialised in download links are used to find the sources of the content.

As SMEPP integrates Service Discovery functionality, we take advantage of this in order to be able to efficiently discover the file a peer wants to download. We could also opt for an implementation based on events, but the file search would be less efficient and the implementation effort would be bigger. SMEPP defines the services through *contracts* (A XML File). The contract provides descriptive information on the service, while the implementation is the executable service (e.g., a Java service) exposed to the middleware through grounding. A service contract describes "what the service does" (viz., the service *signature*), "how it does it" (viz., the service *behaviour*), and it may include other extra-functional service properties (e.g., QoS). When a service is published, SMEPP generates a *key* from the contract; this *key* determines which peer in the group is responsible for this service. The structured SMEPP overlay network (which uses CHORD protocol [9]) determines the range of *keys* a peer is responsible for, and enables a fast search mechanism thanks to the *key space* defined within the group.

To take advantage of this effective search mechanism, we define a contract for each shared file. This way if two peers share the same file, they will publish the same contract, which will result in the same key, and therefore, the same responsible peer. Thus, peers which share the same file, will publish the same service on the same peer.

B. Responsiveness Bonus Management

In addition to the content discovery, the storage (query and update) of the *Rb* is another important implementation issue. The *Rb* represents the overall contribution of a peer, so it has to be *updated* every time a peer participates in a coalition (as downloader or as participant or manager, typically the former will decrease the *Rb* and the latter will increase it.). Every time a manager set up a coalition for a downloader, it has to *query* the downloader's *Rb*, in order to determine the bandwidth that the coalition will provide.

There are several choices to implement this. On the one hand we can opt for a centralised storage server. This would simplify the update and query processes, and would require less communication effort than in a pure distributed scenario. But this came at a price: that of a single point of failure. On the other hand, the pure distributed scenario, requires a complex algorithm for the calculation of the *Rb* value, such as the ones used in distributed consensus systems [20] which requires a lot of effort to maintain consistency. Finally, we can opt to take advantage of the structured overlay network in a similar way as for the approach of the implementation of content discovery. As foreseen, in this paper we choose this last option for the implementation of our mechanism.

The central idea is that every peer has to delegate the task of storing and updating its *Rb* value to another peer, like in [21]. Using the unique identifier of the peer and a hash function, a peer can find the peer responsible for storing the value of any other peer in the network. This functionality is offered by the overlay network

The *Rb* management functionality will be encapsulated into a service, this service will be responsible for storing and updating a peer's *Rb*. When a peer joins the group, it must publish an *Rb* service with its id (as was proposed for files in content discovery). To update or query an *Rb* a peer just need to invoke the middleware primitive *getServices*, specifying the id of the peer it wants to update or query in a contract template.

TABLE III. MECHANISM MESSAGES

Message	***Sender***	***Receiver***
Manager offer	Downloader (C)	Manager (C)
Offer answer	Participant (S)	Manager (C)
Acknowledge	Downloader (C)	Manager (C)
Cancel	Manager (C)	Participant (S)
New Download	*user*	Downloader(C)
Task	Manager (C)	Participant (S)
Query Rb	Manager (C)	*Rb manager(R)*
Update Rb	Manager (C)	*Rb manager(R)*

 ISBN: 978-1-61208-173-1

```
// CREATE NEW SMEPP peer
 String configFile = args[0];
 Credentials myCredentials = new Creden-
tials("");
 PeerManager peer =
   PeerManager.newPeer(myCredentials, config-
File);
//Join P2P group (find and join)
 GroupDescription myP2PGroup =
   new GroupDescription("P2PGroup ",
   new SecurityInformation(1), "P2PGroup");

GroupId[] groupIds =
   peer.getGroups(myP2PGroup);
//(...)
 GroupId gid = groupIds[0];
 peer.joinGroup(gid);
//PUBLIHS SERVICES
//Main Service
 PeerServiceId psid = peer.publish(gid,
ContractLoader.loadFromFile("MainService.xml"),
   new SMEPPServiceGroun-
ding(MainService.class),
   null,
   null);
//Rb management service
   psid =
   peer.publish(gid,
   ContractLoader.loadFromFile("RbMgr.xml"),
   new SMEPPServiceGrounding(RbMgr.class),
   null,
   null);
//File Sharing services
foreach (file f in sharedFiles){
  String fContract =
   GenerateContract(f,"SharingS.xml");

  psid = peer.publish(gid,
   ContractLoader.loadFromFile(fContract),
   new SMEPPServiceGrounding(SharingS.class),
   null,
   null);
  //Invoke local service to start downloads
 }
}
```

Figure 5. SMEPP peer code

C. Final Implementation

As has been stated, in our approach each peer can work as participant, manager or downloader. The functionality required is summarised in Table III. The table shows the messages sent between the different services. At least three services need to be defined in order to fulfil the requirements of this application:

- *Main Service (C)*: This service enables the download and the manager functionality.
- *Sharing Service (S)*: This service encapsulates the functionality of participants.
- *Rb Management (R):* This service is responsible for enabling the update and the query of the *Rb* value of each peer.

Each peer must at least publish one *Main Service*, one *Rb Management* service and as many *Sharing* services as files it shares.

The code of a SMEPP peer running our mechanism is illustrated in the Figure 5. For the sake of simplicity we skip base cases, the steps are the following: first we use the *newPeer* primitive to create the SMEPP peer (this connects the peer to the SMEPP network and assigns it an Id). For security, SMEPP requires the provision of valid *Credentials* in order to successfully join the network. The *PeerManager* object allows us to invoke peer's primitives. Next the peer has to find the concrete group where our P2P content distribution mechanism is running, to do this, it performs a search of a group providing its description, using *getGroups* primitive. Once the group is found, the peer joins it with *joinGroup* primitive. Next, the peer publishes all the, previously explained, services. This is performed using the *publish* primitive. Up to this point, we have a peer in the group sharing files, to start a download the user will invoke the local service specifying the file info (*NewDownload* message), this will start the exchange of our mechanism's message between the different peers in the group, performing the coalition and incentives mechanism (as explained in Section III).

To summing up, SMEPP simplifies the implementation of this kind of application, as the above code shows. Not only abstracting the underlying complexity but also offering an efficient look up mechanism for file discovery. Moreover, it tackles the security issues internally, without additional effort.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented the implementation of a coalition and incentive based P2P content distribution system.

Our mechanism is based on game theory and takes into account the rational and self-interested behaviour of the peers. The central idea is that incentives encourage participation; each time a participant contributes in a coalition they receive a reward. The fairness of the rewards division within a coalition is guaranteed by means of the game theory concept of *core*. These rewards are accumulated in the *Responsiveness bonus*, which represents the overall contribution of the peer to the system, and this is used to increase or decrease the quality of service of the downloads the peer performs. This way our approach manages to promote the cooperation, and therefore, reduces the free riding phenomenon. Moreover, simulations showed that download times are improved.

When dealing with the development of real distributed applications, is has been proven that to use a middleware simplifies the implementation issues. In this paper, we proposed to use SMEPP middleware to ease the distributed implementation of the above mechanism. SMEPP is a Secure Middleware for Embedded Peer to Peer Systems and another of our publications.

 ISBN: 978-1-61208-173-1

For the real implementation of our content distribution system different issues have been taken into account. Mainly, two issues must be addressed: content discovery (taking advantage of the middleware overlay), and the storage, query and update of the Responsiveness bonus. Taking advantage of the middleware, the complexity of the development of the distributed application is abstracted, moreover the process of content discovery is delegated in the middleware, what greatly eases the implementation, as foreseen.

As future work, we plan to implement our coalition approach over Gnutella protocol [22]. The objective would be to compare and analyse the performance of these two implementations.

ACKNOWLEDGMENT

This work is partially supported by EU funded project FP6 IST-5-033563 and Spanish projects TIN2008-01942 and P07-TIC-03184.

REFERENCES

[1] Ipoque Internet Study 2008-2009. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009. 26.09.2011.

[2] Saroiu, S., Gummadi, P. K., and Gribble, S. D, "Measurement study of peer-to-peer file sharing system". In Kienzle, M. G. and Shenoy, P. J., Multimedia Computing and Networkin, pp. 156–170. SPIE, San Jose, CA, USA. 2002.

[3] Handurukande, S. B., Kermarrec, A.-M., Le Fessant, F., Massouli´e, L., and Patarin, S. , "Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems". SIGOPS Oper. Syst. Rev., vol. 40, pp. 359-371, 2006.

[4] Karakaya, M., Korpeoglu, I., and Ulusoy, O. , "Free riding in peer-to-peer networks". Internet Computing, IEEE, vol. 13, pp. 92–98, 2009.

[5] Belmonte M.V., Díaz M., and Reyna A., "A Coalition based incentive mechanism for P2P content distribution systems" Proceedings of the 3rd. international conference on agents and artificial intelligence. pp. 15-24. ICAART 2011. Rome, Italy, January - 2011.

[6] Díaz M., Garrido D., Reyna A., and Troya J.M.,"SMEPP: A Secure Middleware for P2P Systems". Horizons in Computer Science. Volumen III. Nova Publisher 2010.

[7] Díaz M., Garrido D., and Reyna A. "SMEPP and the internet of things". In Workshop on Future Internet of Things and Services. CD.ROM, 2009.

[8] Caro, R.J., Garrido, D., and Plaza Tron, P. , "SMEPP: A Secure Middleware For Embedded P2P", 2009. ICT-MobileSummit Conference Proceedings. IIMC International Information Management Corportaion, 2009.

[9] Stoica I., Morris, R., Karger, D.R., Kaashoek, M.F., and Balakrishnan, H., "Chord: A scalable peer-to-peer lookup service for internet applications". In SIGCOMM, pp. 149–160, 2001.

[10] The JXTA home page. www.jxta.org. 26.09.2011

[11] Deliverable 6.4 SMEPP Validation. www.smepp.net 26.09.2011

[12] Kahan, J P. and Rapoport, A., "Theories of coalition formation", L. Erlbaum Associates, Hillsdale, New Jersey London,1984.

[13] Bertsekas, D.P. and Gallager, R.G., and Humblet, P., "Data networks" Prentice-Hall, New York, NY, USA, 1987.

[14] Karakaya, M., Korpeoglu, I., and Ulusoy, O. "A connection management protocol for promoting cooperation in Peer-to-Peer networks". Computer Communications, 31, pp. 240-256. 2006.

[15] Kulbak, Y., Bickson, D., et al.(2005). The emule protocol specification http://www.cs.huji.ac.il/labs/danss/p2p/resources/emule.pdf; 15.06.2011.

[16] Cohen, B. "BitTorrent protocol specification". http://www.bittorrent.org/beps/bep_0003.html 26.9.2011

[17] Garbacki, P., Iosup, A., Epema, D., and van Steen,M. "2fast : Collaborative downloads in p2p networks". In Peer-to-Peer Computing, IEEE International Conference on, pp. 23–30. IEEE Computer Society, Los Alamitos, CA, USA. 2006.

[18] Karakaya, M., Korpeoglu, I., and Ulusoy, O. "Counteracting free riding in Peer-to-Peer networks". Computer Networks, 52, pp. 675–694. 2008.

[19] Mekouar, L., Iraqi, Y., and Boutaba, R. "Handling Free Riders in Peer-to-Peer Systems". Agents and peer-to-peer computing: 4th international workshop, AP2PC 2005, Utrecht, The Netherlands, July, pp. 58–69. Springer-Verlag, New York, NY, USA. 2006.

[20] Zhou, R., Hwang, K., and Cai, M. "GossipTrust for Fast Reputation Aggregation in Peer-to-Peer Networks" IEEE Transactions on Knowledge and Data Engineering, CA, USA, pp.1282-1295. 2008.

[21] Kamvar, S.D., Schlosser, M.T., and Garcia-Molina, H.. "The Eigentrust algorithm for reputation management in P2P networks". In Proceedings of the 12th international conference on World Wide Web (WWW '03). ACM, New York, NY, USA, pp. 640-651. 2003.

[22] "Gnutella Protocol Specification". Online. http://www. stanford.edu/ class/cs244b/gnutella_protocol_0.4.pdf.26.09.2011.

 ISBN: 978-1-61208-173-1