The EigenTrust Algorithm for Reputation Management in P2P Networks

Presented By
Asim Sinan YUKSEL

What is Eigentrust?

- A reputation-based trust management system.
- Aims to minimize malicious behavior in a peer-to-peer network.
- Computes the agents' trust scores through repeated and iterative multiplication.
- Aggregates trust scores along transitive chains until the trust scores for all agent members of the P2P community converge to stable values.

Problem, Goal, Method

- Problem: Inauthentic files distributed by malicious peers on a P2P network.
- Goal: Identify sources of inauthentic files and bias peers against downloading from them.
- Method: Give each peer a *trust value* based on its previous behavior.

Some Definitions

- **Local trust value: c_{ij}:** The opinion that peer i has of peer j, based on past experience.
  - Each time peer i downloads an authentic file from peer j, c_{ij} increases.
  - Each time peer i downloads an inauthentic file from peer j, c_{ij} decreases.
  - All c_{ij} non-negative
  - $c_{i1} + c_{i2} + . . . + c_{in} = 1$
  - Local trust vector contains all local trust values c_{ij} that peer i has of other peers j.
- **Global trust value: t_i:** The trust that the entire system places in peer i.

Some Approaches

- Past History
- Friends of Friends
- EigenTrust

Past History

- Each peer biases its choice of downloads using its own opinion vector $\mathbf{c}_i$.
- If it has had good past experience with peer j, it will more likely download from that peer.

Friends of Friends

- Ask for the opinions of the people who you trust.
- Weight their opinions by your trust in them.

$$c'_{ik} = \sum_j c_{ij} \cdot c_{jk}$$

What they think of peer k.

Ask your friends j

And weight each friend's opinion by how much you trust him.

$$\mathbf{c}'_{\mathbf{i}} = \mathbf{C}^{\mathbf{T}} \mathbf{c}_{\mathbf{i}}$$

Problem With Friends

- If you know a lot of friends, you have to compute and store many values.
- If you have few friends, you won't know many peers.

Applying Both Approaches

Know All Peers

- Ask your friends: $t=C^Tc_i$.
- Ask their friends: $t=(C^T)^2c_i$.
- Keep asking through all friends: $t=(C^T)^nc_i$.

Minimal Computation

- *Trust vector* **t** converges to the same thing for every peer.
- Each peer doesn't have to store and compute its own trust vector. The whole network can cooperate to store and compute **t**.

Non-Distributed Algorithm

$\vec{t}^{(0)} = \vec{e};$

repeat

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

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

until $\delta < \epsilon;$

$e_i = 1/m$

Algorithm 1: Simple non-distributed EigenTrust algorithm

e vector: the m-vector representing a uniform probability distribution over all m peers

Distributed Algorithm

- No central authority to store and compute **t**.
- Each peer i holds its own opinions $\mathbf{c_i}$.

For each peer *i* {
  -First, ask peers who know you for their opinions of you.
  -Repeat until convergence {
    -Compute current trust value: $t_i^{(k+1)} = c_{1j}\, t_1^{(k)} + \ldots + c_{nj}\, t_n^{(k)}$
    -Send your opinion c_{ij} and trust value $t_i^{(k)}$ to your acquaintances.
    -Wait for the peers who know you to send you their trust values and opinions.
  }
}

Secure Score Management

- Instead of having a peer compute and store its own score, have *another* peer compute and store its score.
- Have multiple score managers who vote on a peer's score.

How to use the trust values t_i

- When you get responses from multiple peers:
  - Deterministic: Choose the one with highest trust value.
  - Probabilistic: Choose a peer with probability proportional to its trust value.

Some Threat Scenarios

- **Malicious Individuals**
  - Always provide inauthentic files.
- **Malicious Collective**
  - Always provide inauthentic files.
  - Know each other. Give each other good opinions, and give other peers bad opinions.
- **Camouflaged Collective**
  - Provide authentic files some of the time to trick good peers into giving them good opinions.
- **Malicious Spies**
  - Some members of the collective give good files all the time, but give good opinions to malicious peers.

Conclusion

Strengths:

- Reduces number of inauthentic files on the network.
- Robust to malicious peers.
- Low overhead.

Weaknesses

- No means of measuring negative trust.
- May punish peers inside college networks. Because college network as a whole consumes by downloading much more than it uploads.


Thank You!