Performance of trust-based spam reduction in open communication networks

J. Krüger

December 15, 2009

Abstract

Open communication networks suffer from abusive behaviour, such as spam in Internet e-mail. Methods other than the well known statistical analysis of messages include rating systems which allow well-behaved participants to be favoured. One such system is EigenTrust, originally developed for filesharing applications. We attempt to apply EigenTrust in a communication system like e-mail, and investigate how particularly adversarial situations affect its accuracy. We suggest combining EigenTrust with a system such as Hashcash for significantly raising the barrier of message delivery for bad and unknown peers while keeping it low for known good peers.

Keywords: Communication, trust, spam

1 Introduction

Today’s communication networks are huge and, unfortunately, full of untrustworthy participants. Virtually everyone who participates in Internet e-mail has received a demonstration of this in the form of a neverending flood of unsolicited bulk mail (”spam”). The reason spamming is so prevalent is that it is easy and cheap; mail servers do not care very much about the identity of whoever drops off e-mails, and unlike in the real world, delivering several thousand e-mails costs nothing but a few seconds, so abuse is almost guaranteed.

Just like in the real world, every participant is able to deliver mails, and there is no reliable way to predict whether a given participant is going to deliver a spam message before the message is received… particularly since most participants are essentially anonymous. Armstrong and Forde[1] paint a bleak picture of the possible consequences of anonymity in networks.

Probably the most well-known measure against spam is statistical content analysis, i.e. software which analyzes incoming e-mails and classifies them as spam or non-spam, based on previous training. This approach has been shown to be highly effective in practice[9]. One might argue, however, that content analysis is costly and other techniques, if feasible, should be used to slow the flood of spam messages, thereby reducing the need for content analysis.

Most approaches to solving this problem depend on the concept of ‘trustworthiness’, i.e. a method of allowing participants to establish virtual identities and associate other participants’ identities with an estimate of how likely it is that someone is going to be well-behaved (and will send only acceptable messages). Ideally, however, it should be possible for participants who trust each other to some degree to share their opinions of other participants, so that they can predict the behaviour of ‘acquaintances of acquaintances’ without having any direct experience with that third party. There are two major problems with this goal: firstly, communicating such information in a huge network either incurs a lot of transmission overhead or needs a central “trust provider” that is ultimately trusted by all other participants and has massive computational resources; secondly, a delicate balance between reusability and reliability of others’ trust information has to be maintained, which is particularly difficult considering that a large number of collaborating evildoers (who will naturally vouch for each other) is to be expected.

Since a sizeable subset of the overall network will consist of “bad peers” who will send spam messages and give incorrect trust ratings to help one another defeat the trust mechanism, one challenge is to fortify the algorithm against cliques of bad peers. At the same time, newcomers to the network should be able to participate without undue difficulty; otherwise it is very unlikely that the network will be widely adopted.

EigenTrust[5] is a trust system originally designed for peer-to-peer filesharing. It has received a fair amount of academic attention and is said to attain good results, making it a good choice for testing trust-based evaluation of communication peers.

As already mentioned, the proportion of bad peers...
is expected to be large. In fact, the so-called Sybil attack\cite{Sybil}, made possible due to today’s huge networks of compromised computers (usually called botnets)\cite{Botnets}, works by distributing the sending of spam in a manner so that any given recipient rarely sees two spam messages originating from the same source address. Clearly, unknown peers should not be trusted significantly more than known bad peers, but this misses the question of how to hinder bad peers while allowing new good peers to join the network. The authors of EigenTrust suggest imposing a cost on joining the network which reduces the ratio of bad peers joining the network.

The original EigenTrust with its focus on filesharing cannot necessarily be applied directly to e-mail due to the different structure of transactions in filesharing and e-mail. In filesharing, a transaction happens like this:

1. A peer requests a file from other peers.
2. Other peers send parts of the file.
3. The receiver validates the received parts through a separate mechanism and rates all senders accordingly.

On the other hand, e-mail works more like this:

1. A peer receives a large number of mails from other peers. Some of the mails are solicited, others (probably the majority) are not.
2. The peer rates senders depending on whether they sent a good or a bad message.

It is much more reasonable to expect of a peer to rate every single transaction if it initiated the transaction (as is the case in the filesharing scenario) than to expect a peer to rate every transaction if those transactions are forced on it (as is the case in the e-mail scenario).

In this thesis we analyze the performance of EigenTrust in scenarios in which there is a significantly lower motivation for peers to rate other peers. Most importantly, we address the question of how the size of the communication network and the number (and accuracy) of ratings in the network affect the accuracy of the algorithm.

After briefly explaining the EigenTrust algorithm in section 2, we present a model of communication in section 3 that we will use to perform experiments in section 4. We discuss the implications of the results in section 5.

## 2 EigenTrust in a nutshell

Given a communication network of $n$ peers, each peer rates messages received from other peers. Each rating is either positive (legitimate e-mail, often called “ham”) or negative (spam). If $spam(i,j)$ is the number of messages peer $i$ received from $j$ that it rated as spam, and $ham(i,j)$ similarly denotes the number of messages rated as ham, the overall rating of a peer can be simply written as the matrix

$$S_{ij} = ham(i,j) - spam(i,j)$$

Generally, a peer who sends ham messages receives a positive score and a peer who sends spam messages receives a negative score.

To prevent peers from giving their friends arbitrarily high scores (which is undesirable especially for malicious peers), each peer’s ratings are normalized:

$$C_{ij} = \frac{\max(S_{ij},0)}{\sum_k \max(S_{ik},0)}$$

This has the side effect of discarding negative ratings, which is in line with our goal of giving new peers the same rating as known malicious peers.

A core principle of EigenTrust is that of pre-trusted peers. These are peers of which it is known in advance that they are completely trustworthy. Often, the founders of a communication network qualify for this role.

The EigenTrust algorithm involves iteratively multiplying this normalized trust matrix with a vector, additionally biasing each iteration towards the perspective of the pre-trusted peers. Therefore, the global trust values of all peers, written as the vector $\vec{t}$, are calculated as:

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

where $a$ is the bias factor and $\vec{p}$ is a normalized vector assigning equal trust to all pre-trusted peers $P = \{p_1, \ldots, p_k\}$:

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

Intuitively, if the normalized trust matrix is treated as an adjacency matrix of a weighted digraph of the communication network, each iteration corresponds to asking the neighbors of all previously asked peers about their opinions of everyone, weighted by the level of trust that has been calculated for them.

The biasing component $a\vec{p}$ ensures that the iteration converges, no matter which initial vector is used, to a global trust vector that is influenced much more strongly by the trust values from pre-trusted peers and their circle of friends then by the incorrect trust values as given by the bad peers.

As a good starting value for the iteration, the authors of EigenTrust suggest $\vec{t}^{(0)} = \vec{p}$; no specific value is suggested for the bias factor $a$. 

---

\cite{v. December 15, 2009, p.2}
2.1 Implementation

Since the algorithm involves a series of costly matrix multiplications (despite the high sparsity of the matrix in typical scenarios, the algorithm's running time complexity as well as space complexity is still obviously polynomial), it is desirable to make the number of iterations adaptive. We follow the popular approach of iterating until

\[ \| \bar{t}^{(k+1)} - \bar{t}^{(k)} \| < \varepsilon \]  

where \( \varepsilon \) is a minimum error value. We choose \( \varepsilon = 0.0001 \).

3 A pessimistic model of communication

Our analysis of the performance of EigenTrust is based on the idea that optimism, while perhaps a good attitude towards life, developing systems for use in near-ideal conditions is almost certain to result in something useless. With that in mind, we present a model that offers a balanced blend of simplicity and pessimism.

The main compromise favouring simplicity at the expense of pessimism is that we assume that bad peers never send ham messages. This is somewhat realistic since it is very difficult to create a message that a large number of random peers consider ham, but it excludes certain attack scenarios such as "corrupted" good peers who send only ham messages but secretly rate bad peers positively.

Thanks to this compromise, we can practically discard the values of \( \text{spam}(i,j) \) from our model; after all, bad peers will receive almost no positive ratings at all, resulting in a normalized local trust value of 0 for them from virtually all good peers.

With these points in mind, the model is constructed like this:

1. All bad peers rate all other bad peers positively. This is the worst case in which all bad peers cooperate.
2. Good peers rate each other with a certain (low) probability \( p_e \) (we call this the rating density). This probability is a simplified version of independently observing two variables: the probability that two peers communicate at all and the probability that each such piece of communication gets rated (correctly).
3. If a good peer rates another good peer, the unnormalized rating is \( k \) with a probability of \( p_m \). This models the inevitable situation in which good peers don't give all other good peers the same amount of positive ratings. We call \( p_m \) the follow-up probability.
4. Sometimes good peers accidentally rate bad peers positively; the exact model for this is explained in section 4.3.

For most somewhat realistic values of these parameters, there will be many more wrong ratings (particularly the ratings by bad peers for other bad peers) than correct ratings, making the situation hopeless for an algorithm that does not use pre-trusted peers or some other way of choosing a good starting vector.

Ignoring the follow-up probability, the graph is constructed using the \( G(n,p) \) model of random graphs by Erdős and Rényi[4].

3.1 Accuracy

We define the accuracy of EigenTrust in our model as the ratio of correctly identified good and bad peers. Given a global trust vector \( \bar{t} \) as calculated by EigenTrust, and a vector \( \bar{g} \) with

\[ g_i = \begin{cases} 0 & \text{if peer } i \text{ is bad} \\ 1 & \text{if peer } i \text{ is good} \end{cases} \]

we take \( m \) as the smallest component value in \( \bar{t} \) and construct the binary trust vector \( \bar{t}' \) which has each component set to one if the corresponding component in \( \bar{t} \) is greater than \( m \) and set to zero otherwise.

At first sight this may look complicated, but the reasoning is simple: while the global trust values for bad peers will probably converge to zero even if some good peers make rating mistakes, we do not have the luxury of iterating until the algorithm converges exactly; therefore we assume that the smallest component value in the vector is what would converge to zero if we continued the iteration. It turns out that all of the bad peers have this global trust, except in extremely adversarial situations.

Having constructed \( \bar{t}' \), the accuracy can simply be calculated as

\[ \frac{n}{n} - \frac{\| \bar{t}' - \bar{g} \|_2}{n} \]

where \( n \) is the number of peers.

4 Some numerical experiments

Generally, we are interested in the effects of the following parameters on the accuracy of the algorithm:

- increasing the rating density,
- increasing the number of peers in the network,
- varying the proportion of bad peers in the network,
• varying the number of pre-trusted peers in the network,
• introducing an error ratio,
• varying the bias factor.

The follow-up probability only serves as a simple way to make the simulation slightly more realistic: consequently, we fix this parameter at the arbitrary value $p_m = \frac{1}{2}$.

In all of the following experiments, each result is the average of five repetitions of the same simulation.

### 4.1 Reducing the parameter space

At this point, we have a six-dimensional space of experimental parameters to explore. For significantly reducing the set of parameters that can be varied, a few rounds of preliminary experiments can show which parameters do not have a significant impact on the results.

In the following preliminary experiments, unless otherwise specified, the examples use the following values:

- Number of peers: 200
- Number of pre-trusted peers: 2
- Proportion of bad peers: 80%

**Bias factor**

![Figure 1: Accuracy for different bias factors](image)

Figure 1 shows the effects of varying the bias factor in a typical situation. It is representative of the many other cases in which changing the bias factor had a similar effect. It is particularly interesting in combination with figure 2, which shows that the main effect of the bias factor is on the average number of iterations according to our convergence criterion from equation (2).

As can be seen in this example, a higher bias factor tends to reduce the number of iterations required. On the other hand, although there seems to be no significant effect on the accuracy in figure 1, we expect from the way the bias factor works that too high values for $a$ might

![Figure 2: Number of iterations for different bias factors](image)

unduly reduce the effect of ratings from peers other than the pre-trusted ones, so we use $a = 0.5$ which seems like a good overall compromise and hence will be used in all further experiments.

**Pre-trusted peers**

![Figure 3: Accuracy for different numbers of pre-trusted peers: 2, 5, 10, 20](image)

As can be seen in figure 3, an increasing number of pre-trusted peers slightly improves the accuracy, both in better averages and lower standard deviations. A simple explanation for this phenomenon is that the more pre-trusted peers exist, the higher the chance that the collective of pre-trusted peers is connected to all good peers.

Since this parameter does not significantly influence the results, we refrain from varying it systematically; instead, we slightly increase it whenever we look at very low rating densities to try to prevent disastrously incorrect results.

### 4.2 Simulating without errors

Initially we look at experiments in which good peers never give bad peers a positive rating. This makes it fairly easy for the algorithm to succeed because there is
no “path of ratings” from any pre-trusted peer to any bad peer.

Due to this, when simulating without errors, there is actually no difference in results between a network with 1000 peers, 50 of which are good, and a network with 50 peers, all of which are good. This saves us a lot of computation effort, and we can simply investigate how rating density and number of peers interact, independent of the number of bad peers in the network.

Figure 4: Average accuracy for rating density up to 0.05, without errors: 100, 200, 500 and 1000 good peers

Figure 4 shows that an increasing number of peers significantly improves accuracy; the situation with 1000 peers converges roughly ten times as quickly (as rating density increases) as the situation with 100 peers does. This result suggests that as the network grows infinitely large, any non-zero rating density will result in perfectly accurate results.

Figure 5: Average accuracy for rating density up to 0.01, without errors: 500, 1000, 2000 and 5000 good peers

This idea is supported by taking a closer look at smaller scales, as seen in figure 5. It shows about the same structure as before, just on a smaller scale. With more computing power available to simulate larger instances, we suspect that the same thing would happen again, leading us to

Conjecture 1. With increasing number of good peers in a communication network, smaller rating densities are sufficient for very good performance of EigenTrust iff good peers always rate bad peers correctly.

If we go by the (very rough) estimate that making a network ten times bigger gets the same level of accuracy with a tenth of the rating density, we can predict that a network of ten million participants would get excellent results even at a rating density of $10^{-6}$.

Figure 6: Number of iterations for rating density up to 0.05, without errors: 100, 200, 500 and 1000 good peers

Figure 7: Number of iterations for rating density up to 0.01 without errors: 500, 1000, 2000 and 5000 good peers

Looking at the number of iterations gives us a similar picture: figure 6 shows the first scale and figure 7 shows the smaller scale.

4.3 Simulating with errors

Unfortunately, in the real world people make mistakes and this complicates matters. We now look at what happens when a small number of peers makes a small number of mistakes.
We assume that only a certain small percentage of good peers make mistakes, and only a certain number of mistakes. Formally, given the set $G$ of all good peers, $F = \{f_1, \ldots, f_k\} \subset G$ with $P(f_i) = p_f$ is the set of good peers who make mistakes, which we call false peers (and we call $p_f$ the false peer ratio). Each false peer rates exactly $f$ randomly chosen bad peers positively (this simple model of mistakes is inspired by a saying that goes a bit like “fool me five times, shame on you; fool me six times, shame on me”). Furthermore, to avoid sabotaging the algorithm too much, the false peers are chosen so that no false peer is also a pre-trusted peer.

Experimental results show quite similar convergence behaviour as in the simulations without errors, so we will forego providing more graphs of convergence behaviour.

**Various proportions of bad peers**
In each quartet of graphs, from left to right and top to bottom, we show the results for 100, 200, 500 and 1000 peers.

Figures 8, 9 and 10 show how EigenTrust performs with increasing proportions of bad peers at unchanged false peer ratio and false ratings. Unsurprisingly, standard deviations increase as more bad peers are introduced, but once again for large numbers of peers the difference is negligible.
Higher false peer ratios
Although the previously used false peer ratio of 1 % with 5 false ratings each seemed reasonably pessimistic, it remains to be seen if even more pessimistic (and probably no less realistic) scenarios are still feasible with EigenTrust. For this purpose, we use a bad peer proportion of 80 % in a network of 1000 peers and 5 false ratings each for a varying ratio of false peers.

Figure 11: Accuracy for varying false peer ratio: 0.02, 0.05, 0.1 and 0.2

Figure 11 shows the results. They are interesting in that while the graph has the shape we’ve already seen in almost all other results, it seems to max out at lower values the more the false peer ratio increases. It seems reasonable to expect that the graph will continue more or less parallel to the x axis.

Indeed, figure 12 suggests that at a false peer ratio of 20 %, accuracy suffers dramatically for a network of 1000 peers.

In this case, even doubling the total number of peers does not help things, as can be seen from figure 13.

The overall structure of the curves is just as interesting as the max out; figure 14 shows a close-up of the left half of the fourth graph in figure 11. We will discuss this structure in section 5.

5 Discussion of the results
Our experiments have shown that EigenTrust can deal with almost any adversity; increasing the number of peers in the network almost invariably improves its overall accuracy, no matter if the adversity consists of:

• an increasing ratio of bad peers;
• a decreasing rating density;
• decreasing the number of pre-trusted peers.

Once the false peer ratio gets relatively high, however, EigenTrust rapidly approaches a performance no better than random guessing.

It is probably impossible to determine exactly how difficult it is for the average (and less than average) user of such a communication network to correctly recognize (and deal with) bad peers and messages received from them. Despite the pessimistic model used in our experiments, we are still optimistic in that we believe that...
these results support the usefulness of EigenTrust in a communication setting as outlined in this thesis.

We observe interesting effects once a non-zero false peer ratio gets introduced. The initial results with no false peers produced a curve with one sudden jump (which is reminiscent of phase transitions in, for example, percolation problems[7], but the rating density at which this jump occurs appears to vary depending on the size of the network). Once false peers are introduced, however, two such jumps appear. For example, in figure 14 we see one such threshold at about 0.005 and another much less clear threshold at about 0.025. There seems to be a delicate interaction between three variables (rating density, error rate and accuracy) that is very sensitive to subtle details in the model. Even our extremely simple model of false peers causes these threshold which seem hard to explain.

We assume that the curve can be understood roughly as follows: below the first threshold, the probability that the pre-trusted peers can “reach” information about other peers along “trust edges” is so low that it is extremely likely that the pre-trusted peers will only encounter “trust paths” that lead to false peers, therefore getting exactly the opposite result to what would be correct (at least in our model in which bad peers rate everyone incorrectly). As the density ratio “catches up” with the false peer ratio, it might be that correct ratings and incorrect ratings start mixing together so that we get a medium accuracy… at least in relation to the final accuracy, which starts appearing once the correct ratings start dominating the incorrect ratings.

Considering our previous conjecture about a general relation between network size and density ratio in section 4.2, we expect that in practical situations, networks will be so large that the density ratio will be high enough to make the accuracy below the second threshold largely irrelevant.

5.1 Applications

One can never mention often enough that it is very unlikely that a given technique will completely eliminate spam: the human error and manipulative schemes employed by bad peers invariably work against even the most intricate technical designs. This raises the question of how EigenTrust can actually be used to reduce spam without locking out new or unknown peers.

One solution is Hashcash[2], a technique to easily and efficiently generate a challenge that is computationally difficult to solve (and exactly how difficult it is can be decided by the challenger). Using Hashcash, a receiver of a message can determine the sender’s global trust value and use it to send the message sender a Hashcash challenge, the difficulty of which depends on how well the sender is trusted. This forces unknown or bad peers to expend a high amount of computational power for delivering a message, but known good peers pay significantly less computation time, to the point where the challenge gets so insignificant that it can be omitted altogether.

Hashcash was proposed as a trust-independent method of introducing a cost on sending messages; Laurie and Clayton[6] have argued that in this case, a sufficiently costly computation would restrict good peers too much. By combining Hashcash with a mechanism like EigenTrust to force much lower computational costs on good peers, the impact of this limitation is significantly reduced, if not eliminated.

6 Conclusion and outlook

The models constructed and the data collected suggest that EigenTrust is a viable system for creating an open, unrestricted communication network that significantly reduces spam, for example by combining it with Hashcash; as long as peers who send good messages do not rate too many bad peers positively.

Further research might investigate how this approach could be improved by tuning EigenTrust or combining it with other trust and ranking algorithms. In the context of filesharing, Donato et al[3] have shown that combining EigenTrust with Truncated PageRank (a technique based on Google’s website ranking algorithm), Bit Propagation and BadRank (a technique that specifically rates the “badness” of peers) produces better results than EigenTrust alone. It remains to be seen if the same thing can translate to other scenarios.

Additionally, we have not thoroughly investigated a specific method of combining Hashcash and EigenTrust (as suggested in section 5). A naive approach might be to scale the baseline computational cost demanded from peers by the inverse of a given peer’s trust value, but it remains to be seen if this approach would be effective enough.

Lastly, the experimental framework we used in this thesis had difficulties with handling networks of more than 2000 peers. While it is quite likely that the storage and time efficiency of the algorithm could be improved significantly, it seems unlikely that EigenTrust will scale to networks of several million peers, such as the communication network made up of all Internet Mail servers.

It might be interesting to develop a restricted version of EigenTrust that has only peers of a certain maximum distance (according to some arbitrary distance measure) exchange trust information.

References

Performance of trust-based spam reduction in open communication networks

J. Krüger


