Self-Regulation in P2P Systems

Ion Craciun
ion.craciun@tecnico.ulisboa.pt
Student Number 66999

Instituto Superior Técnico

Abstract. Peer-to-peer systems became known to general public due to the intensive use of file-sharing. P2P systems can be used for sharing any type of resources between users, but expecting resources in return. There is no real currency in the system, and we need to guarantee that every user actually contributes to the system otherwise it will become depleted of resources. Our objective is to define an abstract of a P2P system and the properties that the system should reach (liveness) and those that should never occur (safety). Then we need to verify that these properties satisfy our system and provide formal guarantees. To achieve this we started by studying the incentive systems, systems that try to motivate the users to contribute to the system, then we studied the existing verification frameworks that were used to prove certain properties in P2P systems.

1 Introduction

Nowadays peer-to-peer systems are known to general public due to intensive use of file-sharing. However, P2P systems can be used for sharing any type of resources between users, including CPU and memory for the execution of tasks, jobs, virtual machines. A P2P network distributes information among the peers instead of concentrating it at a single server, bringing new advantages in information sharing but also presenting challenging disadvantages. One of the biggest disadvantages in P2P systems is the resource sharing. In a P2P system every user can act both as a client and as a server, and in a voluntary or altruistic way he must contribute to the system with resources. Not everyone can or wants to contribute in the system. Many times a user just takes the resources he wants from the system and leaves it. We call these users free-riders. A lot of free-riders in the same network leads to a client-server architecture and decreases the advantages of a P2P system. Our objective in this work is to define an abstraction of a system and study the *liveness* (propertie that a system should reach) and *safety* (property that should never occur) properties of this system that could demonstrate the feasibility of community clouds.

To have a P2P system with no free-riders, we must motivate the users not only to consume resources but also to start sharing them. To achieve this, we studied the existing incentive systems and figured out how to handle out the free-riders. We approach incentive systems based on Game Theories, Trust and

Reputation. In our solution we use the incentive system proposed by Rodrigues *et al.* in [20]. They studied incentive mechanisms for P2P voluntary cycle systems, which enable the correct operation of the systems, imposing a balance between demand for resources and the existing offer. They explored concepts such as reputation and currency, which are used in other systems, enabling a coherent scheme to detect untrustworthy users and reward truthful peers with faster access to the resources. Their verification was however done simply by simulation and no formal guarantees were given.

Our second part of this work was to search and study the verification techniques for a given property in a given system. To define and prove the properties that reach the *liveness* and *safety* of a system, we had to study other systems and properties to understand how it was done. We studied in more details the verification of the security properties proposed by Mark Ryan in [5] using CSP [7] and the verification of the Pastry Protocol in [15] using TLA$^+$ [13].

Finally in the end we have our new system based on [20], and the properties that we need to study and prove using TLA$^+$.

The rest of paper is structured as follows. In Section 2 we have the objectives of this work. Section 3 is the Related Work, with the incentive systems and verification techniques we studied. In Section 4 is and abstract architecture of the new system and the properties that the system should reach. Finally we have Section 5 that concludes our work.

1.1 Peer-to-Peer

A small introduction to the peer-to-peer systems and several P2P architectures concepts that are used later in this work and we explain them in this part.

What is peer to peer? Peer-to-peer (P2P) network is a distributed application that distributes information among the member nodes (peers) instead of using a centralized server. Peers are equal in their capacity for sharing information with other network nodes and they can act as server or client. Each peer has some information available for distribution, and also can establish connections with any other node to download information. The ideal scenario for a P2P network is when a node does not consume more resources than the shared ones. In reality this is hard to achieve as nodes can consume data and leave the P2P network at any time without sharing resources. Nevertheless this model offers advantages in information sharing but also has challenging disadvantages.

1.2 Peer-to-peer architectures types

Centralized Napster is one of the famous examples to characterize a centralized P2P network. Napster has a central constantly-updated directory at a central location (e.g Napster site). Nodes from the network will query the directory to find which nodes from the network hold the data they need. Such centralized

approach does not scale very well and has some failures (e.g we can perform attacks on the central directory) [16].

The decentralized P2P architectures are divided in two categories:

Decentralized structured These systems have no central directory like Napster, but they have some structures in the network. By structure we mean that data, nodes and connections between nodes are organized such way that queries in the network are easier to satisfy.

Decentralized unstructured These systems have no central directory and also have no structures in the network. The network is formed by nodes joining the network following some rules. The data is randomly distributed, so for a node to find a file it must flood his neighbors with queries. These systems are resilient to nodes entering and leaving the network but the search mechanisms used are extremely unscalable.

2 Objectives

Peer-to-peer file-sharing networks are currently receiving much attention as a means of sharing and distribution information. However, the anonymous, open nature of these networks made them to be the target of many attacks and threats. The main threat for P2P system are the free-riders, peers that consume more resources than the provided ones. The ideal P2P system would be a system with no free-riders. However, this is almost impossible to reach, as there is always someone that wants to abuse the system. The objective of this work is to study the P2P systems, and define some properties that a system should reach (*liveness*), and those that should never occur (*safety*). To achieve this, we need to do the following things:

1. Analyze the existing P2P architectures and see how the peers, the communication between them, and the resources are organized in the system.
2. Assess all the attacks and threats that a P2P system can be exposed, and choose the relevant ones for our study. We are interested in the attacks and threats that abuse the resources of the system.
3. Study the existing incentive systems to discover how to reduce the threats and attacks over a P2P system and motivate the users to participate and contribute in the system.
4. Search for verification frameworks that were used to prove properties of P2P systems. We must analyze and understand how these properties were proven by the given framework.
5. Model our P2P system, the associated incentive system, and prove the intended properties using the chosen framework.
6. Try to improve existing systems by developing better incentive systems that satisfy the studied properties.

The first four objectives form the core of our Related Work. We start by describing famous P2P architectures that are widely use. Next we present the attacks and threats that are relevant for our work, and the incentive systems that try to reduce the attacks and motivate the users to participate in the system. Finally we approach the verification framework and present examples of usage to prove specific properties in a given P2P system.

Our architecture and the future work is based on the last two objectives. In Section 4 we present our architecture and the properties that we want to prove.

3 Related Work

In this section we start by approaching some specific P2P architectures. We then describe the major attacks and vulnerabilities that a peer-to-peer network can be exposed and present the incentive systems that try to reduce the attacks. In the end we present the verification techniques that were used to prove certain properties of P2P systems.

3.1 Architectures

Chord Nowadays one of the problems in peer-to-peer systems is to efficiently locate the node with a particular data we need. Chord is a distributed protocol that approaches this issue and provides support for just one operation: *given a key, it maps the key onto a node* [22]. Chord tries to simplify the design of P2P systems and addresses some of the problems as load balance, scalability, decentralization, availability and flexible names. To assign keys to Chord nodes, a variant of consistent hashing is used. Consistent hash function is one that changes minimally as the range of the function changes [11] (e.g to resize a hash table with consistent hashing, only K/n keys need to be remapped on average, where K is the number of keys, n is the number of slots). To address the load balance problem, Chord uses consistent hashing and every node receives roughly the same number of keys, and that's why Chord adapts efficiently as nodes join or leave the system.

Nodes and keys are assigned a **m-bit** identifier using consistent hashing. Node's identifier is chosen by hashing his IP while key's identifier is a result of hashing the key. Using Chord protocol, nodes and keys are ordered in an identifier circle that has at most 2^m nodes with range 0 - $2^m - 1$ and m large enough to avoid collisions. Each node has a *successor* and a *predecessor*. The *successor* node is the next node in the identifier circle in a clockwise direction. The *predecessor* node is the next node in a counter clockwise direction. In figure 1 we have an example with a an identifier circle with $m = 3$. The circle has three nodes: 1, 5 and 6. The *successor(1)* = 1, so the key would be located at 1, key 3 at node 5 and key 7 at node 1.

Pastry Another solution to locate efficiently a node with particular data and efficient routing within the network is Pastry. Pastry is a generic peer-to-peer ob-

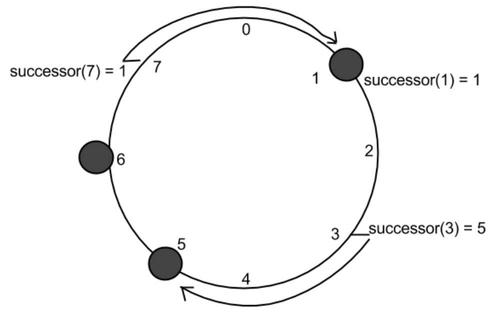

Fig. 1: Chord network

ject location and routing scheme, based on a self-organizing overlay network. Pastry is completely decentralized, fault-resilient, scalable, reliable and with good route locality properties [19]. In this system each node has a unique numeric identifier (nodeID) and each node maintains a routing table, a neighborhood set and a leaf set. As nodes keep track of its immediate neighbors, they notify applications of new node arrivals, node failures and recoveries. Since these IDs are randomly assigned, the set of nodes with adjacent nodeID is diverse in geography, ownership, jurisdiction, etc. Applications like PAST [18] can leverage this, as Pastry can route to one of k nodes numerically closest to the key. When Pastry receives a message and a numeric key, it routes the message to the node with a nodeID that is numerically closest to the key, among all currently working Pastry nodes.

3.2 Attacks and threats

Free Riding A free-rider is a user that exploits the P2P network resources but does not contribute to the network. A free-rider is considered a threat for P2P networks. A large number of free-riders can cause the degradation of the system performance, and augment the vulnerabilities of the system. There are several motivations behind free riding:

- peers with a Network Address Translation (NAT) may act as a free rider: a lot of computers share the same domain IP through **NAT** and if two peers are using **NAT**-based IP, they cannot download files from each other and also cannot upload files, which makes them to be considered free-riders [10].
- bandwidth limitation is another cause: there are peers that have scarce bandwidth and sometimes we assume they don't want to share it by uploading files. In [1] we can see that there is no strong correlation between bandwidth and free riding. A peer with a large amount of bandwidth has the same tendency to be a free rider as a peer with poor connection.

- users are concern of sharing illegal data on their personal computers: peers don't want to be responsible for the data item on any occasion of surveillance, and in many cases users only download content to use it for a short time and delete it afterwards.
- users are concern of security issues if they share something: sometimes peers are just afraid to share resources as they think other peers may discover and attack their personal computer.

There also are several types of free-riders:

- A user does not share anything at all or shares uninteresting files: if a peer does not responds to requests from other peers, there may be two reasons for that, the peer does not have the files that other peers request or the peer does not share any files at all [10].
- A peer consumes more resources than the ones he shares: we can discover a peer that consumes more resources than those the he shares by counting the affirmative responses to other peers, and the requests made by him to its neighbors, and comparing them.
- A peer drops the queries from other peers: if a monitoring peer counts the requests and responses for a neighbor, and they are low, it can be assumed that the neighbor has few connections or drops the queries.

There are several attacks that a peer can perform to conceal himself. One of them is reply with fake messages to their neighbors when he is asked to share some files: he says that he has the file but when asked to share it, he refuses the connection. Another attack is sharing dummy files with popular names in order to cheat querying peers. A lot of free riders in the same network may lead to scalability problems as a lot of download requests will be directed to the same peers. This will lead to a client-server architecture and decrease the P2P network advantages.

Sybil attack / Whitewashing The main idea behind Sybil attack is that a single user can present multiple identities and abuse the system. He can corrupt the files that pass through his peers, he can reroute all queries in the wrong direction thus slowing down the network. When a single user leaves and rejoins the network under a new identity to avoid system penalties, we call him a *whitewasher*. A free-rider chooses to be a *whitewasher* in the network when the system has low cost of acquiring new identities. A solution is to impose a penalty on all newcomers, since it is impossible to distinguish a *whitewasher* from a newcomer. Another solution to avoid these attacks is to have in the system a trusted identification authority [6] (e.g a group of peers that vouch for the newcomer).

Malicous attacks Peer-to-peer file sharing networks have advantages over client-server approaches but the open and anonymous nature of these networks open a door for the malicious peers that want to abuse these networks. The difference between a free-rider and a malicious peer is that the free-rider only

consumes the resources offering nothing in change while a malicious peer tries to infect the network, for example with a virus, to take control over other peers [8].

As P2P network are large networks with a lot of nodes, there is the possibility to find malicious collectives. A malicious collective is a group of malicious peers that know each other and try to cheat the trust incentive systems by giving each other high local trust values and give all other peers low values.

Poisoning and pollution Nowadays peer-to-peer file sharing systems are the predominant sources of Internet traffic and a lot of copyright holders are worried by the potential loss of revenue due to file sharing. They have been exploring several options to decrease the availability of the items that belong to them (e.g, movies, songs, games) [3]. A common technique that they are using to decrease the availability is item poisoning. Item poisoning corresponds to injecting corrupted files into the network with name and meta-data (e.g movie name, duration) as the original item. Pollution, similarly to poisoning, injects corrupted files into the network but has different objectives and impacts on the systems. The goal of pollution is to create noise in the network while poisoning usually targets a specific file in the network and tries to decrease its availability.
There are three strategies to inject bad files:

- **Random injection** - At lower levels, this strategy can be considered as a pollution attack. At higher levels this strategy is inefficient as flooding the network with random bad files does not change the availability of the most popular files. In fact, to successfully poison the network using this strategy we need to inject a big number of corrupted files.
- **Replicated injection** - next strategy is injecting numerous replicas of the same corrupted file. This poisoning attack is easily countered by a reputation system.
- **Replicated transient decoy injection** - we can cheat the reputation system by frequently replacing the replicated corrupted files injected in the network. It does not change the temporal stability properties of network so significantly, and may not leave an obvious statistical signature, which makes this strategy hard to detect.

All this strategies are not mutually exclusive. We may combine them and create very strong poisoning attacks that will be difficult to detect and will lead to drastically decreasing the content availability of the specific file.

3.3 Incentive systems

In this section we present the incentive systems that we studied and are countermeasures against the attacks previously described. The goal of these systems is also to motivate the peers to contribute and participate in the P2P networks. The incentive systems must consider several issues:

- **Simplicity** - the actions observed and reactions to them should be simple to implement and manage.

- **Decentralise** - making decisions and taking actions should be executed in a decentralized way.
- **Cooperation** - should be intensified with coordination among peers.
- **Low overhead** - methods should not cause much overhead. Non free-riders should not devote much resources to prevent free riding.
- **Abuse-proof** - peers may try to walk around mechanism by misreporting their status or implementing their own client programs. Mechanism must not depend on information provided by peers solely.
- **Fairness** - peers with low bandwidth connections may not contribute even when they are willing to do so. The peers with NAT-based IPs also behave like free-riders. Mechanism and policies applied against free-riders should be fair and smart enough to distinguish the peers which are not real free-riders.

Mechanism against free riders The first incentive system we present, is a simple system in which every peer passively monitors the other peers proposed by [10]. In this system every peer has two roles. The first role is a monitor peer that monitors and records the number of messages coming from and going to neighboring peers [10]. The second role a peer can take is controlled peer, which means his messages are monitored and recorded by his neighbors. After examining the messages from a neighbor, and compiling the information recorded about the neighbor and its related messages, a monitoring peer may suspect if is a free-rider. Then he can take counter actions against this suspected peer. In figure 2 we have an example how the peers can be connected and monitor each other.

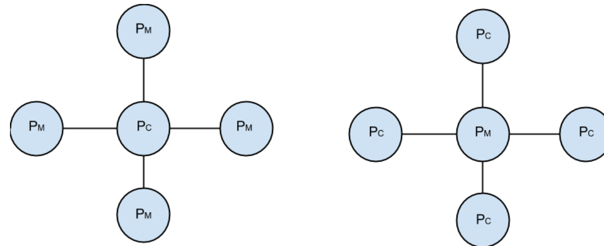

Fig. 2: Peers in two roles: monitoring and controlled

In order to determine if a controlled peer is a free-rider or not, several clues must be derived from the behaviors of the neighbors. Each peer needs to maintain information about each neighbor and its behaviors. The information that each peer maintains about neighbors consists of some statistical counters presented in table 1 [10]. The clues about his neighbors, if they are free-riders or not, are derived from these values. If a peer identifies another peer as a free rider it can take some counter-actions against it:

Symbol	Description
Q_p	Number of Query descriptors submitted by peer P.
RQ_p	Number of Query descriptors routed by peer P.
TQ_p	Number of Query descriptors routed towards peer P.
QH_p	Number of QueryHit descriptors submitted by peer P.
RQH_p	Number of QueryHit descriptors routed by peer P.
SQH_p	Number of QueryHit descriptors satisfying queries of peer P.
N_p	Number of Notify descriptors submitted for peer P.

Table 1: Descriptors observed

- Decrementing TTL(Time to Live) value - when a peer receives a request from his neighbors, first he searches on local files for a match, otherwise forwards the request to the other neighbors peers. Before forwarding the request, the TTL value is decremented by one. If the peer suspects that the request comes from a free-rider, he can decrement the TTL value by more then one before forwarding. This reduces the range of free-riders to search the files they need and also reduces the overhead of the network.
- Ignoring request - a monitor peer can punish a free-rider by simply ignoring his request, simply by dropping the query from the free-rider. In this case the monitoring peer should be very careful about the origin of the request and drop only the requests coming from free-riders.
- Disconnecting from network - in this case if a monitor peer is 100% sure that his neighbor is a free-rider, he drops the connection with him. The difference from ignoring requests is that here the free-rider needs to disconnect and reconnect to the system through a new peer, while ignoring him we can change our opinion if we see any change in the behavior.

We saw how to detect the free riding types and the possible countermeasures. Now we integrate them together using a ECA (Event Condition Action) rule and a FSM (Finiti State Machine). As we saw in Section 3.2 a free-riding peer can be a consumer,a non-contributor, a dropper or a combination of these. If a neighbor is a good behaving peer, it will not show any of the mentioned free riding types. To denote if a neighbor belongs to these categories, three boolean variables are used. N for non-contributor, D for dropper and C for consumer. If a peer has the boolean, for example D (dropper), equal to 1 than we can say it is a dropper, otherwise he is not a dropper. With these booleans and with the values maintained in the log table of a monitoring peer, we can have one of the following eight conditions (Table 2) for a neighboring peer.

As we can see in the table 2, $C0$ gives us the information that there are no free-riders in the system. $C1$, $C2$ and $C4$ gives us information that the peer P is either a non-contributor, a dropper or a consumer. In $C3$, $C5$ and $C6$ the controlled peer exhibits exactly two types of free riding. Condition $C7$ gives us the information that the peer is showing all types of free riding.

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

Table 2: Conditions

The monitoring peer based on these conditions takes the countermeasures against them. For condition $C0$, as the peer shows no free-riding behavior, the monitoring peer applies no counter-action. For the conditions $C1$, $C2$ and $C4$, the monitoring peer reduces the TTL value by more than 1. For $C3$, $C5$ and $C6$ the monitoring peer ignores partially the requests. Finally for condition $C7$ the monitoring peer just drops the connection with this peer. We can integrate these conditions in a FSM with four states: $S0$, $S1$, $S2$ and $S3$. In each state a countermeasure is applied,except in $S0$ where the peer is a non free-rider and so no counter-actions need to be applied. In $S1$ the countermeasures for $C1$, $C2$ and $C4$ are applied, $S2$ for $C3$, $C5$ and $C6$. Finally in state $S3$ the countermeasure for C7 are applied. The transition between states depend on the condition (values of N,D and C) changes upon an update.

Game Theory incentive systems Peer-to-peer systems are self-organizing, distributed resource sharing networks. There is no central authority to command or coordinate the resources that each peer should contribute [2]. Every peer in the systems is a volunteer and that is why the system's resources can be highly variable and unpredictable. For example, one of the problem as we saw in the previous section is the free-riding.

The goal of the game theory incentive systems is to develop mechanisms by which contributions of individual peers can be solicited and predicted. We assume that each individual (peer) is a rational, strategic player who wants to maximize his utility by participating in the P2P system. Next we present some specific game theory incentive systems.

Evolutionary Prisoner's Dilemma (EPD) consists of players (peers) who meet for the games. Each of them has a score which is initialized to 0. In each game, one player is the client and one player is the server. They can swap roles in the next game. The client selects the server using a strategy. Client and server each have the choice of cooperating or defecting. Players decide whether to cooperate or defect using a strategy. They can observe each others actions, but not their

strategies. A player may maintain a history of other players actions. Some of the strategies that exist in P2P systems are 100% cooperate or 100% defect. After client and server's action, a payoff from a payoff matrix (Table 3) is added to each of their score [12]. R, S, T and P in the payoff matrix stand for *reward*, *sucker*, *temptation*, and *punishment*. Each of these values can be positive or negative.

Client\Server	Cooperate	Defect
Cooperate	$\mathrm{R_c/R_s}$	$\mathrm{S_c/T_c}$
Defect	$\mathrm{T_c/S_s}$	$\mathrm{P_c/P_s}$

Table 3: Payoff matrix

The payoff matrix models the benefits and costs of a game, and should meet the following requirements and associated inequalities [12]:

- Mutual cooperation is better then mutual defection $R_\mathrm{s} + R_\mathrm{c} > P_\mathrm{s} + P_\mathrm{c}$.
- Mutual cooperation must be better em terms of payoff than one player suckering the other $R_\mathrm{s} + R_\mathrm{c} > S_\mathrm{c} + T_\mathrm{s}$ and $R_\mathrm{s} + R_\mathrm{c} > S_\mathrm{s} + T_\mathrm{c}$.
- Defection dominates cooperation at the individual level $T_\mathrm{s} + P_\mathrm{s} > R_\mathrm{s} + S_\mathrm{s}$ or $T_\mathrm{c} + P_\mathrm{c} > R_\mathrm{c} + S_\mathrm{c}$.

In a P2P system, 100% Cooperators drive the system to high overall *utility* while 100% Defectors gain more benefit from the system than 100% Cooperators.

Tit-for-tat is a strategy that is believed to be the most effective to enforce the collaboration among selfish users [14]. This strategy is alike EPD, a player must defect or cooperate based on the actions from the other player. The difference is that in tit-for-tat a player initially cooperates, and then responds with the action that his opponent used in previous action. *Tit-for-tat* is applied as a strategy to ensure fairness in distributed networks such as P2P systems that do not have a centralized control facility, and where the users are most of the times selfish players. For example, *tit-for-tat* serves as a design principle for *BitTorrent* [4]. In this system, peers use *tit-for-tat* strategy to provide parts of a particular file as long as they receive something of interest in return. Fairness in P2P system is very important, as any system's performance depends on collaboration.

As we said *tit-for-tat* is considered to be the most effective and successful strategy to enforce the collaboration among selfish users, but in practice it has several problems. In a perfect *tit-for-tat* environment a peer is provided with a certain contribution if it continues to give something in return. In a real P2P system, these things are almost impossible. First we have the new peers joining the system, and they need to download some files or parts of them before uploading them and cooperate with other peers. Second, peers can have neighbors that have no usefull information for them, information that they already have.

In [14] the authors solve these two problems by using a *Seeder Strategy.* The main purpose of this strategy is to boost the download progress by providing newcomers with initial data they can share. This data are sets of blocks, that any new peer can download for free and start interacting with other peers and contribute to the network. Every peer is identified by his IP address and consequently a peer is not able to receive additional data for free, nor leave and rejoin the system. Every newcomer receives different data based on his IP. This strategy has several benefits. First, peers with different IP addresses obtain different data blocks which ensures that the available number of distinct linear combinations in the network is high. Second, the leechers (peers downloading data) are forced to collaborate as seeders provide a small and specific set of blocks and do not provide more data until they receive something in a change from the leechers.

Nash Equilibrium is a game theoretic notion to analyze the strategic choices by different peers. The classical concept of *Nash Equilibrium* points a way out of endless cycle of speculation and counter-speculation as to what strategies the other peers will use [2]. An equilibrium point is a locally optimum set of strategies (contribution levels in our case), where no peer can improve his utility by deviating from the strategy. While Nash equilibrium is a powerful concept, computing these equilibrium is non trivial. No polynomial time algorithm is known for finding the Nash equilibrium of a general N person game.

In a traditional distributed system we assume that all participants work together cooperatively and share a common goal, do not compete with each other, nor try to subvert the system. In a P2P system, the things are different as the system consists of autonomous components. Users compete for shared but limited resources and at the same time, they don't contribute with resources. We can define these interactions between peers as a *non-cooperative game* among rational and strategic players. The players are rational because they wish to maximize their own gain, and they are strategic because they can choose their actions that influence the system [2]. Players obtain benefits from their interactions between them and this benefit is termed as a payoff or *utility.* The *utility* depends not only on their own strategy, but also on everybody else's strategy. Sometimes a player might decide to switch strategy to improve his *utility* but this switch can affect others players *utility.* The collection of players is said to be at Nash equilibrium if no player can improve his utility by unilaterally switching his strategy [2].

Now imagine a real P2P systems with users interacting between them. A peer P_i has a limited set of peers that interacts with him. During these interactions, peer P_i learns of the contribution made by them and tries to maximize his *utility* by adjusting his own contribution. This contribution from P_i is not globally optimal because is limited to the information only from a set of peers. After setting his own contribution, he propagates this information to the interacting peers and they will adjust their own contribution. The reaction of the peers to P_i's contribution can affect P_i once more, and he can readjust better his contribution. Every peer goes through an iterative process of setting its own

contribution. If and when this process converges, the resulting contributions will constitute a *Nash Equilibrium* [2].

Trust/Reputation incentive systems In an open P2P information system, peers often have to interact with unknown peers and need to manage the risk that is involved with the interaction without the presence of trusted third parties or trust authorities. Trust is an important measure that a peer can use to evaluate the risk involved when interacting with other peers and make the right decision about requests from them.

PeerTrust is a simple and effective trust model for quantifying and assessing the trustworthiness of peers. The PeerTrust model has two unique features. First, has three important factors for evaluating the trustworthiness of a peer [24]:

- The amount of satisfaction a peer receives regarding his service, it usually results from the interactions that other peers have with this peer.
- The total number of interactions that a peer has with other peers in the P2P network is another important factor that affects the accuracy of feedback-based trust computation. Reflects over how many services the satisfaction is obtained.
- Balancing factor of trust is used to offset the potential of false feedback of peers and thus can be seen as a way to differentiate the satisfaction of credible sources from the less credible ones.

All three factors play a crucial role in the computation trust of peers because many other models use only the first factor. Second, PeerTrust has a general trust function that computes the trust of a peer by combining these three parameters. We also want to point out two important concepts addressed in PeerTrust: trusting belief and trusting behavior. Trusting belief between two peers is when one peer believes that another peer is trustworthy. Trustworthy means that the peer is willing and able to act in other entity's best interest. Trusting behavior between two peers is when a peer depends on another peer in a given situation with feeling of relative security. PeerTrust uses the trusting belief relationship. Most of the times trusting belief leads to trusting behaviour. In PeerTrust, a peer's trustworthiness is defined by an evaluation of the peer in terms of the degree of satisfaction it receives in providing service to other peers in the past.

EigenTrust An important example of successful reputation management is the online auction system eBay. In eBay's reputation system buyers and sellers can rate each other after each transaction, and the overall reputation of a participant is the sum of these ratings over the last 6 months [9]. For example, each time peer i downloads a file from peer j, it may rate the transaction as positive or negative. Peer i may rate a download as negative, if the downloaded file is corrupted or if the download is interrupted. Each peer i can store the number satisfactory transactions it has had with peer j and the number unsatisfactory transactions it has had with peer j.

The basic concept behind EigenTrust is that each peer j is assigned a global participation value, or EigenTrust score, that is given by the sum of the local participation values assigned to peer j by the peers who have interacted with him. A simple way to keep this value would be for each peer to store it, and report its score when asked by another peer. The problem is that malicious peers can misreport their scores or create a collective and boost each other score. The solution would be to create some pre-trusted (for example, designers of the network) peers that are responsible for storing and managing the scores of their children.

Two ways to reward participatory peers is to award them faster download times, and grant them a wider view of the network. The first incentive is to give active participators preference when there is competition for bandwidth. In other words, if peer j and peer i are simultaneously downloading from another peer k, then the bandwidth of peer k is divided between peer i and peer j according to their participation scores. The second incentive is to assign each peer a TTL based on its participation score, giving active participators a wider view of the network.

Another incentive that deserves some attention is changing the topology of the network based on participation scores. The idea is to connect peers with high participation scores to one another to form the top ring of the P2P network. Those peers with lower EigenTrust score will make their own ring. This EigenTrust-based topologies will move malicious peers to the fringe of the network, limiting the interactions of good peers with malicious peers.

GINGER is a large project, Grid In a Non-Grid Environment, a P2P infrastructure intended to ease the sharing of computer resources between users. Authors main concern in [20] is related to the overall fairness of the system. They want to guarantee that malicious users will not corrupt the entire system by not contributing to the resource pool or delivering false results. Their aim is the balance between a currency and reputation-based system capable of delivering proper incentives to users. They conclude that a good incentive-based application must rely on both reputation and currency and also the existence of brokers that manage a number of peers.

The solution was developed on top of the Pastry and Ginger [23]. The network is represented as a ring of peers connected to their neighbors. Virtual groups of users were implemented to structure the network with a broker assigned to each group. The existence of brokers makes the architecture to be classified as partially hierarchical and structured.

To initialize the system, a set of trustworthy users is needed, to act as super nodes. To be classified as trustworthy, these users need to reside in the system long enough. Brokers will be elected from the pool of peers, according to their reputation. The brokers are the only users capable of introducing currency in the system and are responsible for registering an additional amount of currency to the most reputed users upon the entrance of a new peer [20]. Periodically, to promote activity, a broker adjusts downwards the reputation of a user.

3.4 Verification Techniques

Nowadays a lot of properties verifications are done simply by simulation and no formal guarantees are given. In this section our objective is to study the existing frameworks for verification and the properties they prove. We detach two frameworks. First one is TLA$^+$, that was used for the verification of the Pastry Protocol in [15]. Second framework is CSP used by Mark Ryan in [5] to verify the security properties of P2P systems.

Verification of the Pastry Protol using TLA$^+$ TLA (Temporal logic of actions) was invented by Leslie Lamport and can describe a system by a single formula. TLA provides a nice way to formalize the style of reasoning about systems, known as *assertional* reasoning, that has proved to be most effective in practice. One of the disadvantages of TLA is that it does not have notations for writing long formulas. Mathematicians have developed the science of writing formulas, but they never turned that science into an engineering discipline. The notations they developed were for the mathematics in the small but not in the large. As the specification of a real system can be hundred of pages long, and mathematicians know how to write 100-line formuas, not 100-page formulas, Lamport had to introduce notations for writing long formulas. A new language was born, called TLA$^+$. TLA$^+$ is good for specifying a wide class of systems from program interface to distributed systems. It can be used to write a precise, formal description of almost any sort of discrete systems [13].

In [15] authors modeled Pastry's core routing algorithms and communication protocol in TLA$^+$. To validade the model and to search for bugs, they employed the TLA$^+$ model checker TLC to analyze several quality properties.

In Pastry two of the most important sub-protocols are *join* and *lookup*. The join protocol adds a new node with an unused network ID to the ring. The lookup protocol delivers the hash table entry for a given key. An important property of Pastry that requires that there is always at most one node responsible for a given key is *Correct Key Delivery*. This property is non-trivial to obtain as nodes can leave, drop or join at any moment. Therefore every node holds two *leaf sets* of size l containing its closest neighbors to either side (l nodes to the left and l to the right) and the hash table content of its leaf set neighbours. In [15] authors had several challenges related to Pastry. First challenge in modeling Pastry was to determine an appropriate level of abstraction. They focused the model towards supporting detailed proofs of the correctness properties. They prescind from the notion of time because it does not contribute to the verification of correctness properties. The second challenge was to fill in the necessary details for the format model that are not contained in the published descriptions of Pastry. Another challenge and the main property that the authors were interested is that lookup message for a particular key is answered by at most one ready node covering the key. A ready node is a prepared node to answer lookup and join protocols.

The join protocol is the most important part of Pastry correctness. A new node joins the ring between two ready nodes. This new node receives its leaf

sets from ready nodes, negotiates with both nodes the new leaf sets and goes to status ready.

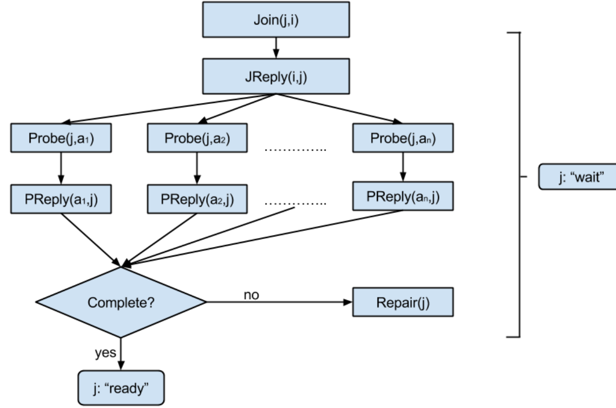

Fig. 3: Join protocol

In figure 3 we have an example of join protocol. Node j wants to join the ring by performing a *Join* action, and keeps its state in status *wait*. Its join request will be handle by a ready node i, that replies with a *JReply* action, transmitting its current leaf sets to enable node j to construct its own leaf set. Node i *probes* all nodes in its leaf sets to confirm their presence in the ring. All the nodes from leaf sets reply with *PReply* action that signals j that the respective leaf set node received information about j and updated its local leaf with j. After receiving a probe reply from every node from the leaf sets, j updates the local information based on the received messages. When the leaf set is *complete*, j goes to status ready, otherwise goes to *repair* if any fault happened.

The overall structure of the TLA$^+$ specification of Pastry is presented next:

$$vars \triangleq \langle receivedMsg, status, lset, probing, failed, rtable \rangle$$

$$Init \triangleq \land receivedMsg = \{\}$$

$$\land status = [i \in I \mapsto \text{IF } i \in A \text{ THEN } \text{"ready"} \text{ ELSE } \text{"dead"}]$$

$$\land lset = [i \in I \mapsto \text{IF } i \in A$$

$$\text{THEN } AddToLSet(A, [node \mapsto i, \ left \mapsto \{\}, right \mapsto \{\}])$$

$$\text{ELSE } [node \mapsto i, left \mapsto \{\}, right \mapsto \{\}]]$$

$$\land probing = [i \in I \mapsto \{\}]$$

$$\land failed = [i \in I \mapsto \{\}]$$

$$\land rtable = \ldots$$

$$\begin{aligned} Next \triangleq \exists i, j \in I : & \lor Deliver(i, j) \\ & \lor Join(i, j) \\ & \lor JReply(i, j) \\ & \lor Probe(i, j) \\ & \lor PReply(i, j) \\ & \lor \ldots \end{aligned}$$

$$Spec \triangleq Init \land \Box[Next]_{\mathrm{vars}}$$

TLA formulas that represents programs can always be written in the same form, for example, $Init \land \Box [N]_{\mathrm{f}}$ where:

- *Init* is a predicate specifying the initial values of the variables.
- $\Box$ is an unary operator read as *always*.
- N is the program's *next-state relation*, the action whose steps represent executions of individual atomic operations.
- f is the n-tuple of all flexible variables.

In our case the system specification *Spec* is defined as $Init \land \Box [Next]_{\mathrm{vars}}$. It requires that all runs start with a state that satisfies *Init*, which is the initial condition, and every transition either does not change *vars* or corresponds to a system transition defined by *Next*. As $Init \land \Box [N]_{\mathrm{f}}$ is a safety property, we can see that this system specification is only for proving safety property. To prove the liveness properties of this model, fairness hypotheses should be added to assert that certain actions eventually occur. Authors left the liveness properties for the future work, when they tend to extend the model. The full proofs are available on Web[1].

Verification of security properties in P2P systems using CSP Communicating Sequential Processes is a formal language for describing patterns of interaction in concurrent systems. CSP was introduced by C.A.R Hoare in his paper [7] in 1978 and is a concept of a system of processes interacting by sending messages to each other via *handshaken* communication [17]. CSP was the first version of all these languages called *process algebras*. Many researchers use the original version, others built upon its ideas to develop their own languages and notations. The majority of these languages have been notations for describing and reasoning about purely communicating systems. Process algebra notations and theories of concurrency are useful because they bring the problems of concurrency into sharp form [17]. We can use them to address the problems at the high level of constructing theories of concurrency and at the lower level specifying and designing individual systems.

[1] `http://www.mpi-inf.mpg.de/~tianlu/software/PastryTheoremProving.zip`

In [5] authors focus on a specific security property, called the *root authenticity* (RA) property of the structured P2P overlays (e.g Chord, Pastry). They propose a P2P architecture that uses Trusted Computing as the security mechanism, formalize the system using CSP, and then verify that it indeed meets the RA property. A P2P system satisfies the *root authenticity* (RA) property when it solves the attack in which the adversary falsely claims that he is the destination of a search key k.

There are a lot of structured P2P systems, each differs from another in its topology, routing or maintenance protocols. In [5] authors consider Chord as it is more popular than the other systems due to its simplicity and efficient routing protocol. In systems like Chord, an attacker may attempt to impersonate the destination of a search key. For example an attacker controls a specific peer and can convince another honest peer that his node is the destination for the data that comes from the honest peer. The RA property implies that such an attack is not possible. The definitions of *root authenticity* (RA) property and *neighbor authenticity* (NA) property are the following:

Root Authenticity (RA): Let P^{t} be the set of current nodes in the P2P system, at a given time t. Assume that the system evolves from t to $(t+1)$ as a new peer joins or and existing peer leaves the system. The RA property is defined as [5]:

$$\forall D, k \in ID, t. V.destVerification(k, D)$$

$$\implies D \in P^{\mathrm{t}} \wedge (\forall D^{'} \in P^{\mathrm{t}} \backslash \{D\}. |D^{'} \ominus k| > |D \ominus k|)$$

Neighbor Authenticity (NA): Let P^{t} be the set of current nodes in the P2P system, at a given time t. Assume that the system evolves from t to $(t+1)$ as a new peer joins or and existing peer leaves the system. The NA property is defined as:

$$\forall L, D, t. V.neighborVerification(L, D)$$

$$\implies L, D \subseteq P^{\mathrm{t}} \wedge (\forall D^{'} \in P^{\mathrm{t}} \backslash \{L\}. |D^{'} \ominus L| \geq |D \ominus L|)$$

The RA property requires that for any key k and a peer D at time t, if $destVerification(k, D)$ return true then D is the closest peer on the right of k at time t [5]. The NA property requires that at time t for any peer L and D in the network, if $neighborVerification(L, D)$ returns true then L is the immediate left neighbor of D at time t. From these definitions the authors were able to show the next theorem that states that if the neighbor verification protocol is correct, then the system satisfies the RA property.

$$NA \implies RA$$

We said that authors in [5] used Trusted Computing in their architecture. What is Trusted Computing? Trusted Computing is a collection of initiatives to root security in hardware. The most noticeable manifestations are the *Trusted*

Platform Module (TPM), Intel's *Trusted eXecution Technology* (TXT) and *Virtualisation Technology* (VT-d). TPM is a hardware chip shipped in high-end laptops, desktops and servers made by all the major manufactures. TPM provides hardware-secured storage, secure platform integrity measurement and reporting, and authentication. In this P2P system, it is assumed that all peers have support for the trusted computing infrastructure and are equipped with TPMs. Also in this architecture a certificate authority (CA) is considered. CA is trusted to issue neighbor certificates as peers join and leave the network, and does not have to run on trusted hardware.

CSP has three denotational semantic models: *traces*, *stable failures* and *failures/divergences*. In this work, authors only used the traces model, especially the *refinement* relation on traces [5]. Let $traces(P)$ and $traces(Q)$ be set of traces of the process P and Q, then Q is said to refine P, written as $P \sqsubseteq_{\mathrm{T}} Q$.

The system model in CSP consists of several agents:

1. *Nonce Manager* - supplies fresh and unique nonces for other agents.
2. *TPM* - models the trusted hardware used in the system. Each TPM has a counter *cid* for P2P operations and is known to all peers.
3. *CA* - is trusted to issue neighbor certificates as peers join and leave the network.
4. *Verifier* - picks a random peer and asks it for its immediate right neighbor.
5. *Adversary* - models the attacker trying to break the RA property.

To prove that the system satisfies the RA property is equivalent to showing the following:

$$\mathit{Spec}(\{\}, P) \sqsubseteq_{\mathrm{T}} \mathit{System}$$

The current CSP model *System* is very large and complex. Even for a network with small number of peers, the model contains too many states and transitions to be checked automatically by a model checker. Authors approach for the verification was firstly to find an abstraction of the original model called *Abstraction*, whose state-space is smaller. The abstraction satisfies: $\mathit{Abstraction} \sqsubseteq_{\mathrm{T}} \mathit{System}$. Next they demonstrated that $\mathit{Spec}(\{\}, P) \sqsubseteq_{\mathrm{T}} \mathit{Abstraction}$, which then implies that $\mathit{Spec}(\{\}, P) \sqsubseteq_{\mathrm{T}} \mathit{System}$ is correct. All the proofs can be found in [5].

4 Proposed architecture

The objective of this work is to to take a P2P system, and provide formal guarantees of *liveness* and *safety* properties, and verify specific properties that we define.

The architecture of our P2P system is similar to *GINGER* [20]. The network is represented as a ring of peers connected to their neighbors. Every peer in the ring is a broker (super-node). Every broker has a group of peers that he must manage. The peers are grouped based on the type of the resources they are going to share (e.g books, movies, music, software, etc.) as we can see in the figure 4.

The peers have the same features as described in *GINGER*. We keep the currency and reputation for a peer, and only brokers are capable of introducing

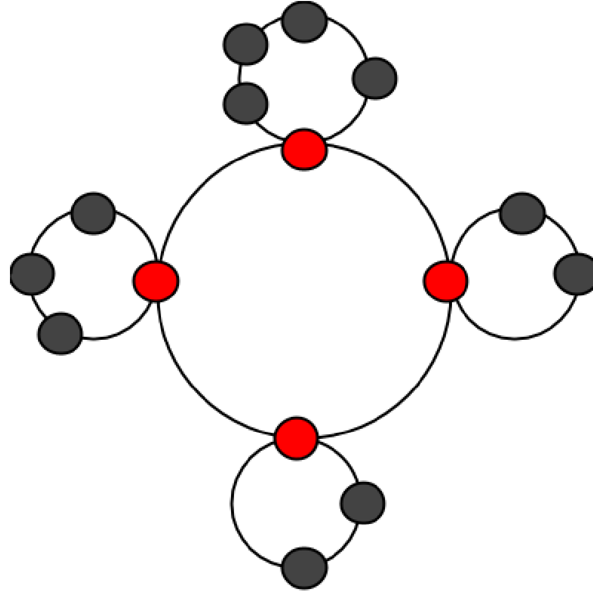

Fig. 4: Architecture of the system

new currency in the system. In a resource exchange, the buyer gives currency to the seller, and as a result the buyer becomes poorer while the seller wealthier. In our case, if the transaction was successful the broker gives positive reputation to both peers that participated in the exchange. To promote the activity in the system, the brokers will periodically adjust the reputation of a user downwards. The brokers are elected among the users with high activity and reputation. In case a broker fails, the system chooses another broker from the respective group among the ones with the highest activity and reputation. We assume that the brokers are intelligent users that won't risk their reputation to attack the system. In case we detect a broker as a malicious peer, we disconnect him from the network and ban him.

Every peer starts his activity in the network with a default currency and zero reputation. A new peer can join a specific group if he meets specific requirements, that we explain in the next paragraph, or he can join an initial group with peers that also have no start resources to share. In the case where new peers have no resources to share, the system will give them some resources so they can start their activity to raise their currency and reputation. When a user wants to buy something from the another peer, but has his uncertainties, he can ask the broker about the reputation of the other peer. To encourage users to report malicious users, a bonus is awarded in case the reclamation is verified.

4.1 Evaluation

In [21] authors propose a new peer group joining protocol. A peer that wants to be a member of one of the groups in the system, must have certain properties or meet the requirements related to the resources they provide to the other peers. For example to be a member of the software group, the peer must provide some open source software that he already has; to be a member of the movies group, the peer has to make available at least 10GB of movies. Also before joining a group a peer may want to verify conditions concerning the peers of this group to avoid joining malicious groups. For example imagine a peer p wants to join the group software. In this case the broker sends the requirements to p:

$$FreeSoftwareMembership \leftarrow @p(Software(Licenced$$

$$= False \wedge size = 1.5\,GB))$$

requiring p to share with the members of the group at least 1.5 GB of open source software. On his side, p is interested in joining the software group only if the peers in the group provide word processing, antivirus and threat removal tools [21], and sends to the broker his requirement:

$$@p(Software(Licenced = False \wedge size = 1.5\,GB \leftarrow \forall\, FreeSoftware.x$$

$$(@FreeSoftware.x(Software(Type = WordProcessor \wedge Type = Antivirus \wedge$$

$$Type = ThreatRemover)$$

The authors in [21] tested and evaluated their approach on an extension of the JXTA P2P platform. However no formal model of the system in order to define correctness proofs of the presented protocols were provided. They left it for the future work.

Our objective is to provide correctness proofs of this join protocol using TLA$^+$. Other properties will be considered and analysed in our future work. For example we will give high priority to the next properties and try to prove them:

- The resources delivered by one peer, will be received by another peer with no modifications and with no problems during the delivery.
- Ensure a downloading peer that the provided information is authentic and not poisoned.
- Once a resource got into the network, it will always be available and never dissapear.

In the end if we manage to prove these properties, and they satisfy our system, then we can say our work was successful.

5 Conclusions

We end our work by making an overview of what we have done. We started our work by studying the vulnerabilities of the P2P systems, and possible attacks

on them. Then we have surveyed the incentive systems, and saw how the P2P system can be improved, especially when we want to reduce the free riding in the network and reward the active users. This allowed us to build a necessary knowledge base for our future system. To define the properties that our system must satisfy, we had to study the verification frameworks and the properties they proved in another systems. We detached two frameworks CSP and TLA^+.

Regarding our proposed architecture we need to improve it a lot. The hard part of the architecture was to define the properties for the system. We want to define properties related with system resources that satisfy our system. However, all the related work we found about verification was related in most of cases with security and authentication properties.

The current architecture is based on *GINGER* [20] and the property we suggest to prove using TLA^+ is from [21]. In [21] the verification was done only by simulation and no formal guarantees were given. We presented additional properties that we would like that our system satisfy.

References

1. Adar, E., Huberman, B.A.: Free riding on gnutella. First Monday 5, 2000 (2000)
2. Buragohain, C., Agrawal, D., Suri, S.: A game theoretic framework for incentives in p2p systems. In: Proceedings of the 3rd International Conference on Peer-to-Peer Computing. pp. 48–. P2P '03, IEEE Computer Society, Washington, DC, USA (2003), `http://dl.acm.org/citation.cfm?id=942805.943825`
3. Christin, N., Weigend, A.S., Chuang, J.: Content availability, pollution and poisoning in file sharing peer-to-peer networks. In: Proceedings of the 6th ACM Conference on Electronic Commerce. pp. 68–77. EC '05, ACM, New York, NY, USA (2005), `http://doi.acm.org/10.1145/1064009.1064017`
4. Cohen, B.: Incentives build robustness in bittorrent (2003)
5. Dinh, T.T.A., Ryan, M.: Verifying security property of peer-to-peer systems using csp. In: Proceedings of the 15th European Conference on Research in Computer Security. pp. 319–339. ESORICS'10, Springer-Verlag, Berlin, Heidelberg (2010), `http://dl.acm.org/citation.cfm?id=1888881.1888907`
6. Douceur, J.R.: The sybil attack. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems. pp. 251–260. IPTPS '01, Springer-Verlag, London, UK, UK (2002), `http://dl.acm.org/citation.cfm?id=646334.687813`
7. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM 21(8), 666–677 (Aug 1978), `http://doi.acm.org/10.1145/359576.359585`
8. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th International Conference on World Wide Web. pp. 640–651. WWW '03, ACM, New York, NY, USA (2003), `http://doi.acm.org/10.1145/775152.775242`
9. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th International Conference on World Wide Web. pp. 640–651. WWW '03, ACM, New York, NY, USA (2003), `http://doi.acm.org/10.1145/775152.775242`
10. Karakaya, M., Körpeoglu, I., Özgür Ulusoy: A distributed and measurement-based framework against free riding in peer-to-peer networks. In: in Peer-to-Peer Networks", IEEE International Conference on Peer-to-Peer Computing. pp. 276–277 (2004)

11. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. pp. 654–663. STOC '97, ACM, New York, NY, USA (1997), http://doi.acm.org/10.1145/258533.258660
12. Lai, K., Feldman, M., Stoica, I., Chuang, J.: Incentives for cooperation in peer-to-peer networks (2003)
13. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
14. Locher, T., Schmid, S., Wattenhofer, R.: Rescuing tit-for-tat with source coding. In: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing. pp. 3–10. P2P '07, IEEE Computer Society, Washington, DC, USA (2007), http://dl.acm.org/citation.cfm?id=1306874.1307172
15. Lu, T., Merz, S., Weidenbach, C.: In: FMOODS/FORTE
16. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th International Conference on Supercomputing. pp. 84–95. ICS '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/514191.514206
17. Roscoe, A.W., Hoare, C.A.R., Bird, R.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
18. Rowstron, A., Druschel, P.: Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. SIGOPS Oper. Syst. Rev. 35(5), 188–201 (Oct 2001), http://doi.acm.org/10.1145/502059.502053
19. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg. pp. 329–350. Middleware '01, Springer-Verlag, London, UK, UK (2001), http://dl.acm.org/citation.cfm?id=646591.697650
20. da Silva Dias Rodrigues, P., da Cruz Ribeiro, C.N., Veiga, L.: Incentive mechanisms in peer-to-Peer networks. In: 15th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS), 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2010). IEEE Press (Apr 2010)
21. Squicciarini, A.C., Paci, F., Bertino, E., Trombetta, A., Braghin, S.: Group-based negotiations in p2p systems. IEEE Trans. Parallel Distrib. Syst. 21(10), 1473–1486 (2010), http://dblp.uni-trier.de/db/journals/tpds/tpds21.html#SquicciariniPBTB10
22. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput. Commun. Rev. 31(4), 149–160 (Aug 2001), http://doi.acm.org/10.1145/964723.383071
23. Veiga, L., Rodrigues, R., Ferreira, P.: Gigi: An ocean of gridlets on a "grid-for-the-masses". In: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid. pp. 783–788. CCGRID '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/CCGRID.2007.54
24. Xiong, L., Liu, L.: Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans. on Knowl. and Data Eng. 16(7), 843–857 (Jul 2004), http://dx.doi.org/10.1109/TKDE.2004.1318566

A Planning

Tasks	Duration
Improvement of current solution	February (1 week)
Introduction to TLA^+	February (2 weeks)
Formal model in TLA^+	March (4 weeks)
Verification of the properties	April (4 weeks)
Evaluation and conclusions	May (4 weeks)
Thesis final report writing	June - September
Review	September
Documentation	February - September
Bi-weekly meetings	February - September

Table 4: Planning schedule