UNIVERSITY OF TRENTO - Italy

Information Engineering
and Computer Science Department

Collusion in peer-to-peer systems

Renato Lo Cigno and Gianluca Ciccarelli
{locigno, ciccarelli}@disi.unitn.it

Technical Report DISI-09-048, Ver. 1.0

August 24, 2009

This page was intentionally left blank

Collusion in peer-to-peer systems

Renato Lo Cigno and Gianluca Ciccarelli
{locigno, ciccarelli}@disi.unitn.it

***Abstract*—Peer-to-peer systems have reached a widespread use, ranging from academic and industrial applications to home entertainment. The key advantage of this paradigm lies in its scalability and flexibility, consequences of the participants sharing their resources for the common welfare.**

Security in such systems is a desirable goal. For example, when mission-critical operations or bank transactions are involved, their effectiveness strongly depends on the perception that users have about the system dependability and trustworthiness. A major threat to the security of these systems is the phenomenon of collusion. Peers can be selfish colluders, when they try to fool the system to gain unfair advantages over other peers, or malicious, when their purpose is to subvert the system or disturb other users.

The problem, however, has received so far only a marginal attention by the research community. While several solutions exist to counter attacks in peer-to-peer systems, very few of them are meant to directly counter colluders and their attacks. Reputation, micro-payments, and concepts of game theory are currently used as the main means to obtain fairness in the usage of the resources.

Our goal is to provide an overview of the topic by examining the key issues involved. We measure the relevance of the problem in the current literature and the effectiveness of existing philosophies against it, to suggest fruitful directions in the further development of the field.

I. Introduction

Peer-to-peer applications have gained a large popularity over the years, becoming an alternative to other communication paradigms and providing services in a more efficient way, by using the network, storage and computation capabilities of the machines that participate as peers.

One factor that still strongly limits the use of this paradigm for mission-critical and money transactions is their perceived lack of security. More than in the other systems, in peer-to-peer networks the users can behave like in a real society, trying to gain advantages over the system or other users in a number of ways. Among the security threats to which a P2P system is subject, collusion has received but a partial attention by the scientific community. Collusion is the phenomenon created by intelligent agents[1] when they cooperate, not only with malicious goals, but also just to exploit at their advantage the flaws of the system. For example, let us consider a system based on incentives, like many existing applications do. In such systems, peers earn points by providing their resources, and can then spend those points to obtain resources. It is easy to imagine a group of peers that collaborate lying about each other's contribution: if only two peers claim that one has provided a service to the other, they can earn points and obtain a better service, that is, use other peers' resources, without actually providing theirs.

Scientific community has designed many systems that address the problem of security in general, and sometimes the collusion threat. Reputation and micro-payment systems have been developed and analyzed, either as research prototypes or as deployed on top of existing systems. Besides, since collusion, as stated before, is a social problem too, economics and social sciences come into play by offering models and theories to support the development of systems that can be proved to effectively fight this kind of threats. Among these tools, game theory and mechanism design have joined the classical solutions provided to ensure the integrity of data in distributed systems, creating a number of approaches to this problem.

A general classification of such partial solutions with a special attention to collusion, however, is currently missing in the literature. Our contribution is the tentative presentation of the problem of collusion in its generality, and the analysis of how well the current systems counter colluders. We provide an extensive comparison among a variety of approaches to gain insights in the problem and determine possible strategies for future development.

The paper follows this outline: in Section II we illustrate the problem in detail and motivate the need of an overview of the literature. In Section III we introduce the concept of incentives and the different ways which they are implemented in; later, we define the approach taken in micro-payment systems (Section IV as a viable alternative to incentives. In Section V we illustrate some approaches that extend known security mechanisms to limit misbehaviour by malicious and selfish collectives, without using incentives. We conclude the paper in Section VI summarizing the insights gained and prospecting future directions for study.

II. Collusion: motivation and flavours

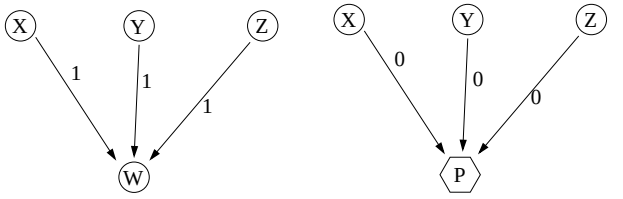
In any P2P system in which peers exchange services, we can coarsely distinguish between *honest* or *altruistic* agents, who behave according to the rules that the system enforces in order to ensure a global benefit, and *dishonest* agents, who do not follow those same rules. Collusion [2], [3], [4], [5] may be defined as the collaboration among two or more dishonest agents aimed at:

- subverting the system, e.g. by partitioning the overlay, through the diffusion of malicious executables (viruses), or the denial of service;
- unfairly gain advantage over/at the expense of the system or of the honest agents;

[1]Specifically, agents are the selfish and rational participants to the protocol (see Nisan and Ronen[1]).

- damage/isolate one (group of) honest peers, or
- any combinations of the above.

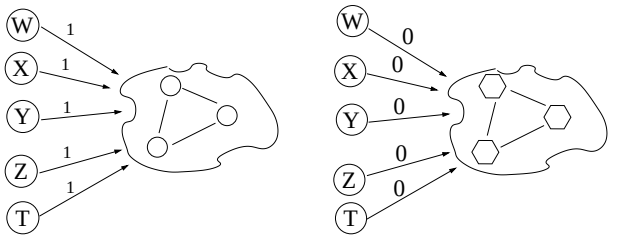
According to the objective of colluders: the collectives that try to create damage to the systems or to honest users can be classified as *malicious colluders*, while the groups that try to gain unfair advantage can be classified as *selfish colluders*.



(a) The colluders X, Y, and Z provide a (false) good reputation to W (promotion of W)

(b) The colluders X, Y, and Z misreport about the honest agent P

Fig. 1. Problem of misreport in reputation systems. The circles represent colluding malicious agents; the hexagons represent honest, collaborative agents. A report of 1 means the reporter received a good service; vice versa for 0



(a) The colluders promote a group of defecting peers

(b) The colluders misreport about a group of honest agents

Fig. 2. Problem of group misreporting in reputation systems.

Two approaches are possible against collusion: reputation systems and micro-payments systems (MPS).

Reputation systems [6], [7], [8], [9], [10] are meant to provide a measure of the reliability and honesty of an agent. The higher the reputation of a peer, the less probable he will be part of any malicious threat, including malicious collaboration. Exchanges build reputation: for each transaction between two agents p and q, where p requests a service and q provides it, agent p expresses an evaluation of the service received, that can be binary (good or bad) or take on any real value in a set, e.g., $[0, 1]$. The reputation of a target agent is computed by combining the evaluations of the agents who had a direct exchange with the target. Since the peers expressing the evaluation may be dishonest too, their reputation is used to weigh their evaluation about the target. The underlying assumption is that the history of the past interactions is shared among agents. Under this assumption, colluders can boost each other's reputation to easily obtain service from honest peers, as illustrated in Fig. 1 and 2.

Collusion as analyzed so far is only one form of the problem. Other types of misbehaviour in reputation systems are simplified as the *Traitor* and the *Mole* problems; while the traitor, however, is not a colluder, the mole is. In fact, a traitor behaves honestly and earns a good reputation for a first period, then starts negating the service. The motivation of a traitor is unclear; still, as no collaboration with other misbehaving agents occur, the traitor is not the object of our study. On the other hand, a mole or dormant colluder (also known as *front peer*) behaves honestly for a first period of time, then starts colluding according to one of the schemes previously discussed, and is therefore of interest to our survey.

Reputation systems oscillate around a trade-off. Given the different flavours of collusion, by aggregating information about trust values it is always possible to detect auto-referential circles of peers, but at the expense of a greater quantity of information needed. An alternative to (subjective) reputation is a central authority that controls the transactions and verifies the actual exchange of services, but this usually requires a greater control overhead.

Using this principle, the micro-payment systems (MPSs) [11], [12], [13], [14], based on the circulation of a form of money (e.g., coins or tokens), are an alternative to reputation systems. A peer can request a service if he can afford it: a transaction is defined as the provision of a service in exchange for money. Current solutions require a central authority, which we call *broker*, to produce the money and control its distribution. The solutions proposed deal with the problem of alleviating the load of the broker by moving it toward the involved peers. In fact, the overall computation performed in the system may become heavy because of the presence of cryptographic primitives (signature and public/private keys generation).

In a MPS, collusion is harder to achieve with respect to reputation systems. Colluders trying to disrupt the system can try to attack the broker, that represents a single point of failure, but existing solutions can be used to alleviate this kind of threat. On the other hand, a colluder interested in the service provided by the system cannot earn money by falsely claiming he provided a service to another colluder, because the transactions are strictly controlled, and the control entity can establish whether a money/service exchange occurred. Colluders can still try to forge money, but this strategy may be very expensive. Forging money is a powerful way to damage honest peers by buying services from them and paying them with false money; these payments, however, are easily traceable by the broker, and the source can easily be kicked out of the system. This implies that the reliability of such systems depends on how easy it is to create and manage new identities: if a peer excluded by the system can easily change identity, and come back in the system forging money[2], then the credibility of the broker may be damaged and honest peers may lose interest in the service provided. Nonetheless, if security is well-designed, forgery and change of identity are very hard to achieve, and MPSs can be considered tougher to attack for colluders.

[2]The phenomenon of frequent, cheap (if not free) change of identities is known as *whitewashing*.

Other than reputation and micro-payment systems, to counter collusion attacks we can use the instruments provided by different fields. In fact, collusion spans over both computer security and social-economic systems, and the most effective solutions currently available make use of tools from both areas. Interesting research approaches are given by game theory and mechanism design, on one hand, and by Public Key Infrastructure, DHT-based overlays, and bandwidth puzzles on the other.

III. Incentives systems

Let us examine now a first approach to making the user's decisions influence the utility of the system. Unlike anti-collusion by design, this approach makes use of incentives to give the peers some benefit in exchange of their resources. A good incentives system encourages cooperation from selfish peers.

A typical design method uses game theory[15], [1], [16]. By modeling the user's behaviour through game theory it is possible to enforce equilibria that will be followed because they are the most convenient choice for all the users.

We proceed as follows: Section III-A shows some attempts to incentivize peers towards cooperation, with a special attention to P2P streaming. In Section III-B, we discuss some incentives techniques based on game theory, subsequently analyzing a specialization of this approach known as mechanism design (Section III-C). In Section III-D we introduce reputation systems, specializing them in Section III-E, by introducing the trust mechanisms to build reputation.

A. Incentives system in P2P streaming applications

Peers in a streaming system can be incentivized to cooperate in order to reach the goal of social welfare. Incentives are given in return of some service provided, for example the provision of upload bandwidth. However, while file-sharing applications can benefit of simple mechanisms as tit-for-tat, results exist that prove this approach poorly effective. Chu, Chuang and Zhang [17] discuss the uniqueness of the case of P2P live streaming over the conventional file-sharing applications, in that commonly used tit-for-tat or bit-for-bit schemes are inefficient, because they strongly limit the benefit that peers can derive from the application. As a concrete example, let's consider the average peer which has an ADSL connection, and can thus benefit of a download bandwidth far larger than their upload bandwidth. With a bit-for-bit scheme, it will receive a stream at the speed determined by its upload bandwidth, that is a waste of the larger download bandwidth: this disincentivizes the peer's cooperation, making him leave the system or try to fool it.

The incentives system introduced by the authors, on the other hand, uses linear taxation to incentivize peers to cooperation. To implement it, an entity (the publisher of the content) is assumed to have the will and the means to enforce a payment by the users. The users in fact incur a cost by providing upload bandwidth, in order to receive a benefit, that can be expressed in terms of received bandwidth. Formally, each peer i has a forwarding capacity F_i and a receiving capacity R_i, while his actual provided and received bandwidth is denoted as f_i and r_i. The cost and the benefit are expressed as c_i and b_i, respectively. With this notation, users are supposed to maximize a utility function that can be expressed as:

$$u_i(r_i, f_i, F_i) = b_i(r_i) - c_i(f_i, F_i)$$
$$subject\ to: \begin{cases} r_i \leq R_i \\ f_i \leq F_i \end{cases}$$

Peers try to gain the maximum received bandwidth possible and to minimize the cost of providing upload bandwidth. The publisher, on his side, implements a taxation model in the form:

$$f = max(t \cdot (r - G), 0)$$

where t is the marginal tax rate decided by the publisher ($t \geq 1$), and G is the dynamically computed demogrant, that is, the bandwidth f_i that exceeds the received r_i and is redistributed evenly to peers. Indeed, the cost that each peer incurs into depends on the quantity of content he receives deprived of the bonus given by the demogrant factor.

To explain in which terms the received and forwarded bandwidth are adjusted, we briefly describe the streaming model used by the system. Peers form a multiple-tree structured overlay, and the stream is divided into stripes that are distributed along different trees formed by the peers themselves. If a peer subscribes to a tree, he receives the content. Multiple Description Coding is used, i.e., the content of the stripes is divided in such a way that if a peer receives more stripes, the quality of the received stream improves. With such scheme, assuming unit increases, joining r trees is equivalent to increase the received bandwidth by r, while the increase of the forwarding capacity is achieved through increasing the fan-out of a node. The publisher has to control the access to trees to enforce his tax policy. The authors do not focus on how the publisher can achieve this control, or which kind of punishment is used to deter peers from misbehaving.

To evaluate the system, the authors simulate the taxation mechanism using a distributed algorithm. Unfortunately, the results show that the overhead produced by the distributed algorithm used to compute the fair tax values for all peers is two times the normal overhead in which the system incurs by just managing the overlay structure. The comparison in terms of fairness proves the system to be effective, in that it strongly dominates the bit-for-bit strategy when peers are heterogeneous (in terms of bandwidth resources), while weakly dominates it when peers are homogeneous. An ideal taxation scheme is used for comparison, created by assuming optimality of the social welfare. The idea of a taxation scheme shows the inherent uniqueness of streaming over file-sharing applications, but the study does not address explicitly the problem of collusion.

Another similar approach is presented by Liu et al.[18]. The authors present a simple tit-for-tat scheme that favour the peers that directly contributed to the uploader. Unlike the solution by Chu, Chuang and Zhang[17], however, the coding scheme

is layer-based rather than MDC-based. This means that layers are ordered and have different importance, and that a higher-order layer can not be used in the decoding process if the lower-order layers have not been received yet.

The architecture in which layers are distributed is of type mesh-pull: peers request video chunks to their neighbours according to their needs. To upload chunks, the server p maintains a queue for each requesting neighbour. The incentive consists in giving priority to the peers that contributed most to p. On the other side of the communication, instead, the receiver q maintains a data structure, the buffer map, to keep track of the chunks requested but not received yet, available but not requested yet, and buffered but not played back yet. The peer q requests chunks according to three parameters:

1) the layer index: in a given instant of time, a lower-order layer is given higher priority than the higher-order layers
2) the playback deadline: the layers of closer deadline for the playback are given higher priority than the farther ones
3) the number of duplicates λ: the priority is given to the rarest chunk among those available at the neighbours

The receiver also selects the neighbour to which forward the request for each chunk. The selection is based on an estimation of the download time made by the receiver. Since the estimation might not be accurate, the receiver also sends probing requests, that are served after the regular requests in case any bandwidth is available at the server's side.

The authors present results obtained through simulation, in order to prove their system's resilience to free-riding. Simulation results prove that layered coding gives higher performance than MDC. Furthermore, free-riders are effectively discouraged because they receive a poor video quality. Although the problem of collusion, however, receives no attention, the results suggest an effective way of punishing misbehaviour which is general enough to be applied as deterrent against colluders.

Ngan, Wallach and Druschel [19] examine a research prototype based on incentives provided by a barter system in a totally decentralized way (no central authority). Peers use the system to manage storage quotas in such a way that each peer receives space remotely when he provides space to store others' files. The attack scenarios consider the presence of collusion and compare the performance of the system when colluders are present or totally absent. The authors assume that the colluders are a minority, and can even try to bribe other peers to let them use their space without offering space in exchange.

The design of the incentives system is based on the concept of random audits and altruistic punishment. A public key infrastructure is assumed to exist. Each peer p owns a *usage file*, digitally signed with his private key, containing the advertised capacity provided to store other peers' files, a local list with the files stored locally for others, and a remote list containing information about p's files stored remotely. The peer can store his files whenever he is *under quota*, that is, his advertised capacity is smaller than the storage he's consuming remotely in other peers' storage. If p wants to store a file F in q, q checks if p is under quota, then two entries are created:

1) an entry in p's remote list: F[3]
2) an entry in q's local list: (p, F)

The main threat against this system is peer x lying about his advertised capacity, claiming he can store locally more than he actually can, or lying about files stored remotely, claiming he stores remotely less files than he actually does. In particular, colluders can form a chain to hide an imbalance between offered and used quota by a misbehaving peer, that the authors identify as the *cheating anchor*.

Attacks are prevented by using random auditing on top of anonymized communication. If p is storing a file F for peer x, he can query x about his remote list. In an anonymous communication situation, since x cannot know who is auditing him, he cannot know which file he can lie about, by hiding the corresponding entry in the remote list. In fact, if x's remote list maliciously lacks the presence of F (x tries to increase his under-quota situation by claiming he is using less space than he actually uses), p can delete it from his local storage, because x is not paying for the storage anymore. The operation of auditing a node that a peer has a storing relationship with is called *normal auditing*.

To discover collusion and chains of cheating, in theory, it is necessary to walk the chain up to its originating cheating anchor, but this is computationally expensive if the system is composed of thousands of peers. For this reason, other than normal auditing, peers are required to perform a *random auditing* on a randomly chosen peer in the P2P system, not necessarily they are having a relationship with. The authors prove that with high probability all the peers in the system are subject to audit, including the cheating anchor. Since the usage file is digitally signed, the misbehaviour of the cheating peer is clearly certified by himself. Peer can ask the system to evict the misbehaving peer.

B. Game theory approach

Game theory offers powerful tools to model environments where autonomous agents, the players (peers, in our case), aim at the maximization of a utility function. The concept of equilibrium effectively describes selfish behaviours and the corresponding strategies, allowing the protocol designer to know when a player does have advantages in obeying the rules of the protocol, and when he does not. In this Section, we compare different approaches to the design of efficient peer-to-peer protocols, where peers have utility in following the protocol, highlighting the possibility of further study wherever the presence of colluders is not tackled explicitly.

Buragohain, Agrawal, and Suri [20] analyze a resource-sharing application and characterize the behaviour of its users with a game theory approach. They study the existence of equilibria that can allow the system designers to build a selfish-resilient application. Peers are assumed rational and strategic,

[3]The underlying P2P quota application makes it possible to find on which peer the file is remotely stored.

that is, they want to maximize their utility and can choose actions to influence the system. The strategy of a peer, in this setting, is the level of his contribution, that is, a peer can decide how much contribute.

D_i denotes the level of contribution for peer i. D can be anything meaningful in the application context: for example, in a file-sharing application, it can be the disk space shared; in a streaming system, it can be the dimension of the buffer map. A peer i incurs a unit cost c_i when he contributes a resource. If a peer contributes D_i, the total cost is $c_i D_i$, and we can express the contribution in a normalized form as follows: $d_i = \frac{D_i}{D_0}$. The designer wants peers to contribute at least d_i.

Peers get a benefit by joining the system, which should be so high as to compensate the cost they incur. A matrix $B = \{B_{ij}\}$ describes the benefit that peer i receives from peer j's contribution. To obtain the benefit, the peer i requests a resource to peer j, who provides it according to a probability distribution $p(d_i)$, that depends on the contribution provided by i. The authors choose the form $p(d) = \frac{d^\alpha}{1+d^\alpha}$, with $\alpha > 0$, but any monotonically increasing function can provide such a probability.

By combining costs and benefits, a peer i decides whether to join the system or not. The utility function describes the way costs and benefits are related: it has the form:

$$U_i = -c_i D_i + p(d_i) \sum_j B_{ij} D_j$$

or, in a normalized form, $u_i = \frac{U_i}{c_i D_0} = -d_i + p(d_i) \sum_j b_{ij} d_j$, where D_0 is a normalization constant. The first term is the joining cost, and the second the total expected benefit. As we see, this expected utility (for i) is the sum of the resources provided ($-d_i$) and the product of three elements:

1) what other peers contribute to the system (d_j),
2) how important their contributions are to i (b_{ij}),
3) and the probability that i will manage to get his requests satisfied ($p(d_i)$).

Given this model, the authors analyze the existence of Nash equilibria, that characterize the social environments where the optimal strategy chosen by a peer is stable, that is, no unilateral deviation from it can produce a higher benefit. First, a homogeneous setting is studied. In this scenario, each peer gains the same benefit from all the others, i.e., $b_{ij} = b$. The utility functions all have the form $u = -d + p(d)(N-1)bd$, so the game can be studied as a 2-player game:

$$u_1 = -d_1 + b_{12} d_2 p(d_1) u_2 = -d_2 + b_{21} d_1 p(d_2).$$

This setting is known as*Cournot duopoly*, which has a well-known Nash equilibrium described by:

$$d_1^* = \sqrt{b_{12} d_2^*} - 1 d_2^* = \sqrt{b_{21} d_1^*} - 1$$

assuming $\alpha = 1$ in the probability function of requests acceptance. The system can be solved as $d^* = (b/2 - 1) \pm \sqrt{(b/2-1)^2 - 1}$.

A solution to this equation exists if $b \geq 4$. We define $b_c = 4$ as a threshold under which the peer finds no advantage in joining the system[4]. For $b = b_c$, only one solution $d_1^* = d_2^* = 1$ $b > b_c$, then two solutions exist, an unstable one and a stable one. We can think about the first solution as the equilibrium that is reached when the users' contribution is so low that a new peer would not gain any advantage in joining the system, because his cost would overcome the benefit. The stable equilibrium is reached when a peer iteratively adjusts his strategy in response to the actions of other peers, until convergence. This equilibrium is stable because no peer would gain any more benefit by deviating from it (contributing less or more resources) after reaching it. For N players, by analogy the system to be solved to find the optimal strategy becomes: $d^* = \sqrt{b(N-1)d^*} - 1$.

The homogeneous case is not realistic, but is used as a model for comparison. Assuming heterogeneous peers, that is, a benefit matrix with generic b_{ij}, the set of equations to solve is $d_i^* = \sqrt{\sum_{j \neq i} b_{ij} d_j^*} - 1$, for each i. Solving such system is infeasible, thus the authors use an iterative learning algorithm to find a solution. In this algorithm, all peers start with random contributions; then, in each successive step, each peer adjusts his contribution according to the other peers' strategy. When the algorithm converges, a Nash equilibrium is found.

The results of the study are mainly meant to address the selfish free-riders, that in such equilibria would not have a reason not to contribute resources in exchange for what they receive. The security issues, however, are neglected. The peers are assumed to be trustworthy and not malicious, and thus to correctly report about their contribution level. The authors admit the need for an audit mechanism to verify the reports from the peers, but no actual implementation rule is described in detail.

Feldman *et al.* [21] propose a model of the peers' interactions based on the generalized prisoner's dilemma (PD[22]). One of the two players is identified as a client, who consumes a service offered by the second, a server, who offers it. The server can untraceably refuse to provide the service, i.e., the player who does not receive the service can not know who denied it. A basic assumption is the lack of any centralized trust or control. The study shows that the manipulation of the typical payoff matrix used in the formulation of the PD problem can be used to adapt the game to the P2P systems. The matrix must satisfy the following requirements:

1) The mutual cooperation leads to better payoff than mutual defection;
2) The mutual cooperation leads to higher payoff than the case when the two players behave differently from each other (one defects and one cooperates);
3) Defection dominates cooperation at the individual level, i.e., the single player gets a better payoff by behaving selfishly, even though the whole system does not.

The scenario in which these assumptions hold is also called *social dilemma* because the individual earns more by defecting than by cooperating, while the system as a whole has a

[4]For general α, $b_c = 4/\alpha$.

better global payoff when everybody cooperate. The players are assumed to be homogeneous, that is, they all have the same resources and can provide the same services, but their availability is not always guaranteed. The clients are assumed to always cooperate because they incur no costs whether they cooperate or defect.

The authors simulate different scenarios generated by modifying the behaviour of groups of peers and studying the effect of those modifications against the incentives mechanism proposed. The simulations are divided in rounds, in each of which every player requests and is requested a service in each round. At the end of a round, a player can behave in four ways:

1) mutate (switch to a randomly chosen strategy),
2) learn (switch to a more convenient strategy),
3) suffer turnover (log off of the system and be replaced by another peer),
4) keep her strategy.

A *traitor* is defined as a player who switches strategy, but keeps identity. The different percentage of nodes following these basic behaviours determine the differences in the scenarios.

To implement the learning process of a peer, any strategy is assigned a rating, given by the ratio between two moving averages:

$$\frac{MovAvg(s_i * age)}{MovAvg(age)}$$

where s_i is the i-th player's strategy's payoff, and age is the number of rounds he used that strategy for. The interpretation of the ratio is the following: it normalizes the payoff of a strategy through a time window (implemented by the moving average), then weighs it with respect to the previous history. The switch to another strategy occurs with a probability proportional to the difference between the currently used strategy's rating and the highest rated strategy's.

The main contribution of the paper is the introduction of a decision function named *Reciprocative*, according to which a peer decides whether to serve a request or not. A *decision function* takes a Boolean decision (cooperate or do not cooperate) according to a history. In particular, the Reciprocative function is based on a measure of the generosity of peers, defined as the ratio between what they provide and what they consume, normalized by dividing the server's generosity and the client's: then the probability to serve a request is proportional to the *normalized generosity* between the client and the server. The need for a normalized generosity arises as in some circumstances the authors found out that the Reciprocative peers may find themselves to have consumed more than they contributed: in this case, if the Reciprocative function used the generosity function, the decision whether to help the client or not would have been negative with high probability, so the reciprocative peers would become defective to each other.

The Reciprocative function is tested in a number of simulations, whose results show that by using a server selection mechanism and a shared history of the service, the system scales to large populations and is resilient to turnover and asymmetry of interests[5]. The *server selection mechanism* is based on the use of two fixed-length lists, one for the served and one for the server players. The client picks uniformly at random from one of the two lists whenever he has to choose a server to interact with, thus increasing the probability for a peer who previously received a service to reciprocate. However, this strategy alone does not scale well to large populations, and is ineffective against high turnover rates. The *shared history* mechanism makes all the peers aware of the previous behaviour of all the other peers, thus increasing the reliability of the reciprocative scheme; on the other hand, it is hard to implement in a completely distributed system (a task usually accomplished through DHT-based storage), and it creates the possibility for the peers to lie about the reported history, thus opening the possibility to collusion.

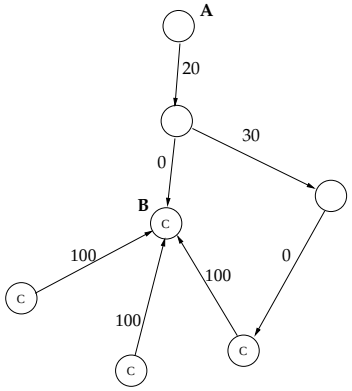


Fig. 3. Example of collusion and how the Maxflow algorithm fights it

To fight the collusion phenomenon, a maxflow algorithm[23], [24] is introduced that allows the Reciprocative function to compute a subjective reputation of a node. In contrast to an objective reputation, a subjective reputation does not use a global knowledge of how much peer agree on the trust of each other peer in the system: in particular, the *maxflow algorithm* operates on a portion of a digraph to give a value to the paths that may exist between the client and the server. The digraph is built in the following way (see Fig. 3): the vertices are the peers, the edges represent the service a client requested to a server, while the weights are the reports about the service that the source (client) received from the sink (the server). The maxflow algorithm computes the path between the node calculating the reputation and the node under examination. In Fig. 3 the node A finds a flow of capacity 0 towards the node B because, even if the colluding nodes (indicated with a C letter) report that B helped them, the node whom A trusts has never dealt with B, thus it is correctly identified as not trustworthy. The subjective

[5]The asymmetry of interests occurs when a server player who is requested of a service does not know that the requesting client has served other peers than her, thus effectively collaborating inside the system but not directly with the server herself.

reputation of the player B perceived by player A is:

$$min\left\{\frac{maxflow(j,i)}{maxflow(i,j)},1\right\}$$

The drawback of Maxflow is its cost in terms of running time, which is $O(V^3)$; however, the authors use a truncated version of the algorithm which presents good properties of scalability, though in some cases no flow is found even if it exists.

Type of misreport	Effect	Comments
No service received	Lower reputation of the server	Expensive (the liar must provide service before lying effectively)
Service received	Increase reputation of the server (Collusion)	Expensive
Service provided	Increase own reputation	Minimized by the structure of Maxflow (the closest entity is trusted more)
Service not provided	No benefit	Ineffective

TABLE I
SYNOPSIS OF THE TYPES OF FALSE REPORTS.

Problem	Solution	Comments
Reputation	*Reciprocative* decision function	Shared history opens the possibility to collusion
Collusion	Maxflow	accuracy vs. running time
Mole	-	Expensive strategy for the mole, thus not used for long
False reports	Trust the closest peer more	Expensive strategy
Whitewashing	Adaptive stranger policy	Trade-off (Being stingy with all newcomers discourages new contributors); ineffective for high turnover rates
Traitor	Short-term history	Trade-off (Long history creates more stable reputation but proves ineffective)

TABLE II
PROBLEMS ADDRESSED IN FELDMAN *et al.* [21]

The paper then addresses the mole problem, false reports, zero-cost identities and traitors. The system can not really address the problem of a *mole*, i.e., a peer who earns a good reputation and then starts colluding; however, the authors show that the mole's strategy is expensive because it takes upon itself the cost of providing the service, which turns out to be an ineffective strategy and is rapidly abandoned. An analogous solution is discussed about false reports of a peer about any other peer (see Tab.I): in particular, it is demonstrated that the false reports are expensive to produce because require a good reputation, to obtain which it is necessary to invest in providing services. The problem of whitewashing caused by the zero-cost identity mechanisms is addressed by an adaptive strategy, with positive results over the fixed strategies, until the turnover is under the rate of 10^-3. Finally, the problem of the traitors is addressed. A *traitor* is a peer who earns a good reputation and then start defecting (i.e., negating service) to other nodes. It is shown that traitors are disruptive for the system whenever a long-term history is kept, while by using a shorter-term history the effects of good reputation are easily removed without destabilizing the system. A resume of the main results of this paper is shown in Tab. II.

Morselli, Katz and Bhattacharjee [25] propose a game-theoretic framework to analyze the robustness of a trust-inference protocol, that is, where the interaction between peers is directed by the trust they have in each other. Without focusing on a single protocol, they give a notion of robustness in a strong adversarial model, in the absence of a central authority, that can be used to analyze a protocol and to compare different incentives systems[6].

Let Π be the protocol that honest peers obey to, and malicious peers do not. The protocol describes the way trust is computed and assigned to each peer, and the actions that peer perform according to the trust values. The adversary, A, knows the details of the protocol, can see every message exchanged between any two peers, and can interfere with the protocol by sending messages.

The peers are identified through pseudonyms. The pseudonyms are distinct, easy to create, but cannot be used by an adversary to impersonate an honest peer. The adversary can use four oracles, that are functions that simulate his power of manipulating the interactions and provide a worst-case for the analysis:

1) `NewUser`: Create an honest user, whom A knows of. Through this oracle A controls the size of the network;
2) `HonestPlay(i,j)`: Make i and j (honest peers) play a game. Through this oracle, A controls the trading partner of the honest peer;
3) `Play(i, id, action)`: Play a game between A (identified by id) and i. A plays $action$, while i plays according to Π;
4) `Send(i, id, message)`: Send a message $message$ from A's id to user i.

The interactions between peers are described through a payoff matrix typical of a classical Prisoner's Dilemma[7], which means that the best strategy (highest payoff) for each player is to defect instead of cooperating. The authors, however, do not explicitly limit the choice of the game.

The robustness of the protocol helps understanding whether following the protocol is the best strategy for the adversary. To model robustness, A is assumed to have the capability to perform a limited number of contemporary actions: in particular, he stores an integer timestamp $t \geq 0$ for each action he performs, and the model assumes a limit to the number

[6]An important consideration is that incentives are meant to encourage selfish peers, whereas the malicious ones have to be countered by other means, more related to security.

[7]See AppendixA.

of new user created at the same time by using `NewUser`, and a limit to the number of simultaneous interactions created by invoking `Play`. The protocol is run normally up to the time when the adversary decides to perform an action, and A can engage more users without giving them the chance to interact (according to the protocol) in the meanwhile. In other words, we can suppose that Π is followed by honest peers, and then, at time t, everything stops, and A interacts with more peers at the same time. After the simultaneous interaction, the honest peers can keep on interacting normally (until the next A's action). Now, each time he plays the `Play` oracle, the adversary increases his utility of an amount $\delta^t \mu$, where $\delta < 1$ is a *discount factor* and μ is the payoff given by the payoff matrix defined above. The discount factor models the decrease in utility as a result of playing later in time. A's goal is to maximize his utility. For this reason, the protocol Π is *robust* if A maximizes his utility by obeying to it.

In this framework, a protocol can satisfy additional desired properties. The *expected utility*, that is, the utility everyone receives when everyone obeys the protocol, should be maximized. A protocol is moreover required to be *efficient* in the usage of network resources (for instance, by limiting the number of overhead messages), and specifically efficient in the policy it uses with newcomers. Finally, it is desirable that the protocol be resistant to faults, in particular that it keeps his robustness when faults occur.

With these properties in mind, a designer can prove his protocol to satisfy formal requirements and compare it with existing protocols. The authors, however, do not provide an example of a protocol which is robust and at the same time admits the presence of colluders, even if the framework allows modeling malicious/selfish coalitions.

Keidar, Melamed and Orda [26] design a peer-to-peer system that counter freeloaders by modeling the adversarial environment as a non-cooperative game for which an equilibrium is devised that ensures all the participants will receive all the content (content distribution) or the content produced from their arrival into the system onwards (streaming).

A single server S distributes a number P of data packets to N peers, p packets per unit of time. A node is assumed to have sufficient upload bandwidth to send p/k (plus a small constant) packets per unit of time to each of his k neighbours. The N peers are assumed to be non-cooperative. Among the strategies they can decide to follow, the authors describe the properties of the *Protocol-Obedient Strategies* (*POS*s), which are the strategies that follow the protocol and define the number of connections and the traffic flowing in them. Users endeavour to maximize their utility, by using strongly dominant strategies, that is, strategies that incur the minimum cost for any choice of other peers' strategies. The cost function user i tries to minimize has the form:

$$c_i(s_1, s_2, \ldots, s_N) = \begin{cases} \infty & \text{if } in_i < P \\ out_i & \text{if } in_i = P \end{cases}$$

where in_i, out_i are the number of packets received and sent, respectively, and s_i is the strategy of peer i. This function is slightly different in case of streaming applications. In fact, a peer can join a stream in every moment, but he is not interested on packets appeared before his join. In this case, P is substituted with mp, where m is the number of rounds the peer has sojourned in the system, and p is the multicast rate from the source.

The central (logical) server S builds an overlay that connects the peers. The overlay is required to satisfy strong constraints to bound the delay of messages and to grant the existence of an equilibrium of the strategies: in particular, all nodes have to have exactly k neighbours. The diffusion of packets is based on gossip: every round, each node exchanges information with its neighbour about the packets received, and provides the packets requested. Each node keeps a balance between himself and each of his neighbours between what it provides and what the neighbour provides him in terms of packets. The balance should never fall below a threshold[8]. If a peer does not own a sufficient number of packets, he asks the source to provide packets on its behalf, and pays a fee in terms of *fine* packets, i.e., dummy packets which do not contribute to their balance, but waste their resource (and the network's). In these cases, the source S is said to behave as an *emulated node*. It is clear that it is not in the interest of the peer to ask the help of the source; on the other hand, the frequency of such situation may limit the scalability of the system for the load the source incurs in.

In each round, a peer is requested to send and receive packets. A peer who wants to minimize his cost decides to pay, in terms of fine packets, rather than fall below the threshold and misbehave. In fact, if a peer does not receive gossip messages, nor requests for packets from one of his neighbour, he assumes the neighbour is misbehaving, and he cuts the overlay link toward him and asks the server to replace it. The punishment for misbehaviour, thus, is the irreversible eviction from the system. It is not completely clear, however, whether the peer can reconnect by using another identity. Since no reputation mechanism is used, a user evicted from the system can just rejoin with another identity. This strategy, however, is ineffective in streaming applications, where being evicted from the system, even for a few seconds, substantially degrades the quality of the user experience.

The authors prove that if all the nodes choose strongly dominant strategies in the set of POSs, then

1) All nodes have an initial set of k neighbours
2) All nodes keep their neighbours until the end of the multicast session
3) All nodes receive all the packets

In particular, no unilateral deviation from such dominant strategy gives the peer a greater utility (lower cost), so this is proved to be an equilibrium. This theorem, however, is proved for the static setting, where no joins or leaves occur. The basic assumptions are that most users use a POS because they do not have the technical skill to hack the application, or

[8]The threshold is conceptually similar to the imbalance ratio described in Li *et al.* [27].

refrain from installing hacked applications. It is also assumed that no out-of-channel communication occurs among users: the results found by the authors depend on the substantial isolation of users. For this reason, the system cannot be proved to be collusion-resistant (nor was collusion-resilience a design objective). In particular, no malicious peer is assumed.

Eventually, the authors describe a dynamic setting with join and leave operations.

Li et al.[27] design a system that encourages collaboration by delaying the benefit until reciprocation is fulfilled (long service maturation). The authors implement a video streaming application, that enforces an *approximated* Nash equilibrium, described by a parameter ε. While obtaining a perfect Nash equilibrium may be very hard or even impossible, due to computational complexity problems related to the distributed nature of the system and to the interactions among its users, an approximated Nash equilibrium is computationally acceptable, and can still provide the desired quality of service, despite the presence of both rational and Byzantine players.

The system involves three types of entity. First, a source S generates P new chunks in each round, and distributes them to a small fraction of peers. Each chunk has an expiration date, after which it becomes useless (adds no utility to the receiving peer). Second, a tracker keeps track of the participants by storing their identities (public keys): each user interested in the stream asks to join to the tracker. Third, the peers participate to the stream in order to get the benefit of watching the stream. Each peer delivers P chunks in a round; if this does not happen, then a loss occurred and the video quality degraded. The goal of the system is to minimize the number of rounds where the quality degrades, or, to keep high and constant the continuity index as defined in [28].

The authors model the adversary according to the BAR assumption, where the acronym stands as a reminder of the three kinds of interacting players in the network, i.e., they assume the presence and interaction of altruistic (A), Byzantine (B), and rational players (R). The latter pursue the maximization of a utility function that can be expressed as $u = (1 - j)\beta - w\kappa$ where j is the average number of discontinuities, β is the benefit obtained when a continuous stream is received, w is the used upload bandwidth (in kbps), and κ is the cost of 1 kbps upload bandwidth. We can interpret the utility as the number of continuous rounds (without losses) minus the cost in terms of upload bandwidth.In such setting, a ε-equilibrium is a steady state where peers deviate if they expect a utility increase of a factor ε.

To reach the design goal, the authors design a basic gossip-based protocol to ensure robustness against up to a fraction of Byzantine players, and then augment it with a number of improvements to increase the performances. The basic protocol comprises four phases. Let's suppose that the source sends both chunks and a linear digest that can be used to prove the chunks authenticity in each round. In each round, the peers follow these four phases:

1) Select a partner using a verifiable pseudo-random algorithm (to avoid malicious selection, and at the same time take advantage of the randomness to implement robustness against faulty nodes);
2) Exchange buffer maps (histories) to decide exactly which chunks exchange according to what a peer owns and what the partner owns;
3) Exchange encrypted chunks (*briefcases*), plus a signed *promise* message that matches the briefcase content. The promise may be used as a Proof of Misbehaviour if the briefcase contains garbage. A peer accused with a POM is evicted from the system.
4) Eventually, upon checking briefcase and promise and finding them matching, the peer sends the decryption key to the partner.

The basic protocol is robust against the presence of faulty nodes, but suffers from unacceptable penalty in performance. To address this issue, the authors plug into the basic protocol a set of improvements. First of all, peers reserve trades one round before performing them, in order to organize the connections with partners and limit a probable excess of requests. Nodes are instructed to refrain from asking for the same missing chunks to more than one partner, thus optimizing the network load; an appropriate chunk selection mechanism and the usage of erasure codes techniques ensure the minimization of losses. A further improvement includes a tolerance in the imbalance between the quantity of data uploaded and downloaded, in order to give disadvantaged peers the possibility of obtain the stream even though uploading slightly less chunks. A detection module monitors the exchange of chunks and suggests a node to initiate more trades in a round to catch up with missing chunks and recover quality of service. Eventually, a further improved partner selection algorithm deals with newcomers and alleviates the load on older participants by bounding the possible partner choices of newcomers to a smaller subset of the entire participant set.

The system gains in performance, but, what is more important, it can be proved to enforce an ε-Nash equilibrium. Let's see now how the authors analyze the equilibrium and how the analysis affects the design of the protocol. The starting point is the consideration that the equilibrium enforced by the system is not valid in every round, that is, a peer may gain more by deviating. Let's consider the optimal strategy and the utility that comes out of it, that can be expressed as u_o. We can describe the relative advantage of the optimal cheating strategy over the strategy the obeys the protocol (that we can call u_e) as follows:

$$\varepsilon = \frac{u_o - u_e}{u_e} = \frac{(j_e - j_o)\beta - (w_o - w_e)\kappa}{(1-j_e)\beta - w_e\kappa} = \frac{\frac{cj_e}{1-j_e} + (1-b)}{c-1}$$

where b is a fraction ($b < 1$) of the bandwidth used to run the protocol (that can be lower-bounded). In steady state, the user has to upload at least $min_ups = \lceil \frac{needed}{1+a} \rceil$, with a corresponding cost of $cost = \gamma + min_ups \times \rho$. The parameter γ represents the fixed cost of a trade in kbps, while ρ is the increase in cost for each chunk uploaded; finally, $needed$ is

the number of chunks a peer needs in each round. To find the equilibrium, we can solve for c with the objective $\varepsilon = 0.1$: we find that this is an equilibrium if the rational peer values the stream at least 3.36 times more than his cost in bits.

The system is proved to be robust against 10% of Byzantine peers, and resilient against selfish behaviour. Long term strategies performed by malicious colluding peers, however, are not considered explicitly and their study is put off as a future work.

Kash, Friedman, and Halpern [29] offer an alternative vision of collusion and Sybil attacks, where the formation of coalitions, be them formed by rational users or by mock users (as is the case for Sybil agents), can have a positive effect on the system's overall welfare.

T	set of types
f_t	fraction of nodes with type t
m	average money per agent

TABLE III
PARAMETERS THAT DESCRIBE THE MODEL USED BY KASH *et al.* [29]

The system is modeled with n agents, each of which can ask for a service and provide it, if he has enough capabilities. The system is described as a tuple (T, f, n, m), whose components are explained in Tab. III. A standard agent is described through a tuple $t = (\alpha_t, \beta_t, \gamma_t, \delta_t, \rho_t, \chi_t)$ (see Tab. IV). The prices for services are assumed to be fixed and normalized to 1\$, that is, a service provided by agent i to agent j costs 1\$ to j and makes i earn the same amount of money. An important assumption is that the random variables that model the behaviour of agents in coalitions are correlated, thus the authors cannot use the concept of entropy to measure the difference among them, but they have to use the concept of *relative* entropy, that measures the weighted distance among the distributions. The agents value the exchange of services with the same valuation γ.

The agent's strategy consists in the simple choice of a threshold k, below which he starts volunteering to earn money. It can be interpreted as the greed of the agent: if k is high, the peer has higher incentive to start volunteering.

	Definition	Interpretation
α_t	Cost of satisfying the request	Amount of resources provided
β_t	Pr{Agent can satisfy the request}	The Agent has the same type of the request and the required resources
γ_t	Utility gained from the agent when his request is satisfied	Benefit received without considering costs
δ_t	Discount rate	Peer's patience: The more patience, the more the utility got in the next round will be similar (in amount) to the same utility received in this round
ρ_t	Service request rate	A peer's need for service
χ_t	Pr{being chosen to give service | the agent is capable of satisfying the request}	If more peers volunteer to offer the service, they are chosen according to this probability distribution

TABLE IV
MODEL OF THE AGENTS IN KASH *et al.* [29]

Given this model, the authors analyze two types of scenario, Sybil attacks and Collusion, finding out that there is an overlapping between them in some cases. A sybilling agent, i.e., an agent that controls Sybils, can be effectively discouraged from Sybil behaviour by making him pay a cost for each identity he creates. The authors observe that the Sybil threat is more effective on the system's utility when the number of sybilling agents increase, thus arguing that the designers have to find a method to stop the phenomenon in the early stages of the system's operations. The conclusion is somehow surprising, because there are situations where the presence of Sybil agents is beneficial to the system. This can be explained by considering that the social welfare almost coincides with the number of requests satisfied. The sybilling agents have a high chance to satisfy requests, thus they tend to have lower k (they earn more, so they are less greedy, too), thus leaving place to the competition among fewer honest nodes. In particular, if we call p_e the probability of earning 1\$ when providing a service, and p_s the probability of spending 1\$ as there is an agent willing to satisfy this agent's request, then sybilling provides a net gain to the attacker when $p_e < p_s$, and a loss otherwise: the longer the distance between the two quantities, the more the gain or loss, respectively. This can be explained by considering that with a cost for the creation of identities, creating an identity is profitable if the need for earning (p_e) is far less than the spending capability (p_s). With these premises, the authors prove a theorem according to which, by carefully manipulating the quantity of money present in the system m, Sybil attacks can be effectively countered, keeping p_e high enough for honest peers to have advantage in staying in the system.

The simulation results show that when the value of m goes over a threshold, the system crashes. Without Sybils, m can be increased up to a value of 10.5 before the system crashes; when Sybils are present, this value decreases ($m = 9.5$). This means that setting m without considering the presence of Sybils can take the system to crash, clearly unveiling the existence of a trade-off between efficiency (higher m) and stability (less m).

Collusion, as said above, can overlap with the Sybil attack whenever colluders pass off a request to another member of the coalition. The authors model the collusion as the presence of coalitions inside which money and resources are shared, thus assuming away the presence of more types for the agents. To separate the study of collusion from the problem of Sybil agents, they also assume that a colluder accepts a request when he can directly satisfy it. The advantage in colluding is that when a peer does not have money to request a service (and only in this case), the members of the coalition volunteer. Moreover, all the request that can be satisfied inside the coalition are indeed satisfied by another colluder. As the size of the coalition grows up, there is less and less need to send requests outside it. This situation has clear advantages for the members of the coalition; it is clear, however, that other non-colluding agents with low p_e have less chances to earn money

because most requests from the colluders will be satisfied inside to the group. The authors obtain results similar to the case of Sybil agents, showing that the system benefits when the number of colluders is in a certain range, but that when the second extreme of the interval is overcome, the system becomes unstable until it eventually crashes.

The collusion can also be encouraged to provide a loan service: a member of the coalition without money can ask to borrow some from another member. This mechanism, however, is particularly sensitive to whitewashing, where agents change identity to escape debts, and its design represents an open question for research.

C. Mechanism Design

Mechanism design (along with Algorithmic Mechanism Design, or AMD, and Distributed AMD, or DAMD) [1], [30], [31] has a strict dependence on game theory, which has been used as a modeling tool for the construction of incentive systems. The characteristics of systems modeled using mechanism design is that they rely on the system inner structure to discourage and combat collusion, that is, the system directly enforces fair behaviours.

Mechanism design (MD) is the discipline that tries to induce a behaviour onto selfish agents by designing the payments and the punishments for good and bad behaviour, respectively. Formally, we have n agents having a *type* each, denoted as $t^i \in T^i$ for agent i. The type is a privately known input, while the other information is publicly known. A mechanism design problem is composed of an output specification, that maps $t = t^1...t^n \mapsto o \in O$, where O is the set of allowed outputs, and a set of utility functions for the agents. According to his type t^i, each agent gives a value, called *valuation*, to any output, in the form $v^i(t^i, o)$. The utility can be expressed as the sum of the number of currency units assigned by the mechanism to the agent (p^i), plus his valuation of the output:

$$u^i = p^i + v^i(t^i, o)$$

This last form of the utility function is known as quasi-linear.

Mechanism design is an economic theory, but it can be combined with tools from the computer science, like approximation, and a particular concern about computational tractability. An overview of Algorithmic Mechanism Design with an example application is provided by Nisan and Ronen[1]. The authors describe MD optimization theory, and then introduce the Vickrey-Groves-Clarke mechanisms that prove useful to solve many algorithm mechanism design problems. At this point, they focus on a specific case study, i.e., a task scheduling problem, introducing the concept of randomized mechanism, and the addition of a *verification* phase to the mechanism, that can be used to punish misbehaviour.

According to the previous outline, let's define what an optimization problem is in the context of MD. A MD optimization problem can be defined by two components:

1) An output specification, given by a function $g(o, t)$ positive and with real values and a set of feasible (computable) outputs, F;
2) An output requirement, $o \in F$ that minimizes g, or an approximation.

In the context of an optimization problem, we can describe the concept of *mechanism*. A mechanism $m = (o, p)$ must assure the required output when agents behave selfishly. It defines payments for each agent, depending on the strategy they used. Particularly, the mechanism defines the possible actions that each agent i can perform as a set A^i from which agent i chooses his strategy a^i. An output function has the form: $o(a^1 \dots a^n)$, that is, it depends on the choices of agents. The payments are assigned according to the chosen strategies, in the form of a vector $p^i(a^1 \dots a^n), \forall i$. As conventional in game theory, we define the vector a^{-i} as the vector a without the component related to agent i. With this notation, a mechanism can be designed with *dominant strategies* if: (1) $\forall i, t^i$ it exists $a^i \in A^i$ (dominant) such that $\forall a^{-i}, a^i$ the mechanism maximizes i's utility, and (2) $\forall a = (a^1 \dots a^n)$, the output $o(a)$ satisfies the specification. A mechanism is poly-time computable when the outputs and the payments are computable in polynomial time.

The optimization problems, as they are formulated above, create a wide spectrum of cases. To further restrict the analysis, the authors use the revelation principle: If there exists a mechanism that implements a problem with dominant strategies, then there exists a truthful implementation as well. If the revelation principle holds, we can concentrate the analysis on truthful implementations.

A particularly useful class of truthful implementation is known as Vickrey-Groves-Clarke (VGC). VGC can be applied to maximization problems where the objective function is given by the sum of the valuations from all the agents. The problem, however, is that VGC mechanisms are often intractable, through optimal. This requires the use of approximated solutions.

To show an example of the approximated approach, the authors describe the classical task allocation problem and find an approximated algorithm for a special sub-case by using AMD techniques. The task allocation problem can be described in terms of MD as follows: We want to assign k tasks to n agents. The type of each agent i is t^i_j, the minimum time that i needs to perform task j. The feasible outputs are the partitions $x = (x^1 \dots x^n)$, where x^i is the set of tasks assigned to agent i, and the objective function is $g(x, t) = max_i \sum_{j \in x^i} t^i_j$. Each agent gives a valuation $v^i(x, t^i) = -\sum_{j \in x^i} t^i_j$, i.e., the opposite of the total time spent to execute the tasks in the set x^i. First, the authors find an upper bound by defining a mechanism where each task is assigned to the agent that can execute it in the least time, and for each task j he is paid in terms of the time that the second fastest agent would have used to execute the same task. The lower bound is a 2-approximation, that is, it performs twice worse than the optimal algorithm.

The theoretical bounds provide two limits to how well a mechanism can perform. The authors, however, prove that a randomized version of a mechanism can perform better than the lower bound described above. Before describing

how, let's see how a randomized mechanism is defined. A *randomized mechanism* is a probability distribution over a family of mechanisms $\{m_r | r \in I\}$ that share the same set of strategies and possible outputs. The mechanism's outcome is a probability distribution over outputs and payments, while the objective function is the expectation computed over all possible objective functions of each mechanism. Now, we can outline the procedure followed in the study to prove the effectiveness of the randomized mechanisms. First, a mechanism is defined, which depends on a parameter β and a bit vector $s \in \{1,2\}^k$. At his point, the random version assigns a fixed value to β , but defines a uniform distribution for the bit vector. It can eventually be proved that the random mechanism so defined is a 7/4-approximation for the task scheduling problem when two agents are in the system.

As we can see, to obtain closed forms for the results we have to narrow down our perspective to very special cases. Another path, however, allows the authors to define an interesting technique to counter malicious peers. Namely, in the task allocation problem, we can distinguish two stages in the strategies of the agents: in a first stage, they declare the times they need to execute the tasks, while in a second stage they actually execute, in a time generally different from what they originally declared, the tasks the mechanisms assigned them. The mechanism, however, assigns the payments *after* the actual execution, according to the difference between what was declared and the actual execution times. The consequent mechanism, called Compensation-and-Bonus, is such that lying agents are punished for their misbehaviour. The mechanism is intractable because it requires exponential time algorithms for the computation of the payments and outputs; it is possible, however, to approximate it with a polynomial algorithm, but losing the truthfulness property. In particular, a sub-case of the task scheduling problem can be approximated with a $1+\varepsilon$ polynomial approximation $\forall \varepsilon > 0$.

Ma *et al.*[16] design a framework that provides service differentiation according to users' contribution and incents peers to contribute resources to the system, with an interesting formal result about the amount of collusion the application can tolerate. The framework consists in a resource allocation mechanism, the Resource Bidding Mechanism with Incentive and Utility (RBM-IU), that induces a competition game among nodes requesting a service to a peer. The competition is modeled through a competition game played by following a network protocol. Peer i expresses his requests through biddings ($b_i(t)$) sent to the source of a service s. The source has a given amount W_s of bandwidth to distribute to requesting peers, according to their biddings and their contribution. The game is proved to have a unique Nash equilibrium, practically implementable using a linear time algorithm by perturbing the theoretical solution by a small positive amount ε.

The RBM-IU mechanism is built upon a max-min fairness model for bandwidth allocation and encourages contribution by assigning more resources to more contributing peers. The theoretical problem of maximization the source solves for the assignment is:

$$max \sum_{i=1}^{N} C_i log\left(\frac{x_i}{b_i}+1\right) \quad s.t. \sum_{i=1}^{N} x_i \leq W_s$$

with $x_i \in [0, b_i] \, \forall i$. This allocation procedure does not waste resources and is proved to be Pareto-optimal, that means that the allocation vector cannot be improved further without reducing the utility of at least one node.

The allocation mechanism generates a competition that can be modeled using game theory. In particular, the parameters W_s and C_i are supposedly common knowledge of every player (the peer). The strategies are defined by the bidding values b_i for every i, and $\vec{b} = \{b_1, b_2, \cdot, b_N\}$ describes a strategy profile. The allocation mechanism can be imagined as a system that takes as input the contributions $\vec{C}$ and the biddings $\vec{b}$, and produces the assignments $\vec{x}$, outcome of the game (the players want to maximize the allocation x_i)[9].

After describing the game structure, the authors prove two results, and then describe how the theoretical analysis can be translated in a practical implementation. Specifically, the strategy

$$b_i^* = \frac{W_s C_i}{\sum_{j=1}^{N} C_j}$$

for each player i is a (unique) Nash equilibrium. Given the validity of this result, it is possible to prove an interesting property of collusion resistance of the system. Let's start from a definition: κ-collusion occurs when a subset of κ competing nodes N_κ use a strategy profile $b_i \neq b_i^*$ and achieve a larger allocation of bandwidth, in the following form:

$$\sum_{i \in N_\kappa} x_i > \sum_{i \in N_\kappa} x_i^*.$$

The authors prove that κ-collusion does not create a better allocation than the Nash equilibrium, thus peers have no interest in deviating from the solution that ensures the maximum social welfare.

In order to translate these nice properties into an existing and feasible system, the authors make some modifications to the theoretical model, proving the persistence of the properties discussed above. The first practical issue is the common knowledge of contribution of all peers by all peers. This problem is solved by making the source send a signal at the beginning of the competition game stating the value of the allocation in terms of user contribution. The source has the interest to apply this procedure because it enforces the Pareto-optimality and thus a fair allocation that the source itself takes advantage of. To take into account the possible resource waste due to network congestion, the authors modify the bidding system with a perturbation factor. In particular, nodes use a bidding value given by $b_i = min\{w_i, (1+\varepsilon)s_i\}$, where ε is a small positive number, w_i is the maximum download capacity of peer i, and s_i is the signal sent by the source. The result

[9]By specifying the players, the strategies and the outcomes, we have described the game in normal form.

is not strictly a Nash equilibrium, but becomes exact when ε is small.

In the context of BAR systems, that is, a scenario where players can be divided into Byzantine, Rational (selfish), and Altruistic (obedient to the protocol), Clement *et al.*[32] argument against the lack of a clear process for the design of a rational-resilient protocol, and propose a protocol which is proved to be resistant to collusion by using the game theory results. The authors show that the players increase their utility when obeying the protocol. The analysis, however, is based on a cost-benefit model that, once changed, weakens the protocol's resilience.

The process of designing a strategy-proof protocol follows four steps:

1) Describe the protocol as a game
2) Enumerate the possible players' violations
3) Design a BAR-tolerant protocol
4) Analyze the protocol

The study leverages on the results that prove that players do follow a protocol when they have no greater benefit by deviating than by obeying it. The authors prove that the results about k-Fault Tolerant Nash Equilibria, that is, Nash equilibria[10] that are tolerant to up to k selfish peers (including colluders), have too narrow a range of applicability (the Walrasian problems), and that the results about (k, t)-robust protocols (which tolerate up to k rational and t Byzantine players) rely on too strong an assumption, i.e., the inexpensiveness of communication among players. According to these premise, the focus moves to Repeated Terminating Reliable Broadcasting problems (R-TRB), which are a class of protocols that are proved to be resilient to Byzantine nodes. Specifically, they prove that the Dolev-Strong TRB protocol [33] is not a Nash equilibrium, thus allowing players to defect and the social utility to decrease.

To counter the problems of R-TRB, the study proposes a protocol based on Dolev-Strong, called *Just TRB*. The design, however, overcomes the weaknesses of Dolev-Strong's solution, classified as follows:

1) Impossibility to distinguish communication failures from selfish silence
2) Possibility of increasing the long term utility by defecting
3) No punishment for misbehaviour

To clearly distinguish between failures and selfish behaviour, the Just TRB protocol uses predictable communication patterns: the number of messages to exchange is always the same, and, when the phase requires fewer, padding messages are sent. A cost balance decreases the utility gained by not sending messages, and an accountability mechanism induces the certainty of punishment.

The protocol is based on the classification of the players, from each player's point of view, into Friends, Ex-Friends, and Enemies. The status depends on the direct interactions between peers, but once a relationship turns worse, it cannot be rebuilt. The presence of ad-hoc penance messages ensures that a player cannot turn a Friend into an Ex-Friend without motivation.

A key contribution of the paper is the rationality analysis of the protocol presented. The authors prove that their protocol is a Nash equilibrium with the assumed cost-benefit model, thus the players do not have any advantage in deviating from the equilibrium strategy because they would in that case decrease their utility. The analysis enumerates all the possible deviations from the equilibrium, and calculates the benefit gained by following them. The proof is based on the following result: the maximum utility gained by deviating is smaller than the minimum benefit gained by obeying the protocol.

Feigenbaum and Shenker[30] survey the state of the art in the Distributed Algorithmic Mechanism Design (DAMD) field, which can be exemplified as the study and the design of distributed systems where participants, assumed to be self-interested, have to be properly incentivized to follow the system's algorithm. As self-interested, the user will not generally follow the algorithm if this is not an advantage for them. Specifically, DAMD addresses both incentive compatibility and computational tractability in systems where agents and resources are distributed. The authors illustrate the open problems inherent to the DAMD problems formulation: first, they show that it is still missing a measure of the hardness of a problem; second, they question the value of approximated solutions to hard problems; eventually, an overview of solution concepts alternative to Nash equilibria and dominant strategies are investigated, and indirect mechanisms are presented as possible candidates to direct mechanisms, where not strictly required by the domain.

Let's provide some specific definitions to further illustrate the problems mentioned above. The system goals are characterized by the *Social Choice Function* (SCF), which generally does not coincide with the single user's utility. A SCF is *strategy-proof* if no agent has an incentive to lie about his utility, it is *group-strategy-proof* if there is at least one agent suffering penalty whenever other agents collude to get benefits. A mechanism pair (M, S), where S is called *strategy space*, from which agents choose a strategy, and M is a function that maps vectors of strategies into outcomes. The mechanism is *indirect* when we know something about players' utility only through the indirect choice of the strategies from S, *direct* otherwise.

The design of a mechanism depends on a model of the problem. While, however, theoretical computer science has defined a notion of hardness of a problem in both the Turing and the PRAM models, the authors point out that there is still no such distinction for DAMD problems. In this context, it can be defined the notion of *canonical hardness*, which comprises the problems where either computational tractability or incentive compatibility can be achieved, but not both. Now, it is desirable to know whether a hard problem can be approximated by a simpler problem, and how good the approximation is. Only basic results exist about approximation: in particular, strategically faithful approximations are problem

[10] See Appendix A.

approximations where the strategic properties of the original model hold, but other properties are approximated. There exist three such approximations:

1) ε-dominance: A strategy vector $(s_1, s_2, \ldots, s_N)$ is an ε-dominant equilibrium if for every agent i, and strategies t_i and $\vec{t_{-1}}$, it is true that $u_i(M(t_{-i} \cup s_i)) + \varepsilon \geq u_(M(\vec{t}))$. In this approximation, s_i is not the best strategy, but it is within a factor ε of the optimal strategy (also see Li et al. [27] for an example of application).
2) Feasibly strategyproof mechanisms: A better strategy exists, but it is infeasible to compute for ALL the agents, that is, the infeasibility derives from computational limits.
3) Tolerably manipulable mechanisms: A mechanism is tolerably manipulable if it is not group-strategyproof, but we can characterize the types of malicious groups that can form, and we can demonstrate that their effects on the community are tolerable. Such mechanisms answer the question: How large can the effects of malicious coalitions be? For this reason, they are particularly interesting for the study of collusion.

As we said above, the problems from a mechanism design perspective are usually studied with a special focus on dominant strategies and Nash equilibria, which are known to be hard to compute. Alternative strategies may well be thought over, however. In fact, it is realistic, in a distributed environment like the Internet, to assume that the agents not only do not know of the existence of each other, but they observe different payoff for themselves applying the same strategy, for mutated network conditions. In this case, we obtain an interesting simplifying assumption by modeling the system as an iterative game, in which the players do not know about the payoffs of other players, but can iteratively learn them by observing the output of the choices as they appear in the network.

Finally, the authors show that the design of indirect mechanisms offers a trade off between the privacy of the agents, who just partially (if at all) reveal their utility functions, and the network complexity that derives from the need to retrieve that information in other ways, which usually involves the explosion of the number of messages to exchange. It is not proved that indirect mechanisms inherently require a network complexity higher than direct mechanisms, however, so that it would be a result to show examples of scenarios where the former incurs in lower network complexity than the corresponding direct mechanism.

D. Reputation systems

General overviews of the literature are provided in the works by Marti and Garcia-Molina [34] and by Despotovic and Aberer [35].

Marti and Garcia-Molina[34] classify the design needs for P2P applications that require a reputation system. Reputation systems are used against three types of adversaries: selfish peers, malicious peers and Byzantine peers; only one type of adversary, however, (especially selfish peers) is usually considered, while others are neglected.

Against any adversary, a reputation system complies with a number of sometimes contrasting requirements. First, as any system, P2P applications are required to be stable, so as to provide a good service to well-behaved users: resilience against churn (peers connecting and disconnecting from the network) exemplifies this problem, because high percentages of churning might disrupt the service and the quality of user's experience. Second, to build a reputation, it is fundamental to persistently and securely store the data that describe the behaviour of users: in fact, one of the main targets of an adversary is the removal of the proofs of his misbehaviour. Third, users in a distributed environment must be subject to proper admission policies to access the resources of the system, but ideally (and in opposition) also having the option to remain anonymous. Anonymity and access control are clearly contrasting requirements. Persistence of data, on the other hand, is hard to achieve in a continually mutating system.

For a reputation system to manage reputations and resources, a control entity is required. Control can be central or distributed. Naturally, a distributed system should favour the latter; a centralized control, however, reduces the number of entities to trust, while making them a more isolated target for malicious agents.

Now that we have explained the requirements, let's focus on the way to organize the architecture. The main components of a reputation architecture are:

1) the policy of information gathering about peers' interaction,
2) the ranking system with the ways of using it, and
3) the system's response mechanism to a misbehaviour.

The information about a peer binds an identity to a behaviour. It is impossible to build a reputation without assigning it to an individual; the requirement of anonymity, however, may prevent system designers from binding a real identity to a virtual *persona*. As stated previously, for example, Sybil attacks are a way of misbehaving about identities and to falsely build a reputation by using more coordinated identities. The information about a peer can be gathered by hierarchically expanding the single peer's trust: starting with direct interaction, that can be easily recognized as good or bad, a peer may trust his real-life friends' judgment about other peers, then using the neighbours' experience, and so on. A general assumption is that, by aggregating more opinions, the accuracy of information about a peer is likely to become high. Related to the problem of reputation building is the stranger policy, i.e., how to relate with peers that for the first time join the system and thus have no reputation at all.

The ranking system establishes which behaviour influences the reputation. Particularly, since it is hard to prove misbehaviour, it is generally possible to measure the number of good transactions. As we will see later on, transactions, on their side, may have different importance, according to their value for the peers involved and with respect to the resources used. For this reason, both the quality and the quantity of interactions

have an impact on the score assigned. While some solutions use a single scalar value to rank a peer, some systems, like TRELLIS[36], assign arrays of values to separately mark different faces of the reputation of a peer.

The way the system uses the reputation is the last design dimension of reputation systems. A punishment limits the behaviour of malicious users without encouraging contribution by selfish users. Punishments can range from the ejection from the system to the emission of fines, where this latter alternative requires strong identity infrastructure and solid anti-collusion mechanisms. In fact, a group of colluders may slander an honest peer and make the system unfairly fine him. On the other hand, scores can be used as incentives, which encourage the cooperation of selfish peers. Incentives can be money in a money-based system (see Section IV) or a better quality of service.

Despotovic and Aberer [35] make a comparison between the reputation systems implemented using one of two approaches, namely, probabilistic estimation techniques and social networks. The terms of the comparison are:

1) the performance of the estimation of the behaviour of peers given the history of their past behaviours;
2) the computational cost of the estimation;
3) the possibility of using the estimated reputation values to build trust.

In order to formalize the definition of a reputation system, the authors introduce the concept of *trust multigraph*: it is defined as a digraph in which the peers are the nodes and the interaction between a peer p producing a service and a peer q requiring the service defines an edge $p \rightarrow q$ which is weighted by a value that represents an evaluation produced by q about the service received by p. A definition of a *P2P reputation system* is then given as a quadruple (G, A, W, T). G is a directed weighted multigraph defined by (P, V), P being the set of the peers and V being the set of vertices of the graph. W is the set of evaluations from which peers extract the value to assign to the service received, while A is an algorithm that exploits the information contained in the multigraph to produce the value $t \in T$ of the trustworthiness of a peer in the network.

The information that the algorithm A uses to build the multigraph can be imagined as a table storing *(key, value)* pairs representing the identifier of the peer and the trustworthiness value. This table can be stored differently according to the underlying P2P overlay: in a structured P2P overlay, the table can be assigned to a node in a pre-specified way, because the keys are assigned to peers by the key space; on the other hand, in an unstructured overlay (which lacks an organization of the peers' disposition) each peer can store the information about itself and its outgoing edges (the evaluations given to interacting nodes). Since the information stored by peers in a structured graph does not concern themselves, they can earn a profit by misreporting it: in this case, ad-hoc voting systems should be conceived to identify misreported content. The most expensive network operation performed by the algorithm A is the exploration on the graph to gather aggregate feedback and build reputation, which coincides with an exploration of the underlying overlay graph[11]. Now, the main difference between social networks and probabilistic estimation techniques is that the former need to aggregate the feedback values of all the peers of the network to compute the trustworthiness t of any peer, while the latter only need information about the peer and its interactions (conveyed by a fraction of the trust multigraph). The algorithm A can actually exploit this difference in the quantity of information to collect when the underlying overlay is structured, performing the necessary search operations in $O(lnN)$ time (N being the number of peers), unlike the case of social networks, where a flooding is necessary which requires a cost of $O(N)$. However, the costs become the same in case of unstructured overlay networks, where even to build a partial multigraph a flooding is necessary (in both cases, the cost is $O(E)$ where E is the number of edges of the whole trust multigraph).

The main contribution of the paper, other than the analysis of the two approaches, is threefold. First, a definition of a set of collusive patterns which peers can follow in case of misbehaviour is proposed and analyzed through simulation. The scenarios analyzed are four. The population is assumed to divide into two groups: the honest peers, who always report honestly, and the liars (dishonest peers), who misbehave in different ways. Be p_i the probability for peer i to behave trustworthily: the analyzed scenarios are the following:

1) Simple collusion. The liars always misreport about honest peers, and always report 1 (trustworthy behaviour) about peers of the same group.
2) Collusive chain. The liars form an ordered circular set (chain): if there are n liars, then peer c_i always reports 1 for peer c_{i-1} and misreports on all the others, with the additional rule (circularity) establishing that $c_0 \equiv c_n$. Chains are most effective when loops bring gains, which normally is the case in social networks.
3) Two collusive groups. The population of liars further splits into two groups, which we call L_1 and L_2: a peer i belonging to L_1 always behaves honestly ($p_i = 1$), but always report 1 for the service provided by any peer from L_2; a peer j belonging to L_2 always report 1 for the services provided by any peer from L_1, but we do not do any hypothesis about the service it provides (p_j is not necessarily 1). This means that peers of group L_1 gain high reputation in force of their honest behaviour, and acquire high credibility to recommend peers of the group L_2.

A fourth scenario is called *Independent misreporting*, that we report for the sake of completeness but we believe not directly related to the subject of this survey. In this case, the liars always misreport on any other peers, even those from the same group; however, this can *not* be considered a collusive scenario, because malicious peers do not coordinate.

[11]The authors deem the building of an another overlay for the reputation management as too much an expensive operation if compared to the improvements obtained with respect to using the existing overlays.

The second main contribution is the comparison between the two kinds of approach. The results show that the probabilistic estimation techniques perform a better prediction of the peers' behaviour than social networks when the population of collusive peers is a fraction sufficiently far from half of the entire population (between 0.1 and 0.3 and between 0.6 and 0.9), with the only exception of the simple collusion scenario. The only case when the social networks are preferable is when the population of the collusive peers is equivalent in number to the population of the honest peers; however, the costs required to aggregate the feedback, as said before, are to be taken into account.

A third contribution is about the number of interactions required for each peer to provide a sufficient amount of feedback information for the estimation to be performed with tolerable error. The simulation results show that a significant variation in the number of peers in the network has a little impact on the number of interactions: it has been found that 20-30 are sufficient to obtain a good prediction of the trustworthiness of any peer (by *good* meaning that the absolute mean error between the prediction and the actual value is around 0.3). Obviously, different requirements may demand higher- or lower-quality estimations.

As the authors admit, there are some assumptions which may not be valid in the real world. First, the probabilistic models used in the simulations are not proved to model the behaviour of P2P communities: this means that further work is required in this direction. Second, they assume that the information about the trust multigraph is available, while this might not be the case in presence of churning (nodes disappearing from the network may have stored non-replicated content), or simply if not all nodes are on-line. This last objection, however, can be removed whenever effective replication strategies are used.

Marti and Garcia-Molina [37] show how much a pool of trusted peers in a file-sharing system can reduce the number of attempts a peer has to perform in order to obtain a valid resource (a file, in the study). In a system of n agents, they suppose that a peer queries for a resource, and a number of peers owning a copy of that resource replies. Users are supposed to verify the validity of a file by using a function that requires a cost, and the query is re-issued in case the file obtained was corrupted.

The authors conceive a threat model that takes into account the collusion of a group of peers, particularly focusing on front peers that always provide good files but lie about the reputation of malicious peers. To avoid the attacks, a reputation system is designed. The reputation of a target peer is based on two components: the direct opinion of the sender, if he already had interactions with the target, and the (indirect) opinions that other peers have about the target peer. The two components are weighed according to the trust the sender has in the peers that express an indirect opinion about the target.

When peers reply to a query, the sender selects one of them according to different policies. A local policy selects peers which the sender has a direct opinion about, and whose identifiers are stored in a Friend Cache. According to the local policy, a trade-off must be decided about, by choosing whether the most reputable peer should always be selected, thus intensifying the load over him, or if a peer should be selected probabilistically, with probability proportional to their reputation, thus distributing the load among the most reputable peers. A local policy corresponds to a weight of zero about other peers' opinions about the target peer. A voting policy, on the other hand, also considers the other peers opinions. Now, to select the other peers which the sender trusts the opinion of, a selection policy can be used among two (voters policy): select the neighbours, that not necessarily had an interaction with the sender in the past, or select the voters in the Friend Cache. In an adversarial, untrusted environment, choosing from the Friend Cache turns out to be fundamental.

The authors perform experiments with different adversarial settings, analyzing the efficiency of their system in terms of number of files to download to obtain a valid one, the load on good peers, and the traffic generated by the system messages to manage the reputations. The results show that collusion is better countered when peers make an extensive use of the Friend Cache. Particularly, as could be expected, collusion is more effective when indirect opinions are given more weight. As in any reputation system, furthermore, the problem of whitewashing caused by cheap identities may significantly reduce the effectiveness of the reputation system. The authors propose to use a login server to limit the effectiveness of this type of collusive behaviour.

Now that we have a general view of reputation systems, let's examine some solutions described in the literature. The problem of reputation calculation is addressed by Gupta, Judge, and Ammar[38] with a partially decentralized solution that is based on the presence of a central authority, the *Reputation Computation Agent* (RCA), which is assumed to be trustworthy, and two different computation systems. The context the authors have in mind is a file-sharing application, in which a query-based search phase precedes a download phase. Given this scheme, a peer gains a high score (1) by ensuring his contribution in the processing and forwarding of the queries, and (2) by staying on-line during the transfer phase if he is chosen by the requester (the client of the interaction) as the server of the content. By considering these two basic types of contribution, the reputation score is built using objective criteria, that is, peers' contribution do not depend on how good the receiver considers the transaction, as is the case in subjective reputation systems such as EigenTrust[39], but by using objective elements (the quantity of data transferred, weighted through the capability of the involved agents).

More precisely, the reputation score is computed according to two elements: the peer's behaviour, i.e., his contribution in both the search and the download phases, and the peer's capability, i.e., his resources in terms of CPU power, memory, storage, and bandwidth. Once computed, the score is stored locally, but a proof on the validity of its value is replicated in the RCA's records.

The authors illustrate two schemes for the computation

of the reputation score: a Debit-Credit (DC) and a Credit-Only (CO) scheme. The contribution is measured among four dimensions:

1) the credit acquired by processing queries,
2) the credit acquired by uploading content,
3) the debit accumulated by downloading content, and
4) the credit acquired by sharing content,

where the third lacks, of course, in the CO scheme. To compute these components, the system requires a registration procedure for the peers who want to benefit of it, and thus give up their privacy (by making every interested peer aware of their contribution). Each enrolling peer sends a pair of (public, private) keys (PK_p, SK_p for peer p) to the RCA, who then uses a digest of the peer's public key to identify him. The querying peers sign their queries, that are then stored by the serving peers to prove they actually got a request. For this reason, the signed queries are called Proofs of Processing (PPs). The RCA stores a *history of the transactions* between peers, thus preventing a server from asking for credit more than once for each transaction. At the same time, if no interaction occurred between a client and a server, the lying client can be proved to lie because his signature identifies him, and a lying server can be proved to lie because he will not hold a signed request of processing. The RCA's transaction history, however, has to be kept under limited size, thus starting losing records after a given time: this can create problems about accuracy of the reputation score where a peer does not ask for his credit from the RCA for too long.

In the DC scheme, the file transfer produces a credit for the server, and a debit for the client. The procedure is validated by the presence of a *receipt*, that proves the request from the client and the actual delivery of the content by the server. The client produces a signed Requester Portion of the Receipt (RPR) before the transfer occurs, that the server stores as a receipt after checking its authenticity against the requester's public key. At this point, the server serves the content. The requester is forced to produce a RPR to make the transfer start, while the server is forced to serve the content when it accepts, otherwise the client can complain with the RCA for misconduct and lower the reputation of the server or making him kicked out of the system. In the CO scheme, on the other hand, the debit does not exist, so the RPR is sent only after the transfer. The receiver has no disadvantage in sending it, since it does not produce debit; he has no advantages, either. This makes the system dangerously subject to unpaid credit, that could discourage altruistic agents from participating to it.

The sharing credit is eventually computed by the RCA by inspecting his transaction history, thus retrieving the uptime of a peer from the timestamps of the first and last transaction, or by periodically asking the peer himself about his shared content. Both the methods, however, are inaccurate. The first one may miss the presence of peers that are in the system, but for some reason do not participate to transfers; the second method is still more inaccurate, since it depends on the frequency of the polls and on the system used to verify that the polled peer is not lying.

The authors do not explicitly address the problem of multiple identities. In the CO scheme, since no debits are used, it is preferable for each peer to use a single identity to accumulate reputation score. In the DC scheme, however, a peer can use two identities, one for uploads, that accumulates reputation, and another one for downloads, that accumulates debits. The lack of a method to enforce expensiveness of identities creates the possibility of collusion. In particular, in the DC scheme a peer can use multiple identities to avoid debit, while collusion among more agents is ineffective, since one of the colluders has to pay for the download. In the CO scheme, the situation is worse because colluders can claim to have transfers between them, thus accumulating reputation score. Even though less credit could be assigned to peers that always interact with the same partners, this is not wise because the frequent interaction might be the consequence of common interests.

The solution introduced by Ntarmos and Triantafillou [40] is based on a two-layer modular infrastructure which uses a DHT to store reputation values, and uses them to schedule the requests stored in each peer's queue. The two layers separate the problems tackled by the system. A first layer, SAL, monitors the peers' behaviour, and is meant to fight selfishness by using a reputation system. The second layer, SVL, tackles the problem of malicious peers that try to subvert the system.

The reputation system is based on the concept of favor. A peer p asks a favor to a peer q by requesting a resource, r. For each favor granted, the client stores an entry in a list of owed favors, while the server stores an entry in a corresponding list of granted favors. The selfishness of a peer is measured by comparing the number of favors granted to the number of favor owed: the more the list of favors granted is longer than the list of favor owed, the more a peer is altruistic. To implement the infrastructure that stores the reputation values, the SVL layer uses a DHT system that can be created ad hoc when the underlying application does not provide it, or can exploit an existing DHT used by the application itself. Peers use an asymmetric key pair, and are identified by the hash of their public key. This system prevents malicious peers to position themselves in a well-chosen location in the DHT, because their position is determined by their public key. All resources are identified by a UUID, and so are the interactions between peers.

To create reputation, peers interact through transactions. Either peer in a transaction stores a Transaction Receipt (TR) in the form $TR = (client.ID||server.ID, r.ID, timestamp)$, which has the important property of giving a quantitative information about how important a favor is, thus providing a differentiation between transactions involving a small amount of resources from the more resource consuming interactions. The server, as said above, stores the TR in his credits list, while the client stores it in his debits list. When a server is requested a resource, moreover, he can redirect the requesting peer to one of the peers who owe him a favor. The redirect can be recursive, i.e., the peer pointed to can point to another peer

who owes him a favor, and is regulated probabilistically by a node-defined parameter, according to which the peer accepts the redirect or refuses it.

The transactions may involve altruistic and selfish peers. According to their behaviour in a transaction, the peers gain or lose reputation. The reputation enforcement is based on black and white lists. When a peer p interacts with peer q and observes a deviating behaviour, p publishes on the DHT a blacklisting request (BLR). The receivers of the BLR grants the misbehaving peer a second chance according to a probability $P(SC)$, or blacklist him creating a blacklist record. The BLR is structured in such a way that the storing node cannot read the identity of the blacklisted node. Every node periodically probes the DHT overlay looking for black-records regarding him, and if he finds any, he can clear his reputation by starting reciprocating favors to the blacklisting peer.

The other mechanism that determines the reputation of a peer is the white-listing. The local list of favors granted from each peer serves as white-list and is used when requesting resources to a server. Each request receives a *score*, computed as follows: The client chooses a subset of his granted favors and sums up the sizes of the corresponding resources granted. From this value, sent to the server together with the request, the server subtracts the sum of the resource sizes involved in the transactions subject to black-listing, and the size of the requested resource itself. According to the score, the request is inserted in the server's queue. Thus, the scheduling of the requests in the queue depends on the client's reputation, i.e., the size of the favor he has granted.

The system considered so far, however, assumes that the participants are not malicious. To counter the presence of malicious (isolated or coordinated) peers, the second layer (SVL) is designed to provide security mechanisms to prevent the attackers from controlling at their own advantage, and at disadvantage of the system, the resources that describe the reputation system. To this end, SVL slightly modifies the mechanism of the SAL. The TR is now signed by both parties, for accountability. Each node periodically verifies the validity of black records by asking for the corresponding BLR and the TRs stored at the client and server who interacted. The periodical check also gives the nodes unfairly blacklisted (slandered) to file a blacklist request for the perjurer[12]. In the same way, more control is put on the white lists. When a client sends a request, the server checks the validity of a part of his white list according to a probability parameter $P(C)$. The black records, which constitute the second term of the sum that defines the priority of the request, are checked with probability $P(B)$. The values of the probability parameters allows for a trade-off between the accuracy of the control procedures and the overhead introduced in the network to check the validity of the information exchanged.

The authors explicitly consider two types of attack, namely, the Sybil attack and the presence of colluders. The Sybil attack is discouraged by the reputation system, that erases the reputation of a peer when he first joins the network as a newcomer. The presence of colluding malicious coalitions, however, is not tackled extensively. A group of colluders can build a fake reputation for any of its members, and the only way to counter this phenomenon is the counterbalance offered by the presence of black records. If black records are avoided and white records are provided by colluders, the system may prove less effective than in more optimistic cases with weak, uncoordinated adversaries. Moreover, the combination of Sybil attacks and collusion may make the adversarial threat destabilizing.

Banerjee *et al.* [41] propose an agent-based system to build trust in a P2P environment. The following threats are recognized as destabilizing for trust:

1) Free-riding, i.e., the phenomenon of acquiring benefits without providing any. This problem is effectively addressed by mechanisms that provide benefits proportionally to the resources τ shared by the peer p who requests the resource x to another peer q;
2) Collusion. Authors recognize that in order to avoid using the information provided by a clique of colluding peers that try to promote a malicious node, it would be necessary, in the worst case, to exchange messages with the whole network (in the worst case in which it is impossible to be sure about the honesty of the other peers).
3) *Zero-cost identity*, which gives the malicious agents the chance to behave selfishly and cancel their low reputation by logging out of the system, creating a new identity and then logging in again, exploiting the bootstrapping mechanisms meant to favour participation. As we have seen before, this problem is partially addressed by giving greater advantages to identities with high reputation, so that peers are encouraged to cooperate to gain high reputation and greater benefits[13];

In order to fight the presented problems, the authors propose a mechanism based on agents installed at each peer and on the exchange of histories of interactions between pairs of peers (shared history). A client peer q is granted a resource requested to a server peer p according to the *expected utility* p can gain by helping. The expected utility is calculated by exchanging messages with a limited number of neighbour peers in order to collect information about the past behaviour of q: in particular, if the requested resource is of type τ, p will help q if the expected utility $E(p, q, \tau)$ evaluates to a positive number. In fact, in that case the peer p can expect with high probability that the peer q will help it back in the future.

The probability for a peer q of being helped by a peer p at time $\bar{t}$ is the sum of two quantities, in the following form:

$$Pr_{pq}(\bar{t}) = (1 - \alpha) \times local_p(q) + \alpha \times remote_p(q)$$

[12]The randomness of the checks does not provide a tight bound on the time it takes for a perjurer to be punished for his behaviour.

[13]This design choice leaves open the chances to gaining a high reputation and then exploiting it selfishly, and to forging high-reputed identities by malicious users.

where $local_p(q)$ is the information that peer p collected about peer q during past interactions, while $remote_p(q)$ is, on the contrary, the information that p gathers from other peers who had interactions with q in the past. In other words, the $local_p(q)$ is the trust that p grants to q, while $remote_p(q)$ is the recommendation that p receives from trusted neighbours. The factor α weights the two quantities, starting from small values in the initial moments after the system starts, then rising to higher and higher values as the history for each peer becomes more and more populated. This mechanism fights the free-riding problem: if many peers report about the free-rider, it cannot earn a high reputation, thus the probability of being helped is constantly kept low. There is still the possibility for the peers to misreport in collusive scenarios: this problem is discussed in what follows.

The second major problem which the authors try to address is the collusion among peers, who may lie about the reputations they have collected about other peers. Collusion is a real possibility in this system, because it is assumed that there is a network gain when giving and receiving help to/from a peer (as can be observed by the simulation settings[14]). To limit this problem, a peer p evaluates the trustworthiness of the other peers' reports about a peer q through a Bayesian estimation: in a first step, the peers reporting about q are considered trustworthy; then, the estimation of p is updated with the results of the actual interaction with q. The simulations show that after an initial period (which is half of the simulation length) the reciprocative agents correctly identify the selfish colluding agents; unfortunately, the authors do not show which schemes of collusion (e.g., in the sense of Despotovic and Aberer [35]) have been used in the simulations, so it is hard to give an opinion about this result.

The zero-cost identity problem is addressed by introducing a threshold for the reputation that a peer must reach in order to receive help from a server agent. The results show that the reputation scheme is not effective for newcomers, because for their short past interactions history they have few elements to decide whether to help another peer or not. The simulation results show that the scheme works effectively only when the population of colluders is the minority in the system.

E. Reputation through trust

A way to build reputation can be based on the notion of *trust*.

Kamvar, Schlosser and Garcia-Molina [39] present a decentralized algorithm to build reputation in a P2P system, and analyze its effectiveness in several scenarios, including collusive threats. The algorithm is based on the concept of global reputation, which is computed aggregating the local trust information of each node belonging to a subset of the network. Nodes belong to regions of responsibility according to the DHT system described in Ratnasamy *et al*. [42], known as CAN. The reputation of a target peer is computed by a set of M *score managers*, chosen by hashing the unique ID of the target node through M different hash functions. A score manager has to know the set of peers interacting with the target node, known as *daughter*, either receiving a service or providing it[15]: in particular, the nodes that received service from the daughter give the score manager the reputation values about the daughter, which are used in turn by the manager to compute a trust value. The authors show that this trust can be obtained by an iterative computation of the principal eigenvector of the normalized local trust values.

The study analyzes different collusive scenarios. The first collusive threat model is based on the assumption that colluders can form a chain inside of which they assign high trust values to each other. The algorithm effectively limits the attack, but the pre-trusted peers play a key role in this result, because they help the system keep the reputation of colluders low enough to prevent them from being chosen as download sources. However, without pre-trusted peers the algorithm has no means to combat the attack and the colluders irreversibly take over the system.

A variation of this scenario is when malicious peers provide malicious content with a probability f, behaving collaboratively for the rest of the time; at the same time, they form a chain as described above. The simulation results show that in this case colluders have an effective negative effect on the service of the system as a whole, because they earn trust by providing authentic content; specifically, if $f = 50\%$, they obtain the maximum result of diffusing a 30% of inauthentic content. The authors argue that this scenario forces the colluders to spend resources in the system to gain their advantage; however, we believe that this threat model is sensible when we consider a malicious peer whose goal is not the disruption of the system, but a better service than he deserves (see section II.

Collusion starts becoming effective when malicious peers organize into separate groups. The authors suppose that colluders split into two groups: a first group (the Infiltrators) behaves collaboratively with every agent and earns high reputation, but always assigns high trust values to a second group of colluders, who never collaborate (the Parasites). The Parasites earn high reputation for the scores assigned by the Infiltrators. With the same effort spent in the previously described strategy, malicious agents diffuse twice the polluted content. The results compare a trust-based system with a system without trust-based download source selection, showing better performances int the first case. However this result was expected, and the fact is that this kind of threat creates great damage with acceptable effort, and is therefore one of the most effective collusion schemes analyzed.

Another form of (virtual) collusion is the famous Sybil attack. Although it may not be considered a collusive attack (since there is usually only one attacker that creates many ghost identities under his control), a natural variation may be when a group of colluders takes control over a group of already

[14] A peer who helps another incurs a cost of 10, while getting help from a peer gives a saving of 1000.

[15] The paper considers the exchange of files as the service.

trusted peers. The EigenTrust algorithm does not deal directly with the Sybil attack. The authors propose to assign a cost to each identity to restrain malicious agents to create multiple ghost identities that provide high trust values. The algorithm itself performs poorly in this condition.

Eventually, the authors analyze a scenario when a single peer provides with a given probability a malicious executable to the peers requesting legitimate content. This is not, strictly speaking, a form of collusion, but it would be interesting to know how disruptive can be an attack of the same kind set by many colluding agents.

An application of the EigenTrust algorithm is used to perform an empirical study of an existing P2P application.

Lian *et al.* [4] describe the results of an empirical study directed at defining the characteristics of colluding behaviours in a peer-to-peer file-sharing system. The analysis is devoted to prove the validity of four novel measures (*detectors*) by applying them to one month of system logs to detect suspect behaviours. The system analyzed is a file-sharing application where incentives are designed in such a way that the users who upload content are rewarded with, and the users who download content are deprived of points. The key concept is that a user with more points (i.e., with a better reputation) gets faster downloads. This incentives mechanism, managed by a single, central entity, assigns users more points per byte for uploads rather than it takes away for downloads: this means that uploading and downloading the same amount of data produces a net gain. This property can be exploited by colluders to earn fake reputation (that is, without actually providing any benefit to the system) and use them to increase their own benefits (in this case, the download speed). The assumptions made for the definition of detectors are the following:

1) Colluders produce a large amount of traffic with the same content to minimize the number of data uploaded and maximize the number of points gained (*repetition detector*). This approach is ineffective when the colluder covers her traces by slightly changing the uploaded content, because the detector assumes the perfect identity of the content of repeated transmission;
2) Pairs of colluders can upload to each other large amounts of (any) data with respect to the amount of data provided to the rest of the users (*pair-wise detector*);
3) Many identities on the same machine might be an attempt of a colluder of gaining reputation by uploading content to herself. This threat exploits the inexpensiveness of identity creation (like in the Sybil attack) and the corresponding detector is known as *spam account detector*[16];
4) Colluders are likely to keep a facade behaviour by uploading to many peers while at the same time misbehaving by directing most of the uploaded data to a single partner. This behaviour is highlighted by a useful indicator named Traffic concentration (TC), which produces the *traffic concentration detector*. One drawback of this detector (as well as of the previous one) is that it assumes that the identity of the user is univocally determined, which may be actually hard in presence of DHCP/NAT systems.

[16]The authors find out that by combining the pair-wise and the spam account collusion, users can deceive the detectors.

Even though the analysis provides interesting insights into the collusive behaviour patterns, the authors do not know whether the users detected as colluders are actually such, so they cannot contrast their results against a known system. However, in order to prove their detectors' validity, they compare their results with those of the EigenTrust algorithm applied to the same logs. The different results are explained as follows: the EigenTrust algorithm assigns low values (i.e., low reputations, also known as *EigenRanks*) to peers with low-reputation clients. This means that the following relevant scenario is misjudged: a LAN in which a server provides content to the local network, where its clients download most from the external network. The local server is detected as a colluder by the algorithm. On the other hand, colluders may raise their EigenRank by collaborating with the set P of pre-trusted peers.

Eventually, the authors make two reflections:

1) There is a similarity between the collusion threats in peer-to-peer systems and analogous scenarios in the web page ranking research field;
2) The ideal solution would be the joint use of the EigenTrust and the Maxflow[21] algorithm, though not feasible due to the fact that Maxflow is expensive to implement in a distributed environment;

One limitation of the approaches presented so far is that the trust values are computed regardless of the context in which the application runs. Moreover, all transactions receive the same evaluation, without considering their relative importance. In the following, we present a study that addresses these limitations.

Xiong and Liu [43] evaluate a system called *PeerTrust* to build trust in an adversarial environment. Each peer p is assigned a trust value $T(p)$ that measures its reputation in the system, that depends on four intermediate measures. To explain the way trust is computed, let's consider two peers v and w that exchange a service in a transaction; in particular, let's assume w provides the service to v. First, a peer v expresses a feedback in terms of satisfaction he received from the transaction, expressed as $Sat(v, w)$. The feedback of v has a certain degree of credibility $Cr(v)$, which is the second value influencing the trust computation and depends on v's history. Third, a transaction has a value, denoted as Transaction factor ($TF(v, w)$), related to its importance: for example, a query-response interaction to locate some content has a different weight than the transmission of the content itself. Finally, the single community can determine the weight of the trust itself according to internal conventions. This value is expressed by the Community factor, $CF(p)$. If we call $I(u)$ the set of peers that had an interaction with p, then the trust of peer p has the

following expression:

$$T(p) = \alpha \sum_{i=1}^{I(p)} Sat(p,i) \times Cr(i) \times TF(p,i) + \beta CF(p)$$

where α and β are further weights used to give more importance to one term or the other.

The simulation results use a simplified form of the equation, dropping the second term and putting $\alpha = TF(p,i) = 1$, while the credibility $Cr(i)$ is evaluated in the two ways. First, we can express the credibility of a peer recursively using his trust value, dividing the trust value of peer i by the aggregated trust of all the peers that had interactions with p. Alternatively, suppose peer p and peer i have interactions with different sets of peers, $I(p)$ and $I(i)$, whose intersection is obviously the set of peers who interacted with both. If both store a vector where each element is a rating of satisfaction received by a peer, we can define a similarity between the two vectors and measure the credibility of i with respect to the distance of his ratings from p's.

Given the metric for trust computation, we can describe the system. A component installed on each peer serves two purposes. First, an archive stores a subset of the trust values of all the peers in the system; second, a trust manager submits feedback and evaluates trust. Multiple replicas of the trust values are stored distributedly, so to prevent data pollution by coordinated voting. The authors describe a caching system to speed up the aggregation of trust values. A PKI infrastructure ensures the expensiveness of the creation of identities.

An interesting analysis of collusive behaviour is provided. The simulation results show that the credibility computed as a function of similarity between ratings vectors proves effective in filtering out the distorted ratings of groups of colluders. Moreover, the system is studied to rapidly adapt to changes in peers' behaviour, trying to counter the effects of peers who accumulate trust and then abuse of it by misbehaving: in fact, a peer's trust grows up slowly, because only a large number of successful transactions build a high trust value; at the same time, however, trust decreases fast and few bad-rated transactions are enough to drop it. A proper use of a time window prevents peers from using long-time up-time periods and past good behaviour to misbehave effectively in the present: simulations show that oscillating behaviours are chased by correct trust values that follow the behaviour of peers by rising when they behave correctly and rapidly falling down when they misbehave.

An interesting problem for which the authors suggest further study is the one-time attack, where a peer builds a strong reputation and then exploits it once: such attacks are impossible to fight with current reputation systems.

Another way to get past the limitations of the reputation systems is to combine different sources of reputations. Silaghi *et al.* combine a direct and an indirect mechanisms to compute the trust of peers in order to overcome the limitations of the EigenTrust algorithm, explicitly addressing the collusion problem in a peer-to-peer grid system used for distributed computation. In the original system , volunteer nodes (workers) provide their CPU power to run experiments over a large amount of common data sets. A master node distributes computation tasks for workers to run over the data sets, while data sets themselves are distributed using BitTorrent. Collusion is countered by using replication and consensus, that is, a result is deemed valid when a majority of the workers agree upon it. The original system always uses replication to validate the results, with large computation overhead.

To alleviate this load, the authors propose a weighted voting system to assess the validity of results, using trust values to compute a validity score for results. The setting is the following: we have n workers that are assigned a work replicated n times. The n results are collected by the master, who stores a table containing trust values for each worker. Each result r_j is assigned a score s_j in the form:

$$s_j = \frac{\sum_i \phi_{i,j} t_i}{\sum_i t_i}$$

where $\phi_{i,j}$ is 1 if peer i ran the work j, 0 otherwise; the value t_i is the trust for peer i, stored in the aforementioned table. If we define $s_j^* = max_j s_j$, then the result r_j^* is accepted if $s_j^* > \theta$, where θ is a threshold properly chosen to guarantee the coherence of results in the presence of low reputation peers, but always greater than 0.5. Moreover, to avoid that low reputation workers (maybe forming a malicious coalition) undermine the correct result provided by a high-reputation worker, authors require that the lowest reputation peer in a pool delivering the result r_j, say, w_l (pivot), has a trust value $t_l > 0.5$.

The major contribution of the study is the method to build the reputation of the peers. Each peer has two roles in the system: he is a worker, computing the result, and a peer in a content distribution application. The master that assigns the works can observe directly the behaviour of the peers as workers, but not their behaviour in the distribution network.

The direct observation is based on the presence of computation quizzes, used as explained in the following. A new peer starts with a reputation of 0. The master establishes an error rate, ε, tolerable on the number of correct results over the total number of results. The general result is that to obtain an error ε, a worker has to solve $m_{max} = \sqrt{1/\varepsilon} - 2$ quizzes. In this system, however, the authors do not use separate quizzes, but rather, for each result of peer i validated by the master, the peer's reputation is increased by $1/m_{max}$. To avoid the classical front peer attack, the validated results are timestamped: this means that a peer that does not produce results for a given time gets his reputation decreased. Specifically, the master discards the results after a time d_t, subtracting $1/m_{max}$ from the worker that produced it.

The indirect observation is based on the application of the EigenTrust algorithm to the local trust values computed by the peers about their partners. The local reputation that peer i has towards peer j is the ratio $d_{i,j}$ between what i downloaded by j and what he uploaded to j. Moreover, a peer can ban a partner when he discovers that the partner sends corrupt data

or otherwise tries to damage the system. No explicit focus is given to the problem of peers unfairly banning partners (slandering). When the master needs the trust value for peer i, he asks i's partners about their local trust values. If i is in a pool of n workers, and $\lceil n/e \rceil$ workers have him in their blacklist, then the master deems p malicious and discards his result, halving his trust value t_i. If no worker turns out to be malicious through this phase, the master computes a matrix $D = d_{i,j}$ and applies the EigenTrust algorithm. At the end of the process, the diagonal of the matrix D contains the trust values for the peers examined, that are used to tune the trust values obtained through direct observation.

IV. Micropayment systems

This Section introduces a completely different alternative to incentive systems, based on micropayment systems, and compares different implementations.

Yang and Garcia-Molina [13] propose a micropayment system to regulate the interactions between peers. The system is based on floating, self-managed currency (*coins*) under the control of a central entity called broker. To be floating, the coins are required to be taken and given by the same agent without the involvement of a central authority, which is exactly the case in P2P networks, where agents play the roles of both the server and the client; the currency is said to be self-managed because the security issues are managed by the peers, still without intervention of the broker. If we denote by PK_b, SK_b the pair of public and secret key of the broker b, a peer p requests a coin to the broker by paying a sum, and the broker sends him a signed message in the form

$$C = p, sn_{SK_b}$$

becoming the *owner* of the coin. The sn parameter is the unique serial number of the coin.

From this moment on, the owner is responsible for the maintenance of the coin. The owner uses it to pay another peer q in exchange for a service, thus granting q the right to become the *holder* of the coin through an *assignment* that has the form

$$A_{pq} = q, seq_1, C_{SK_p}$$

The broker has no participation in the assignment phase. The holder can *reassign* the coin to a third peer, notifying the owner and making the older assignment no longer valid. In every moment, the owner is aware of who holds the coin and of the history of exchanges, in order to have a proof of acceptance or relinquishment in case of disputes with any peer that has held or still holds the coin.

When any party but the broker happens to be in a downtime phase, the remaining agent addresses the broker to require the reassignment or the cashing of the coin. Since this creates a load for the broker, he charges both the requester and the owner (when it come on-line again) to perform the operation: this encourages peers to stay on-line as long as they can.

The security analysis is based on three invariants:

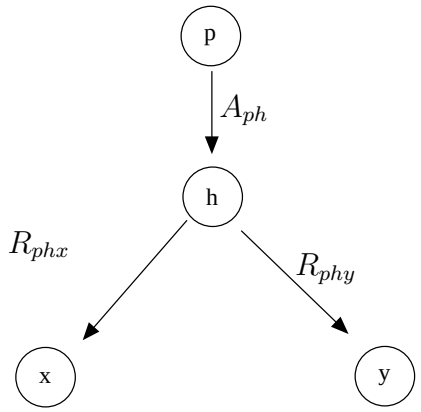


Fig. 4. Coin replication: h tries to reassign C to both x and y

1) If h is the valid holder of coin C owned by p, then p cannot prove that h has relinquished his valid assignment
2) Successive assignments reflect successive sequence numbers
3) The owner can refute any assignment made by an unauthorized holder

The authors analyze how the system is resistant against common attacks to a micro-payment system; in particular, they focus on coin replication, malicious/wrongful denial of an assignment, and double-spending. A brief analysis of these attacks gives us a useful insight into the problems common to most micropayment systems. The first threat described is the coin replication: suppose the owner of a coin, p, assigns it to a peer h, who thus becomes the holder of the coin. Peer h can reassign the coin to a third peer x, and try to reassign it again to another peer y. The sequence is illustrated in Fig. 4. Peer p can prove that the second assignment is invalid because of invariant 3. Suppose now that the owner decides maliciously or wrongfully that the current holder of a coin, h, does not own a coin by valid assignment. Given invariant 1, p cannot maliciously claim that a valid assignment is invalid: it is his responsibility, as the owner, to prove that a holder is using an invalid assignment. Finally, the possibility of double-spending a coin exists. In this case, the owner assigns twice the same coin to two different peers. The broker knows the serial number of each coin associated to each owner, the identity of the owner and the identity of the holder that cashes the coin. If two holders try to cash the same coin, the broker asks the owner to refute one of the assignment: should he be able to refute h_i's assignment, h_i will be punished by the broker; otherwise the owner will be punished. Invariants 1 and 3 ensure the punishment of the owner who double-spent the coin assigned.

The system is extended by adding some features designed to alleviate the load of the broker. First, peers are given the possibility to print coins on their own, given the authorization of the broker. Second, a shortcut mechanism is provided for reassignment, which does not require anymore the involvement of the owner in each reassignment and is based on the concept

of layered coins (a coin gains a layer for each transfer to account the identities of the peers involved). Finally, the introduction of soft credit windows exploits the symmetry of exchanges to avoid useless cashing operations: peers that exchange queries pay each other, thus nullifying the balance of the two transactions.

Collusive attacks against a system like PPay are limited if at all possible. Colluders could, for example, act as owner o and holder h of more coins. Peer h claims it received coins from o and wants to reassign them, but o is offline[17], so he asks the broker to reassign the coins. The goal is to obtain the deposit made by o at the entrance into the system. This strategy is ineffective because h gets the coin reassigned, but o cannot come on-line again, otherwise the broker would charge him with the cost incurred for the reassignment. Even if h gives the sum to o, no gain is obtained, because the recovered money is the original deposit of o, who already owned it.

Forgery of coins, however, can become a problem, because the honest peers may start providing service without actually being paid for it. Although the authors do not address this threat explicitly, it is reasonable to assume that forgery is hard to achieve under the control of the broker.

Based on the prototype developed for PPay, Wei *et al.* [14] consider the anonymity issue WhoPay leverages on the system architecture of PPay, but ensures anonymity of peers that perform a transaction by using group signatures as discussed by Chaum and Heyst[44]. For the sake of fairness, however, the system requires the presence of a trusted entity, the *judge*, that, in conjunction with the broker, can identify the actors of each transaction. By using group signatures, agents are guaranteed to preserve their anonymity, unless they misbehave: in this case, the judge (and only him) is ensured to have the means to identify the peers involved in the transaction.

As an extension of PPay, WhoPay retains the basic structure with a broker and coins that can be purchased, issued, and deposited, but adds the features that make the system anonymous and fair. To highlight the key differences, let's now examine the coins life-cycle. When an agent p decides to buy a coin, he generates a random pair PK_{C_p}, SK_{C_p} and asks the broker to sign the public key. In response to such a request, the broker sends a coin in the form:

$$C = \{p, PK_{Cp}\}_{SK_b}$$

As we can see, the coin is identified by the owner's public key rather than by a serial number. Given this difference, the operations of transfer and issue are perfectly analogue to the corresponding operations in PPay.

This system relies on the involvement of the broker only when the coin has to be produced, deposited or issued, and whenever the downtime protocol has to be executed. In this case, however, the owner and the holder do not incur any fee when the owner is offline and the broker has to operate on his behalf. The authors argument that the operations that are performed in the system are mostly transfer and renewal, that do not involve the broker unless the owner is offline, so the system can be considered scalable. In fact, the other operations' load is evenly distributed among peers, and their number increases as the number of peers themselves.

Basic collusive attacks are briefly discussed and proved to be easy to neutralize by the security architecture. An adversary can collude with the coin owner to force the holder to relinquish the coin; however, the holder can challenge the owner to prove the validity of the transaction, and, once proved it is illegal, he can make the owner be punished for his misbehaviour. The authors, however, do not address the basic attacks based on whitewashing and misbehaviour followed by change of identity. Furthermore, no particular attention is given to systematic collusive attacks, e.g., like distributed denial of service.

The solution presented by Catalano and Ruffo [45] is based on the PPay mechanism and introduces some improvements to further alleviate the load on the broker. In the described system, each time a peer decides to buy copyrighted content, he can do it directly from the author or by any peer who has previously bought it, i.e., reselling is allowed. The original author asks to the broker to issue a certificate that binds the author himself to some content. The certificate is used to prove the relationship author-content. Any time the item is sold, two coins are paid by the buyer, of which one goes to the seller, while the other goes to the author of the content (that may coincide with the seller). Therefore, the interactions involve always the seller, the buyer and the author of the content item.

As an improvement of the basic interaction system, in order to avoid the involvement of the broker in every passage of coin from one peer to another, delegation of accountability is used. The accountability is the possibility of linking an item, be it an object, an action or a right, to a responsible subject, who thus becomes accountable for it. The authors propose a mechanism to pass the accountability of a coin from one peer to another: the first peer, the grantor, passes to the grantee his right to delegate. To implement this mechanism, a second pair of public, private keys is required in addition to the usual one used to identify peers in front of a Certificate authority (the broker). A delegation token is issued from grantor to grantee for each passage, thus it is always possible to reconstruct the chain of exchanges.. The responsibility of such a verification is assigned to the grantee.

The study analyzes the effects of some collusive threats. As an example, efficiency reasons suggest for each peer to verify only the last step of delegation, thus allowing the chance for collusive peers occupying the last two steps in the delegation chain to provide counterfeit coins. Larger groups of colluders may create longer sub-chains, making it harder (that is, more demanding in terms of computation because more steps have to be verified) to discover the misbehaviour. The forgery can be detected from the broker at the end of the passing process (i.e., when the coin has to be cashed), or by any peer that examines the whole (in the worst case) delegation chain. The authors, however, recognize that collusion is not in the scope of the paper and suggest that the topic is an open research

[17]The system prescribes h to ping the owner before contacting the broker.

field in the digital right management discussion.

The micropayment systems considered so far are based on virtual currency, that is cashed cumulatively. A natural alternative is the payment of real money for each transactions, in the sense explained in the following. Nair *et al.* [46] propose a system to incent agents in a BitTorrent-like system to favour the download of content by other agents. Each peer p, at his entrance in the system, generates a $\{PK_p, SK_p\}$ pair and contacts a central authority (the broker b, managed by a content provider), sending it the PK and the coordinates of a valid credit account, used for the payments. As a second step, b sends to p the contact of a tracker, that in turn provides a list of candidate peers to select from and download content. In the same transaction, the broker b also sends p a pseudo-random sequence of numbers, the *tokens*, that p will use to pay the providers of the pieces in which the content is divided. Specifically, one token will be given in exchange of one piece of the content. After the download, p will send the provider q the token, that in turn can decide to redeem it immediately or after some time by contacting the broker. Upon such request, the broker will take the corresponding money from p's account and transfer it to q's. This system incents the upload of content by peers who are paid for their service, at the same time exploiting the resources of the network rather than the resources of the content provider.

The system provides a good defense against the types of collusion we propose at the beginning of the paper; besides, identities are expensive, so whitewashing is not profitable. Let's consider the defenses the system puts against different types of collusion. First, consider a peer x who tries to ask the N pieces of the file to N distinct peers, receives the pieces and then claims he did not actually receive them. If the providers complain against x, the broker can decide to punish him. A colluder, however, can help x by giving him a piece of the content without payment and without complain with b, thus giving x a way to fool the system. The authors study this scenario and find a constraint that relates the number of pieces of a file, the number of outstanding tokens (maximum number of tokens accumulated without asking the broker the payment) and the number of peers in the system, and then show that by carefully selecting these parameters it is possible to reduce this type of attack: conceptually, if the number of pieces is far larger than the number of peers that can be contacted simultaneously, then it is hard for x to find enough colluders to provide him a way to gain his advantage.

While colluders can collaborate to attack the broker using DDoS attacks, the authors propose existing methods to alleviate the problem. Colluders can, for example, try and impersonate the broker by intercepting the requests of the peers; this would, however, require to know the private key the broker uses for any transaction in which it has to ensure about his identity. Still, colluders could simply intercept requests to block them and negating this way the chance for honest peers to get paid for their contribution, making the system's reputation fall down.

Finally, isolating peers is hard to achieve. Suppose for instance that a group of colluders decide to complain against the broker about another honest peer. The system design defines the number of complaints that must receive in order to ban an agent from the working system, so the number of colluders must be quite large (assuming the parameters are wisely chosen by the designers) to fool the broker. In any case, the possibility exists.

V. Collusion prevention enforced by the system

In this section we provide some examples of collusion-resilient systems that do not rely on incentives, be them abstract or in the from of virtual currency. The system, in such cases, can try to base its defense against colluders by checking the location and existence of the content, or the location of the peers in the overlay.

A way to counter collusion is to have a mechanism to demonstrate that an agent owns a content. Specifically, if a peer p declares he received some content from peer q, it is desirable to have a proof that this exchange actually happened: this would prevent the collusion between p (that does not own the file or received it from another peer, e.g., x) and q (that does not own the file or did not upload it to p). This idea is studied by Reiter, Sekar and Zhang [5], who show preliminary simulation results by applying their system to the Maze P2P file-sharing application [47].

An entity, the *verifier*, wants to verify that a set of peers have a resource (file): he sends to each peer a bandwidth puzzle, i.e., a question that can be easily and quickly answered only by the peers who own the file. The question's answer can be found by hashing portions of the file's content in bits; the hash function is universally known by the peers and is modeled as a random oracle[18]. A threshold θ represents the time by which the peers under trial have to solve the puzzle: if a peer exceeds it, then he becomes suspect of misbehaviour. The threshold is chosen in such a way that peers cannot collaborate: a peer that does have the file has just the time to solve his puzzle and send the response. A legitimate owner, however, can solve the puzzle for another peer who lacks the file, at the expense of being accused of misbehaving, thus the system can detect the number of suspects, but not necessarily their identity. One difficulty in choosing the value for θ is that if peers have strongly heterogeneous computational resources, then the value to be chosen should allow the slowest machine to solve the puzzle within the threshold; this, however, means that a fast machine can solve its and other peers' puzzles.

The authors consider the Sybil attack [48], [49] a form of collusion where the Sybil identities collude with their master to boost his reputation. The system counters this kind of threat as well. The mechanism of proof is based on the existence of a hash primitive modeled as a random oracle. Under this assumption, the article shows that a bound exists to the number of puzzles a set of colluders can collectively solve. This bound has a closed form but is hard to compute. To solve

[18] A *random oracle* is the abstraction of a function that can produce a truly random output, and gives the same response to the same query.

this problem, the authors prove the existence of a tighter but computable bound, not in a closed form. The model of collusion is based on the work by Lian et al.[4], especially the collusion graphs.

Although in theory the scheme does not require a central authority, in the sense that any peer can be the verifier, the simulation provides results from the Maze system, which has a central authority that is chosen to act the role of the verifier. The results are encouraging and show that the bandwidth puzzles prevent the colluders from:

1) degrading the performance of the legitimate users
2) obtaining an unfair advantage from fake transactions

An interesting approach is the separation of nodes into classes: a class, less numerous, provides the message passing services and is designed to be hard to pollute with the presence of colluders; another class uses the former to look for content. In this case, like in the systems analyzed so far, the global welfare of the system and its participants is enforced through design rather than by the user's behaviour. Anceaume *et al.* [50] combine existing techniques, originally developed to ensure the resilience of distributed systems against malicious attacks, to design an architecture that limits the negative effects of collusion. They describe a structured P2P system where the management of the overlay is provided by a subclass of the peers, called *core* members, where the other peers (*spare* peers) do not participate directly to the routing operations. All the peers are identified through a secure hash of their network address, and those who share a prefix in the resulting ID form a *cluster*. Each cluster contains both core and spare members, with the former providing interconnection with other clusters. Clusters together form a hypercube, which is a structure proved to exhibit properties of easy management, because the process of interconnection is recursive and easily automated. Data are positioned in the overlay according to an identifier, taken from the same address space of the peers' IDs, called *key*. All the peers in the same cluster are responsible for the same data. The operations performed by the peers are designed to be resistant to attacks by groups of peers, colluding or not.

Peers use three primitives to obtain service and interact with the system. Peers looking for data perform a `lookup(key)` to locate a data item: spare members forward the lookup query to a core member inside the same cluster, that forwards it to other clusters when needed. The path to the destination is retrieved by consulting the routing tables of core members. Core members use a broadcast primitive to forward the duplicates of the message into more than one path. The presence of multiple copies gives the possibility of using the following mechanism of consensus: The forwarding of messages between clusters (on either way) is disciplined through a quorum-based consensus algorithm: if the forwarder receives at least $quorum$ identical messages, it can be proved that with high probability the message has not been polluted by malicious peers. The number of malicious collectives tolerable by such system is well defined and determines the probability of pollution.

The *join* and *leave* operations are used by peers to access or go away from the system. The *join* operation prevents the possibility for a malicious peer to put itself in the core set, by broadcasting a request for all core members to form the core set *ex novo*, while newcomers are always inserted as spare members. The *leave* operation is performed by a leaving peer, or automatically issued by a group of core members of a cluster upon detection of a peer failure.

After a sufficient number of *join*/*leave* operations, the dimension of a cluster may become so small or so large that the maintenance of the hypercube topology becomes no longer possible. For this reason, two thresholds are established: S_{max} is the number of members a cluster can contain before splitting into two smaller clusters, while S_{min} is the minimum number of members under which the cluster is required to merge with another. With this mechanism, a smart adversary can control a number of malicious peers and make them leave or join the system to make a cluster oscillate between a number of merges and splits. This attack is countered by delaying the formation and split of clusters in such a way that a predefined number of colluders in a cluster can not cause them: namely, if a fraction μ of the peers in a cluster tries to issue such an attack, the mechanism can be designed in such a way that a fraction $x > \mu$ of joins/leaves is required for the cluster to actually change dimension by split/merge.

The way of choosing the members of the core set prevents colluders from issuing an Eclipse attack, i.e., a pollution of the routing tables. The structure of the lookup primitive is based on independent paths to ensure that more than one message is forwarded and then more than one reply comes back, thus making the consensus algorithm work.

Colluders can be isolated by leveraging techniques borrowed from distributed systems, like majority consensus. The impact of colluders can be effectively limited if proper voting schemes are designed and colluders are not a majority of the population. With this idea, Corman, Schachte and Teague [51] describe a protocol (the *Secure Group Agreement* protocol or *SGA*) to form groups of peers with a majority of honest agents with a predefined probability, on top of a structured DHT-based P2P overlay. For this system, in theory no central authority is required but to act as a public key infrastructure, needed when the nodes first join the network and are assigned a key pair[19]. Given the infrastructure, the authors make basic assumptions about the characteristics of colluders. The study, in fact, assumes the presence of a strong attacker, able to create, delete or modify the network traffic, and a secure routing infrastructure, such that the mapping *key-NodeID* is reliable and secure.

Now we describe the basic functioning of the system. Honest peers gather into groups sharing a purpose, that depends on the application: for example, in an on-line game, a group may want to share the state of a portion of the game. A group is created by a group initiator p, who is supposed to be able to verify the membership of all the nodes inside the group. To

[19]We think, however, that the verification of signatures makes constant use of a PKI, thus limiting the validity of this assumption.

create it, he sends a message $\{content, hash(content)\}_{SK_p}$ to all the members of the group. The $content$ value of the message is composed of five elements: a timestamp, p's unique identifier *NodeID*, the group's purpose, the group's size, and a string of concatenated unique identifiers to help members understand whether they are part of the group or not, and optionally verify the membership rights of other group members. The timestamp defines a time window inside which the results of the hash are valid, to limit the number of guesses an attacker can do at the output of the hash function. The group size is explicitly given, so each member can know it, and no attacker can make an honest peer believe that a minority of colluders is the size of the group.

Once the message is received, all the peers reply to all the other peers of the group. Two observations can be made on the message passing mechanism: first, malicious peers have no advantage in deleting messages because honest peers will send them to any group member they have not received any notification from; second, this kind of message flooding is particularly expensive when the group size is large, because it requires $O(m_2)$ messages if m is the size of the group.

Now that we have described the mechanism, let's see how the formation of groups is guaranteed to be safely ensured against being subverted by colluders and corrupt peers. To quantify the security of the group, the authors compute the probability of forming a group with at least t corrupt members as a function of the number of peers n in the network, the size of the groups m, the number of guesses an attacker can perform (g), and the number of corrupt nodes in the whole network (c) and in the group (t):

$$Pr\{t\ over\ m\} = 1 - \left(1 - \frac{\sum_{i=t}^{m} \binom{c}{i}\binom{n-c}{m-i}}{\binom{n}{m}}\right)^{g}$$

According to the previous expression, the probability of success for the attacker to correctly guessing the output of the hash function increases as g increases, thus it is sensible to endeavour and limit it. The value of g is limited by the freedom of the attacker in varying the parameters, his computational power, and the width of the time window inside which the guess is valid. It is a design trade-off to decide between large timestamps that define large time windows, and the increased possibility for the attackers to having more time to perform their guesses. Another trade-off is the size of the group: small groups have cheap communication overhead, but give greater possibilities to colluders to pollute the majority. The size of the group can, in turn, be influenced by the size of the whole network: larger populations of peer increase the difficulty for colluders to effectively pollute the application.

Even a system designed by balancing the previous trade-offs, however, does not completely defeat a collusive collective. Attackers can in fact form groups of only colluders with past timestamps, declaring the group existed before. This problem is not explicitly addressed, because its danger is related to the specific application. In general, we can conclude that the work provides an interesting quantification of the probability that a sufficient number of colluders can subvert a system, where the notion of sufficient number varies according to the specific application domain.

VI. Conclusion

In this work we analyze the problem of collusion among autonomous agents by comparing several current solutions designed to provide security and trust in P2P applications. Our main contribution is the clarification of what can be meant as collusion, the definition of the scenarios in which collusion can show up, and the analysis of how well the current systems fit to the purpose of countering colluders. We distinguish between malicious collectives and selfish collectives, in that the former want to create harm to the system or to its users or to both, while the latter use their number to earn advantages (i.e., better service).

Even though the literature about security in P2P systems is vast, there are not so many solutions explicitly designed with the purpose of limiting the collusion. We identify, however, a number of relevant results from the recent literature that help us to understanding the main issues related to the topic. The lessons learned can guide the research effort in a more defined direction and put the problem in more consideration by the research community for further study.

Appendix

In this appendix we give a brief explanation of what we mean by Nash equilibrium, and we will illustrate its application in a famous setting, the Prisoner's dilemma (PD). We consider a particular class of games, the static games with complete information, characterized by the simultaneity of the moves played by the players, and by the knowledge that each player does have about the utility functions of each other.

In the case of static games with complete information, a game can be formally described in *normal form* when it is possible to formally express three elements: first, the number of players involved; second, the strategies they follow; third, the utility functions which depend on those strategies and that players want to maximize. We can indicate the generic player by a number i in a set of N players. The strategies are the possible actions the player can play. In this case, we can express a game as $G = \{P, S, U\}$, where P is the set of players, S the set of strategy spaces $S_1, S_2, \ldots, S_N$ from which the players choose a strategy $s_i \in S_i$, and U is the set of utility functions (payoffs) $u_1(\vec{s}), u_2(\vec{s}) \ldots, u_N(\vec{s})$, $\vec{s}$ being the vector defined as $\vec{s} = (s_1, s_2, \ldots, s_N)$.

A Nash equilibrium is a set of strategies $\vec{s^*}$ such that no player i can increase his utility by unilaterally (i.e., regardless of all other players' choice) deviating from the corresponding strategy s_i^*. The Nash equilibrium is a strong state in which players know that each player (who is rational) will abide by the strategy described by the equilibrium formulation for his own advantage.

Two-player games are a particular class of static games with complete information, where the payoff of each player can be described by a matrix. The element (i, j) of the matrix is the

	Betray	Do Not Betray
Betray	1,1	0,5
Do Not Betray	5,0	4,4

TABLE V
THE PRISONER'S DILEMMA

payoff that player 1 and 2 obtain by playing the strategies i and j, respectively. This said, let's consider the payoff matrix in Table V. This matrix describes a well-known game called the *prisoner's dilemma* (PD). Two players are supposed to be arrested and put in separate cells, so neither knows what the other player's move. The players have to possible strategies: they can stay silent, or they can betray the accomplice. As we can see, if neither betrays, both are condemned to 4 years of prison; on the other hand, if both betray, they obtain a discount. The interesting thing is that if only one of them betrays, he is set free, while the accomplice gets the maximum damage. It is possible to demonstrate that the only Nash equilibrium of the game is for both players to betray, without knowing what the other player will do. Intuitively, we can see why this is an equilibrium strategy by analyzing the perspective of player 1. This player can decide to play Betray or Not Betray. If he chooses the first, then, depending on his accomplice's choice, he can get the minimum cost (0) or a cost of 1. If he plays Not Betray, he can get 4 in the best case (the accomplice does not betray either), but 5 in the worst case. By considering the worst cases for both the strategies, the choice that minimize the risk is the betrayal (the choice is between 1 and 5).

REFERENCES

[1] Noam Nisan and Amir Ronen. Algorithmic Mechanism Design (extended abstract). In *STOC '99: Proc. of the 31st annual ACM symposium on Theory of computing*, pages 129–140, New York, NY, USA, 1999. ACM.

[2] Tirthankar Ghosh, Niki Pissinou, and Kia Makki. Collaborative Trust-based Secure Routing Against Colluding Malicious Nodes in Multi-hop Ad Hoc Networks. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 224–231, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Hui Zhang, Ashish Goel, Ramesh Govindan, Kahn Mason, and Benjamin Van Roy. Making Eigenvector-Based Reputation Systems Robust to Collusion. *Algorithms and Models for the Web-Graph*, 3243(2004):92–104.

[4] Q. Lian, Z. Zhang, M. Yang, B.Y. Zhao, Y. Dai, and X. Li. An Empirical Study of Collusion Behavior in the Maze P2P File-Sharing System. In *IEEE ICDCS'07: Proc. of the 27th Int'l Conference on Distributed Computing System*, 2007.

[5] Michael K. Reiter, Vyas Sekar, and Zhenghao Zhang. Making Contribution-Aware P2P Systems Robust to Collusion Attacks Using Bandwidth Puzzles. Technical Report CMU-CS-08-156, School of Computer Science, Carnegie Mellon University, 2008.

[6] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[7] Shanshan Song, Kai Hwang, Runfang Zhou, and Yu-Kwong Kwok. Trusted P2P Transactions with Fuzzy Reputation Aggregation. *IEEE Internet Computing*, 9(6):24–34, 2005.

[8] Chrysanthos Dellarocas. Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 150–157, New York, NY, USA, 2000. ACM.

[9] Jordi Sabater and Carles Sierra. Reputation and Social Network Analysis in Multi-Agent Systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 475–482, New York, NY, USA, 2002. ACM.

[10] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. Notions of Reputation in Multi-Agents Systems: A Review. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 280–287, New York, NY, USA, 2002. ACM.

[11] Silvio Micali and Ronald L. Rivest. Micropayments Revisited. In *CT-RSA '02: Proc. of the The Cryptographer's Track at the RSA Conference on Topics in Cryptology*, pages 149–163, London, UK, 2002. Springer-Verlag.

[12] N. Asokan, Phillipe A. Janson, Michael Steiner, and Michael Waidner. The State of the Art in Electronic Payment Systems. *Computer*, 30(9):28–35, 1997.

[13] Beverly Yang and Hector Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. In *CCS '03: Proc. of the 10th ACM conference on Computer and communications security*, pages 300–310, New York, NY, USA, 2003. ACM.

[14] K. Wei, A.J. Smith, Y.-F.R. Chen, and B. Vo. WhoPay: A Scalable and Anonymous Payment System for Peer-to-Peer Environments. pages 13–13, 2006.

[15] Alvin E. Roth. The Economist as Engineer: Game Theory, Experimentation, and Computation as Tools for Design Economics. *Econometrica*, 70(4):1341–1378, 2002.

[16] Richard T. B. Ma, Sam C. M. Lee, John C. S. Lui, and David K. Y. Yau. Incentive and Service Differentiation in P2P Networks: A Game Theoretic Approach. *IEEE/ACM Trans. Netw.*, 14(5):978–991, 2006.

[17] Yang-hua Chu, John Chuang, and Hui Zhang. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *PINS '04: Proc. of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 205–212, New York, NY, USA, 2004. ACM.

[18] Zhengye Liu, Yanming Shen, Shivendra S. Panwar, Keith W. Ross, and Yao Wang. Using Layered Video to Provide Incentives in P2P Live Streaming. In *P2P-TV '07: Proc. of the 2007 workshop on Peer-to-peer streaming and IP-TV*, pages 311–316, New York, NY, USA, 2007. ACM.

[19] Tsuen-Wan Johnny Ngan, Dan S. Wallach, and Peter Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *Proc. of 2nd Int'l workshop on Peer-To-Peer Systems*, 2003.

[20] C. Buragohain, D. Agrawal, and S. Suri. A Game Theoretic Framework for Incentives in P2P Systems. In *P2P 2003: Proc. of the 3rd Int'l Conference on Peer-to-Peer Computing*, pages 48–56, Sept. 2003.

[21] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proc. of the 5th ACM conference on Electronic commerce*, pages 102–111. ACM New York, NY, USA, 2004.

[22] Robert M. Axelrod. *The Evolution of Cooperation*. Basic Books, New York :, 1984.

[23] Raph Levien and Alexander Aiken. Attack-resistant trust metrics for public key certification. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium*, pages 18–18, Berkeley, CA, USA, 1998. USENIX Association.

[24] Michael K. Reiter and Stuart G. Stubblebine. Authentication Metric Analysis and Design. *ACM Trans. Inf. Syst. Secur.*, 2(2):138–158, 1999.

[25] Ruggero Morselli, Jonathan Katz, and Bobby Bhattacharjee. A Game-Theoretic Framework for Analyzing Trust-Inference Protocols. In *Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[26] Idit Keidar, Roie Melamed, and Ariel Orda. EquiCast: Scalable Multicast with Selfish Users. In *PODC '06: Proc. of the 25th annual ACM symposium on Principles of distributed computing*, pages 63–71, New York, NY, USA, 2006. ACM.

[27] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience vs Choice in Cooperative Services. In *OSDI'08*, Dec 2008.

[28] M. Haridasan and R. van Renesse. Defense against Intrusion in a Live Streaming Multicast System. pages 185–192, Sept. 2006.

[29] Ian A. Kash, Eric J. Friedman, and Joseph Y. Halpern. Manipulating Scrip Systems: Sybils and Collusion. *CoRR*, abs/0903.2278, 2009.

[30] Joan Feigenbaum and Scott Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *DIALM '02: Proc. of the 6th int'l workshop on Discrete algorithms and methods for mobile computing and communications*, pages 1–13, New York, NY, USA, 2002. ACM.

[31] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism Design for Policy Routing. In *PODC '04: Proceedings of the twenty-third*

annual ACM symposium on Principles of distributed computing, pages 11–20, New York, NY, USA, 2004. ACM.

[32] A. Clement, H. Li, J. Napper, J.-P. Martin, L. Alvisi, and M. Dahlin. BAR primer. In *DSN'08: IEEE Int'l Conference on Dependable Systems and Networks With FTCS and DCC*, pages 287–296, June 2008.

[33] D. Dolev and H. R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[34] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: Categorizing P2P reputation systems. *Computer Networks*, 50(4):472 – 484, 2006.

[35] Z. Despotovic and K. Aberer. P2P reputation management: Probabilistic estimation vs. social networks. *Computer Networks*, 50(4):485–500, 2006.

[36] Yolanda Gil and Varun Ratnakar. Trusting Information Sources One Citizen at a Time. In *ISWC '02: Proc. of the 1st Int'l Conference on The Semantic Web*, pages 162–176, London, UK, 2002. Springer-Verlag.

[37] Sergio Marti and Hector Garcia-Molina. Limited Reputation Sharing in P2P Systems. In *EC '04: Proc. of the 5th ACM conference on Electronic commerce*, pages 91–101, New York, NY, USA, 2004. ACM.

[38] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A Reputation System for Peer-to-Peer Networks. In *NOSSDAV '03: Proc. of the 13th int'l workshop on Network and operating systems support for digital audio and video*, pages 144–152, New York, NY, USA, 2003. ACM.

[39] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW '03: Proc. of the 12th int'l conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM.

[40] N. Ntarmos and P. Triantafillou. SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks. In *Proc. of the 4th Int'l Conference on Peer-to-Peer Computing*, pages 116–123, Aug. 2004.

[41] D. Banerjee, S. Saha, S. Sen, and P. Dasgupta. Reciprocal Resource Sharing in P2P Environments. In *Proc. of the 4th int'l joint conference on Autonomous agents and multiagent systems*, pages 853–859. ACM New York, NY, USA, 2005.

[42] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[43] Li Xiong and Ling Liu. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.

[44] D. Chaum and E. Van Heyst. Group signatures. In *EUROCRYPT'91*. Springer-Verlag, 1991.

[45] D. Catalano and G. Ruffo. A Fair Micro-Payment Scheme for Profit Sharing in P2P Networks. pages 32–39, Oct. 2004.

[46] S.K. Nair, E. Zentveld, B. Crispo, and A.S. Tanenbaum. Floodgate: A Micropayment Incentivized P2P Content Delivery Network. pages 1–7, Aug. 2008.

[47] Mao Yang, Yafei Dai, and Xiaoming Li. Bring Reputation System to Social Network in the Maze P2P File-Sharing System. In *CTS '06: Proceedings of the International Symposium on Collaborative Technologies and Systems*, pages 393–400, Washington, DC, USA, 2006. IEEE Computer Society.

[48] Douceur, John R. The Sybil Attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer 7Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[49] J. Dinger and H. Hartenstein. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8 P2P.–, April 2006.

[50] E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro. PeerCube: A Hypercube-Based P2P Overlay Robust against Collusion and Churn. In *SASO '08: Proc. of the 2008 Second IEEE Int'l Conference on Self-Adaptive and Self-Organizing Systems*, pages 15–24, Washington, DC, USA, 2008. IEEE Computer Society.

[51] A.B. Corman, P. Schachte, and V. Teague. A Secure Group Agreement (SGA) Protocol for Peer-to-Peer Applications. In *AINAW'07: Advanced Information Networking and Applications Workshop*, volume 1, pages 24–29, May 2007.