Reputation Management in P2P Networks: The EigenTrust Algorithm

by
Adrian Alexa

supervised by
Anja Theobald

1 Introduction

Peer-to-Peer networks is a fast developing branch of Computer Science and many researchers are developing new algorithms for such systems.

After the success of systems like *Gnapster, Kazza*, the use of such networks became a necessity. Unfortunately the open and anonymous nature of these systems, makes the surveillance of the network almost impossible, leaving many open doors for malicious individuals, who want to profit from what the system is offering.

This *lack of accountability* opens new problems for *P2P* Systems, like: **Anonymity, Scalability, Incentives, Security, Trust, Load Management, Correctness, etc**.

In this paper we will focus on the *Trust Management* for the *P2P* networks trying to explore the possibilities of this kind of management in completely decentralized systems. We will describe the EigenTrust Algorithm proposed in [Eigen 03] for this problem.

Our setup is a *file-sharing* peer-to-peer system (similar to *Kazza*). Our aim is to reduce the number of inauthentic files that are shared in such a system. We also assume that we have mainly two types of peers that are participating in the network:

- *good peers:* peers that join the network for receiving and offering services from/to others peers. This peers will share mainly authentic files, and we can assume further that the probability that an inauthentic file came from a good peer is low;

- *malicious peers:* peers that are joining the network just for destroying it or for taking control of it. This peers will always share inauthentic files.

The assumption that all inauthentic files are coming from malicious peers is not a good one, because mistakes can happen with good peers (for example a good peer will not delete the files it downloads immediately, so this peers can share this files without knowing if them are authentic or not). But we will think malicious peers as the source of inauthentic files.

Now our problem can be restated as:

minimize the number of inauthentic files spread by malicious peers in a P2P System.

Thus we will set our goal to *identify sources of inauthentic files and bias other peers belief when downloading from them.* We will see that a nice and elegant way to do this is to *assign peers*

global trust values based on previous peer's behavior.

But before going further into details we should clarify what for *trustworthy peers*, and what is Trust Management?

> **Trust Management** is a mechanism that allows to establish *mutual trust* ("allows participants to a network to cooperate in a *game-theoretic* situation that corresponds to the repeated prisoner dilemma and leads in the long term to an increase aggregated utility for the participants" [MTrust 01]).

The best framework for explaining the notion of Trust Management is the *e-commerce* framework and the most representative example is **eBay** online auction system. In eBay, users are selling and buying products one from each other and accumulate *experience* with each transaction made. When a user wants to buy a product, he will search for users who are selling that product. He will decide to buy from a seller with a good *experience*. After the transaction is done both parts will rate one each other depending on how the transaction finished and the actual rating will be add to the previous experience. The same happens when a user wants to sell a product. Thus in this kind of systems users are encouraged to *behave nice* because otherwise they will be isolated by all other users and thus from the system.

In eBay the *experience* or *reputation* of an user, represents the **trust** that all other participants puts in that user. It is worth noticing that this reputation score is unique per user, so is a *global trust* score. We will follow this idea of user ratings after each transaction in the development of the algorithm. The main problem with eBay Trust Management is that it is a **completely centralized** system where the reputation score is managed by some servers (eBay authority).

We want to solve exactly this centralized management problem. For this lets see what are the main issues that we need to address when designing a *P2P* network:

- **anonymity:** peer's reputation should be associated with an *opaque* ID. For example in Gnutella peers know each other by a user-name and not by their IP address, which is an external identity.

- **self-policing:** there is no central authority. Peers should define and impose shared ethics between them.

- **minimal overhead:** peers should not spend too much time and bandwidth for computing their reputation score.

- **robust to malicious collectives:** the system should try to isolate collectivities who want to subvert it.

- **newcomers are not privileged:** malicious peers can't gain anything by changing their identity. Also newcomers should obtain their reputation by good behavior within the system.

We can see that this issues are quite general and they should be considered for any *P2P* system, not only for a file-sharing system.

2 Global Trust Values

Before starting to design our algorithm we will show how we can use *global trust values* in a *P2P* system to asses different issues. First we examine in which way global trust values can help in reducing the load over peers and how decisions on the peers to download from can be made.

2.1 Load Management

In a *P2P* network in which there is no notion of trust one peer is choosing the peer from which to download the desired file randomly from all peers who respond to its query. In this manner the total load of the system will be distributed uniformly among all peers.

This should not happen in a system in which we know that malicious peers exists. So lets see how we can use the notion of global trust for this. Suppose peer i submits a query in the network for searching a file, and receives acknowledgement for R peers with trust values: $\{t_1, t_2, \ldots, t_R\}$. We can think at two strategies:

- **choose deterministic:** peer i chooses peer j which has the highest trust value: $t_j = \max_k (t_k)$. As we can see this is not a wise strategy because there can be greedy good peers that will share lots of files and will respond almost to all queries. This peers will accumulate trust, increasing their global trust value, but the load on them will increase even more leading to network congestion for that peer. We can say that this peers tend to become *local servers*. Also this strategy blocks new peers in accumulating trust (they will never be chosen as a download source).

- **choose probabilistic:** peer i chooses a peer j with probability:

$$P(i \text{ choose } j) = \begin{cases} \frac{t_j}{\sum_k t_k}, & \text{if } t_j \neq 0 \\ 0.1, & \text{if } t_j = 0 \end{cases}$$

  Thus we give a chance of 10% to new peers but also malicious peers to be chosen as a download source. Here 10% is a *magic* number and usual is inferred from simulations. We can easily see that this approach tries to balance the load per each peer, without giving malicious peers too much chance.

A general algorithm for load distribution can look like this:

Load Distribution Algorithm

get $T = \{t_1, t_2, \ldots, t_R\}$

repeat

choose a peer j with $t_j \in T$ (deterministic or probabilistic)
if receive inauthentic file form j **then**
delete t_j from T

until authentic file received

In Figure 2.1 is shown the performance of both strategies versus a *P2P* system in which there is no notion of trust. It can be seen that the deterministic approach is performing poorly favoring one peer. The probabilistic approach is more close to the non-trust based system. More details on how these experiments were done can be found in [Eigen 03].

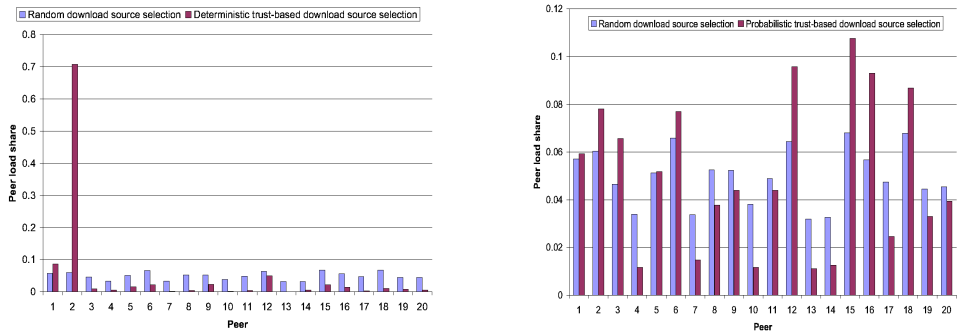

Figure 1: *Different strategies for Load Distribution in trust based and non-trust based systems*

2.2 Dealing with Malicious Peers

There are two ways in which we can use global trust values when dealing with malicious peers. One is to use global trust values for *isolating malicious peers* and the other is to *incent peers to share*. We explain shortly how these are done:

- **Isolating Malicious Peers**
  As we state in Section 1 we will try to isolate (eliminate) malicious peers from the network in order to minimize the number of inauthentic files spread in the network. In order to do this we must not choose malicious peers as download source. Thus we fell mostly in the same cases that were discussed at Load Management:
  - Download from the most highly trusted peer. We have seen that this is a very bad strategy, lifting the global trust for some peers, and not allowing newcomers to build trust.
  - Select peers from whom to download based on the distribution induced by peer's trust value. This seems more reasonable and the experiments have shown that this policy isolate malicious peers without penalizing newcomers.
  - But peers can be more conservative about the global trust. If a peer have had a bad experience with some peer that has a high global trust, will try to choose another peer from whom to download, thus combining *local trust* with the global trust:

    $$t = at_{\text{global}} + (1 - a)t_{\text{local}}$$

  Combining the last two strategies seems a good way to minimize the amount of inauthentic files that a peer will receive.

- **Incent Peers to Share Files**
  Also we want the system to be attractive for new peers that want to join and somehow to *"force"* peers to share. The system can reword high trusted peers. As an example these peers can receive greater bandwidth. Also rewording hight trusted peers can incent them in deleting inauthentic files that they accidently downloaded. Another benefit of rewording based on trust is that this policy will reduce the number of free-riders in the network: either these peers will decide to share files or they will be isolated (their trust will remain close to 0) from the network.

3 Local and Global Trust Values

We have seen above why we need global trust values. Now we will start to show how can one obtain such global trust values in a *P2P* file-sharing network. We start from local trust values modeled after eBay reputation management system and we show what modification need to be made for aggregating these values without centralized management. Then we will see that aggregating them can led to some global values, not surprisingly if we think our scores as probabilities. Lets begin with local trust values.

3.1 Local Trust Values

Like in eBay, in a *P2P* file-sharing system peers will rate each other after every transaction made. If peer i downloaded a file from peer j then if the file was authentic it will set $tr(i,j) = 1$, otherwise it sets $tr(i,j) = -1$. We set the local trust to $s_{ij} = \sum tr(i,j)$. More formally if:

- $sat(i,j)$ = the number of satisfactory transaction
- $unsat(i,j)$ = the number of unsatisfactory transaction

Definition 3.1 *The local trust value that peer i have in peer j is:*

$$s_{ij} = sat(i,j) - unsat(i,j)$$

The main problem with this values is that we cannot aggregate them. One reason will be that the values of s_{ij} cannot be well interpreted. For example the fact that a peer i has $s_{ij} = 10$ for a peer j and another peer k (different from i and j) has $s_{kl} = 1000$ would not tell us too much, because it is possible that peer i is new and have made only 10 transaction with peer j, and peer k has made lots of transactions: $sat(k,l) = 2000$ and $unsat(k,l) = 1000$. Thus aggregating these values would not help. Another drawback is that malicious peers can report a very high local trust, subverting the system. To overcame this we will normalize the local trust in the following manner:

Definition 3.2 *The normalized local trust, c_{ij} that peer i has about peer j is*

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)}.$$

Remark 3.3 *c_{ij} can be seen as a probability:*

- $0 \leq c_{ij} \leq 1$
- $c_{i1} + c_{i2} + \cdots + c_{in} = 1$

We will see later in next section how this interpretation can help us in developing an algorithm for computing trust values.

There are some drawbacks even with this normalized trust. One is that now we cannot distinguish between peers that have bad experience or did not interact at all with peer i ($c_{ij} \approxeq \max(s_{ij}, 0)$). Other is that c_{ij} is not defined if peer i doesn't have any interaction with all its neighbors or the neighbors with whom he interact gave him inauthentic files. Also this problem will be addressed later.

Next step is to aggregate these normalized local trust values. From now on, we will use local trust values for normalized local trust values.

3.2 Global Trust Values

One big problem with *local* trust values is that peer i doesn't know all peers in the network and usually only for few j the local trust is nonzero. Suppose now that peer i wants to download from peer k that we have never seen, thus $c_{ik} = 0$. A natural way for peer i to know about the trust of peer k is to ask all his *friends* (here friends means the peers with which peer i has interact) what they are knowing about peer k. But because not all friends are *trusty* it will be better to weight there opinion about peer k. This technique is called **transitive trust**. So the trust that peer i will place in peer k is defined by:

$$t_{ik} = \sum_j c_{ij} c_{jk}$$

First of all we have seen that t_{ij} is also a probability: $0 \leq t_{ik} \leq 1$ and $\sum_j t_{ij} = 1$. Thus t_{ij} is similar to c_{ij} in properties, but also a bit more powerful.

Then we can see that we can write t_{ij} in matrix notation. If $C = (c_{ij})$ is the matrix of all local trust values, then we write $\vec{t_i} = C^T \vec{c_i}$, where $\vec{t_i}$ is the vector containing all values t_{ik}.

We can generalize this notion of trust to include the opinion that friends of our friends have about peer k. Thus if we ask friend's friends peer i will have a broader view of the network. The formula is not difficult to obtain, and it looks like: $\vec{t_i} = (C^T)^2 \vec{c_i}$. Going deeper with asking we will reach a point when we have asked all peers in the network what is their opinion about peer k (there is a chain of friends). So we obtain a trust vector for peer i which looks like this: $\vec{t_i} = (C^T)^n \vec{c_i}$. Here n is the number of peers in the chain and not the number of peers in the network. n can be the number of peers in the network, but this is highly improbably since we don't have a chain of such length (if there is such a chain than the use of a *P2P* system is *useless*).

Remark 3.4 *If n is large enough, than the trust vector $\vec{t_i}$ will converge to the same vector $\vec{t}$, for all peers i:*

$$\vec{t} = (C^T)^n \vec{c_i}$$

This is a crucial point in the development of EigenTrust algorithm, because aggregating local trust values using transitive trust, we can obtain a *global* trust value. More, we know that $\vec{t_i}$ will converge from linear algebra, and we also know to what it converge: the left principal eigenvector of matrix C. Its components t_i represent the trust that all peers place in peer i.

In the rest of our paper we will show how we can compute this values. Will start with few assumptions and we will add more complex features to our algorithm as stronger assumptions are needed. But before this we will give the probabilistic idea behind all this notion of global trust.

3.3 PageRank Similarity

One can think c_{ij} as probabilities, as we saw previously. If there is an *agent* who wants to see what peer is the more reputable peer in the network, then it can *jump* from peer i to peer j with probability c_{ij}. After some numbers of jumps there will be a vector of probabilities for each peer in the system, that is the agent will be at peer i with the probability found in the i^{th} component of this vector.

This is almost what PageRank ([PageR 98]) is doing, except that here we jump with a different distribution, induced by the local trust values, versus PageRank, in which we jump

with an uniform distribution induced by the number of links that a page has. Thus our agent is a Random-Surfer.

Another difference, that we will address later, is that in PageRank we force the network matrix to be ergodic.

Finally a more closer look in [PageR 98] will show us that our global trust vector, $\vec{t}$ is the *stationary distribution* defined by the local trust matrix C.

4 EigenTrust Algorithm

In this section we show how we can compute the global trust vector $\vec{t}$ without too much computation.

First we begin with a simple setup in which we assume that there is some centralized server for computing the trust vector. Also this centralized server knows the matrix C.

4.1 Basic EigenTrust

The first thing that we can see when we want to compute $\vec{t} = (C^T)^n \vec{c_i}$ is that this equation does not depend on i, because for all i we will obtain the same vector. So we can replace $\vec{c_i}$ with any distribution one can think, and because we want when starting all peers have the same chance, we can put the uniform distribution: $\vec{t} = (C^T)^n \vec{e}$, where $e_i = \frac{1}{m}$ for all i, and m is the number of peers in the network.

Secondly, it will not be a good idea to compute the n^{th} power of the matrix C, one reason being the fact that we don't know n. But we can use the probabilistic interpretation of c_{ij} and looking at the PageRank computation we found that the following algorithm works perfectly.

Simple EigenTrust Algorithm

$\vec{t}^{(0)} = \vec{e}$, $e_i = \frac{1}{m}$

repeat

$\vec{t}^{(k+1)} = C^T \vec{t}^{(k)}$

$\delta = \|\vec{t}^{(k+1)} - \vec{t}^{(k)}\|$

until $\delta < \epsilon$

Figure 2: first attempt

There are some big drawbacks with this algorithm. The first is the lack of a prior notion of trust, then newcomers cannot gain trust (the Random-Surfer will get stuck when reaching a peer that has all $c_{ij} = 0$. Also a newcomer will not be known by the other peers). The third issue is that malicious peers can form *groups or collectivities* and in such situation the Random-Surfer can get stuck in these collectivities increasing the trust of these peers and decreasing the trust of all other peers. We address in more detail these problems:

- **Prior Notion of Trust**
  In the previous algorithm we choose to start with an uniform distribution over all peers, but this strategy doesn't hold if there can be malicious collectives , since if for example, one quarter of the network is forming such a collective, then there is a probability of $\frac{1}{4}$ to

choose them in the beginning, getting stuck later. Also is not good to choose a peer i and start with $\vec{c_i}$.

Thus we need to know a *trusty* peer to start with. A natural way to choose a peer or better **a set** of *trusty* peers is to consider the peers who first joined the network. We will call these peers **pre-trusted peers**. The reader may think that we are trying to get an excuse for introducing some special peers that look more like big authorities (servers). This assumption is not true since these peers will not have special duties. The only thing that is required is that there are peers in the network that can be trusted, and they will never change their *side*, becoming malicious peers . The peers who join first (or build) the network have no interest to subvert the network.

Thus if there are P pre-trusted peers , we can build a distribution $\vec{p}$ on them by setting:

$$p_i = \begin{cases} \frac{1}{P}, & \text{if} \quad i \in P \\ 0, & \text{otherwise} \end{cases}$$

The first way to use this distribution is by setting this value as the start value: $\vec{t} = (C^T)^n \vec{p}$. We describe other ways how this notion of pre-trusted peers can be used next.

- **Inactive Peers**
  We have seen that newcomers have in the beginning the local trust undefined, since $\sum_j \max(s_{ij}, 0) = 0$. Using pre-trusted peers we can force newcomers to trust them. So we can redefine the normalized local trust as:

$$c_{ij} = \begin{cases} \frac{\max(s_{ij},0)}{\sum_j \max(s_{ij},0)}, & \text{if} \sum_j \max(s_{ij}, 0) \neq 0 \\ p_j, & \text{otherwise} \end{cases}$$

  In this way, if a peer doesn't know any other peer, we put it, to trust the pre-trusted peers . In this manner we let newcomers to build trust (every time the Random-Surfer reach such a peer, it will jump to a pre-trusted peer).

- **Malicious Collectives**
  These collectives are groups of peers who know one each other, and each peer from these groups give all peers in these groups high local trust values and for all other peers low local trust. Thus if the Random-Surfer enters in such collectives, it will get stuck.

  Another idea from PageRank is to jump every time to another peer in the network with some probability. But we can be smarter than this. We can jump to one pre-trusted peer with some probability. This should enforce the trust that peers have in pre-trusted peers . We can see that this is simple done by weighting the trust vector in the following manner:

$$\vec{t} = (1 - a)\vec{t} + a\vec{p}$$

  Again the value a is a *magic* number ($0 < a < 1$), obtained from simulation. This convex combination helps in making the matrix C ergodic (for more details see [PageR 98] and [GoogleMat 03]).

  Thus by choosing this strategy the system should be able to *break collectives*.

Now we are ready to give a enhanced version of the algorithm from Figure 4.1. The algorithm is shown in Figure 4.1. We emphasize again the power of pre-trusted peers and that such an assumption is not against the notion of *P2P* systems. Moreover, we can search for

Basic EigenTrust Algorithm

$\vec{t}^{(0)} = \vec{p}$

repeat

$\vec{t}^{(k+1)} = (1-a)C^T\vec{t}^{(k)} + a\vec{p}$

$\delta = \|\vec{t}^{(k+1)} - \vec{t}^{(k)}\|$

until $\delta < \epsilon$

Figure 3: second attempt

algorithms that choose pre-trusted peers and not only at the beginning, but also can change them dynamically.

Next we will see a nice trick to transform the Basic EigenTrust algorithm in a distributed one.

4.2 Distributed EigenTrust

The algorithm above has one good part and one bad part. The good part is that it converges and is somehow robust to malicious peers as we wanted. Unfortunately is using a server to compute the trust vector. So apparently we have not improve in what eBay is doing (moreover we have complicated it).

But there is a way out from this, leading to a completely distributed algorithm. Before giving the main idea we emphasize that for a while we will assume that malicious peers are malicious in the sense that they only give inauthentic files and *do not lie* about what they are computing.

Looking again at the Basic EigenTrust algorithm we see that every peer can store C and $\vec{t}$, thus eliminating the need of a server. But this is way too redundant and the computation and storage will *crash* the network. Again there is a way out from this, and this looks like:

- put every peer i to store its local trust vector $\vec{c_i}$
- put every peer i to store its own global trust value t_i

We see here that the assumption about malicious peers not lying is very important.

The main idea is to write the equation

$$\vec{t}^{(k+1)} = (1-a)C^T\vec{t}^{(k)} + a\vec{p}$$

component wise. A simple calculation give us the following formula:

$$t_i = (1-a)(c_{1i}t_1 + c_{2i}t_2 + \dots c_{ni}t_n) + ap_i$$

Thus each peer need only its local trust vector and its global trust value. More important is th fact that a big part of c_{ji} from the above formula are zero, since in a real *P2P* network only very few peers have interacted with peer i, thus the computation for peer i will be not so time consuming.

This observation leads us to the algorithm from Figure 5.

Another thing worths mentioning here, and this is that pre-trusted peers remain anonymous during the computation. Its no need for other peers to know the value p_i. To see this we look

Notations:

- A_i: set of peers which have downloaded files form peer i
- B_i: set of peers from which peer i has downloaded files

Distributed EigenTrust Algorithm

foreach peer i

ask peer $j \in A_i$ for c_{ji} and ${t_j}^{(0)} = p_j$

repeat

$t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \ldots c_{ni}t_n^{(k)}) + ap_i$

send your opinion c_{ij} and trust value $t_i^{(k+1)}$ to all peers $j \in B_i$

wait for all peers $j \in A_i$, to respond with their opinion c_{ji} and trust value $t_j^{(k+1)}$

until $|t_i^{(k+1)} - t_i^{(k)}| < \epsilon$

Figure 4: distributed attempt

more careful at the algorithm. We see that p_i is appearing in two places. One is in the beginning, were peer i will ask all the peers which has transacted with about their opinion. Here we set things in such manner that only pre-trusted peers will respond and so peer i will obtain the p_i directly from them. The second case is in the computation of $t_i^{(k+1)}$ in which the term p_i appears. But we see that if peer i is not a pre-trusted peer than $p_i = 0$. So this assure us that no other peers will know the identity of pre-trusted peers .

Another strange issue is what means convergence for a distributed algorithm. In our case is not so difficult, because we know that the vector $\vec{t}$ converge, so this will do its components. But may happen that some t_i converge faster than others. This is not a problem since after a t_i has reached convergence it will stay there not modifying its value until others components will converge. The authors give more details about this. Also [GoogleMat 03] is explaining why we have this phenomena.

5 What about Security?

In the previous section, we assume some restriction on malicious peers in order to find some algorithms for computing the global trust values of peers.

The most restrictive assumption was that a malicious peer will not lie about his global trust value. With this assumption a malicious peer will not more be a malicious peer ! Unfortunately in practise we encounter malicious peers that can do more than just uploading inauthentic files. So we must consider this scenario in which malicious peers can report false trust values.

We will give only the basic ideas behind securing the Distributed EigenTrust algorithm, leaving the reader to consult [Eigen 03] for a deeper understanding of the method.

One idea is easy to see: if some peers will lie about the values they compute, than maybe is better to put another peer to compute and store one peer's trust value. We will call such peers **Score Managers**. Other idea is to put more peers to compute one peer's trust.

In this way we can eliminate malicious peers who act as a Score Manager and it will also return wrong values. Thus having more peers that compute one peer's trust, every time peer i wants its trust value, it will ask all its Score Managers for this value, and if it receives more different trust values, it will *vote* which one to choose.

Now the real problem begins. In which manner should we chose the Score Managers? How can we guarantee that there will be as few malicious peers as possible? How can we assign Score Managers for one peer, without them knowing for whom they compute?

The answer to this questions can vary allot, and one can think at different strategies for assessing the issues razed by these questions. The authors give a very nice way to solve these issues.

Thus for assigning Score Managers we will use Distributed Hash Tables. There were some nice results in this area of research in *P2P* systems and the results are very promising. The most known DHT are Chord and CAN. We will not discuss them here, but we will use some of their properties here. So the dynamics of the system will relay on the dynamics of the DHT. One property of a well design DHT is that it will spread the *keys* uniformly in the *hashed space*. Second property is that it should be almost impossible to revert a hashed value. That is: a Score Manager should not know the identity of peers for whom is computing. In this manner malicious peers are not encouraged to report wrong values, because they can report wrong values for a malicious peer decreasing its trust. Such behavior is not wanted by malicious peers. Other observation is that every peer is a Score Manager for some peer (in fact one peer is computing more trust values, for more peers).

Next we present the secure version of the algorithm seen in the previous section. Again more details can be found in [Eigen 03].

Notations:
each peer has a number M of SM — $\vec{pos}_i$ the hash mapping of peer i
D_i - the set of peers for whom i is SM — for each $d \in D_i$, c_d^i is LTV of d maintained by i

Secure EigenTrust Algorithm

foreach peer i **do**

submit local trust values to score managers

collect local trust values

submit local trust values c_{dj} to score managers

foreach daughter peer $d \in D_i$ **do**

ask peer $j \in A_d^i$ for $c_{jd}p_j$

repeat

$t_d^{(k+1)} = (1-a)(c_{1d}t_1^{(k)} + c_{2d}t_2^{(k)} + \ldots c_{nd}t_n^{(k)}) + ap_d$

send your opinion c_{dj} and trust value $t_d^{(k+1)}$ to all peers $j \in B_d^i$

wait for all peers $j \in A_i^d$, to respond with their opinion c_{jd} and trust value $t_j^{(k+1)}$

until $|t_d^{(k+1)} - t_d^{(k)}| < \epsilon$

Figure 5: Secure EigenTrust

We resume some highs that this secure algorithm shown above has:

- **anonymity:** every peer doesn't know for which peer is computing the trust value.
- **hashing:** newcomers cannot select where in the hash space they should be mapped. In this way peers cannot place themselves at the same location where their ID will be hashed.
- **redundancy:** for every peer there are more Score Managers that compute its trust.

6 Results of Simulations

We have now completed the description of the EigenTrust algorithm. We have shown that we can compute global trust values in a distributed and secure way, without imposing too many restriction on the peers.

We recall that in Section 2 we emphasized the roll of global trust in *P2P* networks. The first was that using them we can be very close to a *P2P* system where the download source is chosen randomly (system which is the best one can gets), when talking about Load Managing. Also these policies reduce the number of inauthentic files spread in the network.

The article [Eigen 03] is half full with experiments, showing that the authors intensively test their system. We will not discuss here in detail the experiments, but we will summarize some of it. The reader is encouraged to look in the article for details. One more thing that worth mentioning is that the authors simulate the experiments, thus we can think that results are biased. For seeing how the simulation is done in more details, one can look in [SimulP2P 03]

There were four scenarios discussed for achieving the performance of the algorithm in minimizing the number of inauthentic files spread in the network. A nice thing about these scenarios is that they are really worst case scenarios, thus the authors try to cover almost all cases that can happen in practice.

- **Malicious Individuals**
  In this scenario we deal with peers that *always* will provide inauthentic files. There is no interaction between these peers. It is easy to see that if these peers will replace the notion of authentic file with the notion of inauthentic file, then these peers will trust more peers which upload inauthentic files, so helping themselves.

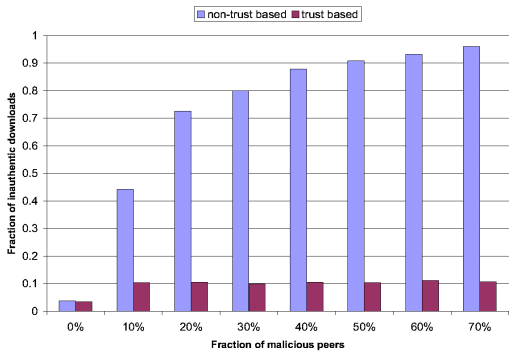

Figure 6: *Malicious Individuals*

The experiments are shown the plot in the Figure 6, and it can be seen that the system is robust for this scenario, even for a huge 70% of peers to be malicious peers .

- **Malicious Collectives**
  Other scenario is the one in which we have peers that always provide inauthentic files, but this time they know one each other. Thus the malicious peers can give one each other good opinions, and give other good peers bad feedback.

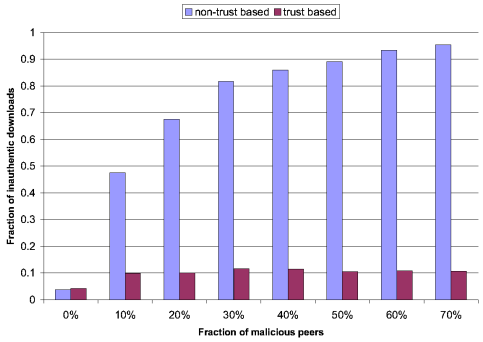

Figure 7: *Malicious Colectives*

Again the experiments give the graph in Figure 6. The system remain quite robust for 70% of the peers being malicious peers .

- **Camouflaged Collectives**
  Now we begin to assume that malicious peers are more intelligent. Thus we have some peers that provide authentic files *some of the time to trick* good peers. In this way good peers can be tricked and malicious peers can achieve good feedback.

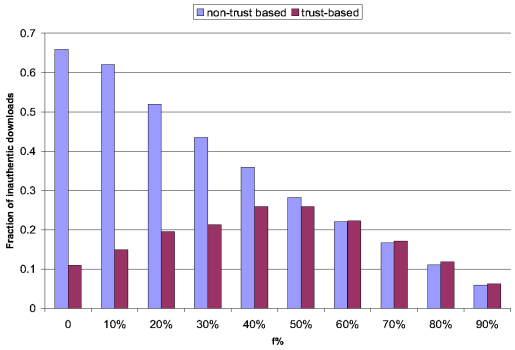

Figure 8: *Malicious Colectives*

The system begins to fall, if there are more than 40% of malicious peers in the network, that know each other. This is still a very pessimistic assumption, so we can say that the system performs well even in these conditions.

- **Malicious Spies:** In this scenario, some members of the malicious collectives give good files *all the time*, but give good feedback to malicious peers. In this way they construct a bridge between the malicious collective and the rest of the network, assuring that the Random-Surfer will have a high probability to crawl their collective. This is a more pessimistic scenario and the authors sustain that the algorithm will remain robust to this scenario for not more than 40% of the peers being malicious peers .

7 Conclusion

In this paper we have presented the EigenTrust algorithm, an algorithm who reduces the number of inauthentic files spread in the network, by isolating malicious peers from it. The assumption was that by eliminating the malicious peers from the network we will drastically reduce the numbers of inauthentic files in the network. Then we started to construct a reputation management based on the idea behind eBay. We have seen that we can compute global trust values in a distributed and secure way.

The experiments weren't so bad, and we have seen that this method really works (at least in the simulation). Also the overhead imposed by the computation was acceptable.

We don't say that this algorithm is *perfect*, but it can be the *base* of future research in the field of Trust Management for *P2P* systems.

One thing is not clear. What happens with a dynamically changing network? This issue should be addressed in most *P2P* systems. Of course that the robustness of the DHT will play an important role in this, but we should think also about the computation. Is not trivial to see what is the best strategy to choose when a peer leaves the network.

We think that this area of Trust Management will bring new enhancements in *P2P* networks, and the use of such networks will spread in more applications like *e-commerce, ad-hoc networks, etc.*

References

[Eigen 03] S. Kamvar, M. Schlosser, H. Garcia-Molina, **The EigenTrust Algorithm for Reputation Management in P2P Networks**, WWW Conference 2003

[MTrust 01] K. Aberer, Z. Despotovic, **Managing Trust in a Peer-2-Peer Information System**, ACM CIKM, New York, 2003

[PageR 98] L. Page, S. Brin, R. Motwani, T. Winograd, **PageRank Citation Ranking: Bringing Order to the Web**, Technical Report, Stanford Digital Library Technologies Project, 1998

[GoogleMat 03] T.H. Haveliwala, S. Kamvar, **The Second Eigenvalue of a Google Matrix**, Technical Report, Stanford University, 2003

[SimulP2P 03] M.T. Schlosser, S. Kamvar, **The Second Eigenvalue of a Google Matrix**, Technical Report, Stanford University, 2003