The EigenTrust Algorithm for Reputation Management in P2P Networks

Sepandar D. Kamvar

Mario T. Schlosser

Hector Garcia-Molina

Stanford University

P2P Networks

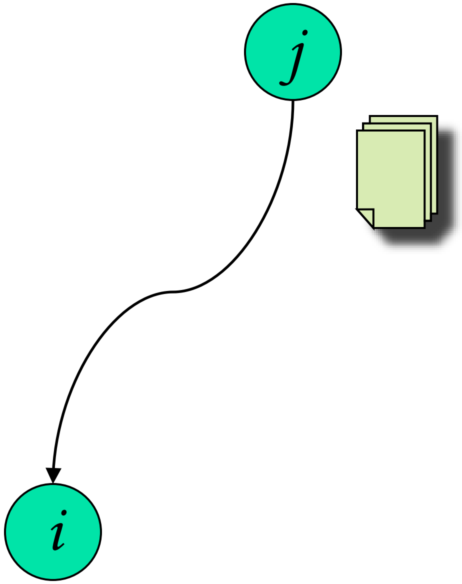
- Open and anonymous
  - Benefits
    - Robust, Scalable, Diverse
  - Problems
    - Malicious peers
    - Inauthentic files
      - Viruses/Malware
      - Tampered files

Identifying malicious peers is more pressing than identifying inauthentic files

Reputation Systems

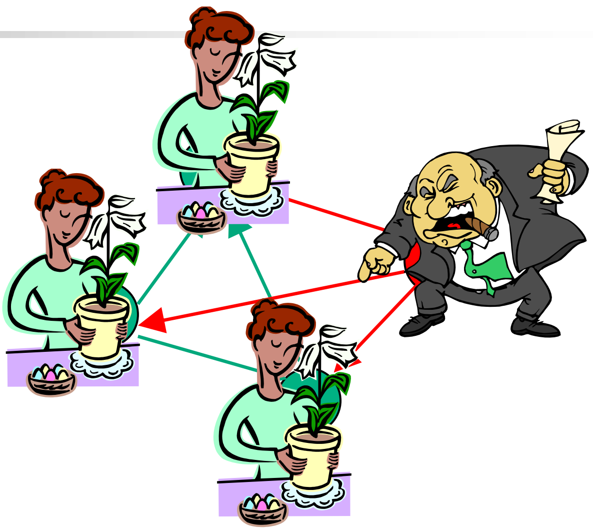
- Reputation Systems
  - *Global*: Centralized system (eBay)
  - *Local*: Distributed System

- *Key Idea of EigenTrust:* Each peer *i* is assigned a *global* trust value that reflects the *local* experiences of all the peers in the network



Problem

- **Problem:**
  - Reduce inauthentic files distributed by malicious peers on a P2P network.



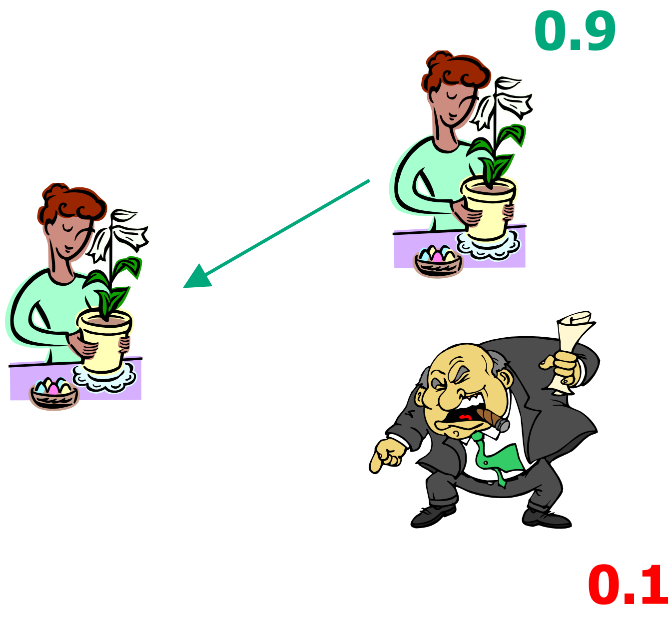
- **Motivation:**

"Major record labels have launched an aggressive new guerrilla assault on the underground music networks, flooding online swapping services with bogus copies of popular songs."

-Silicon Valley Weekly

Problem

- **Goal:** To identify sources of inauthentic files and bias peers against downloading from them.
- **Method:** Give each peer a *trust value* based on its previous behavior.

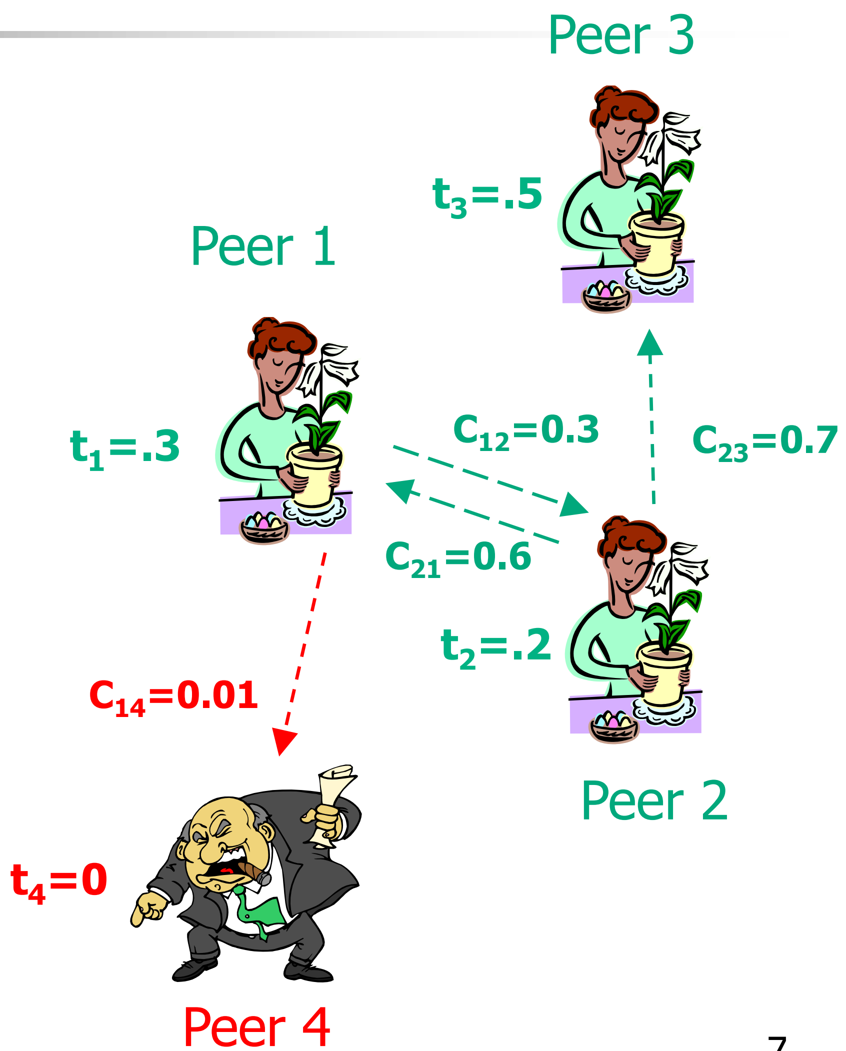


Some approaches

- Past History
- Friends of Friends
- EigenTrust

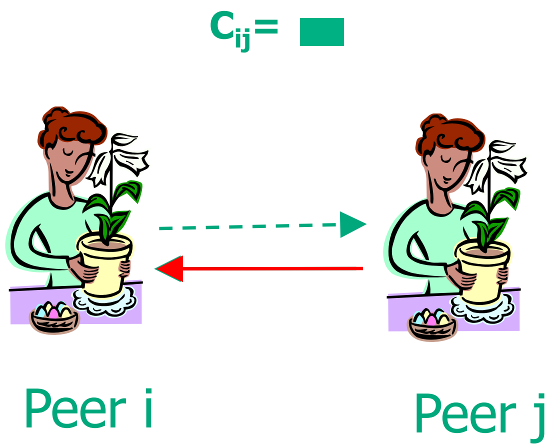
Terminology

- **Local trust value: c_{ij}.** The opinion that peer i has of peer j, based on past experience.
- **Global trust value: t_i.** The trust that the entire system places in peer i.



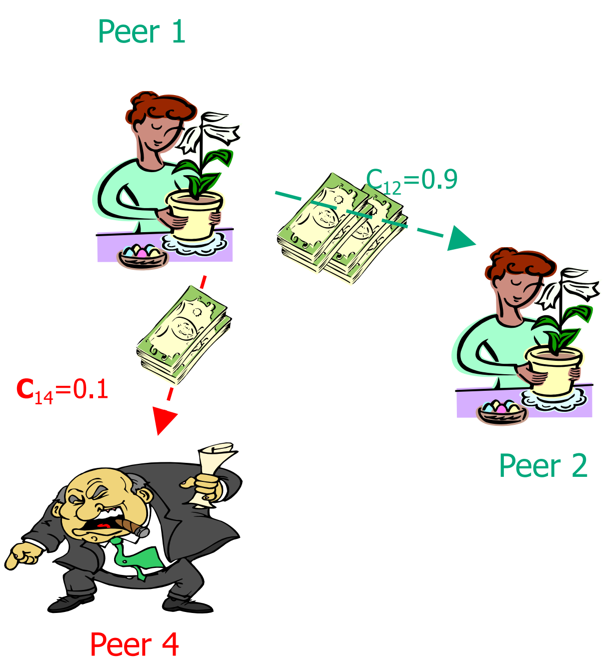
Local Trust Values

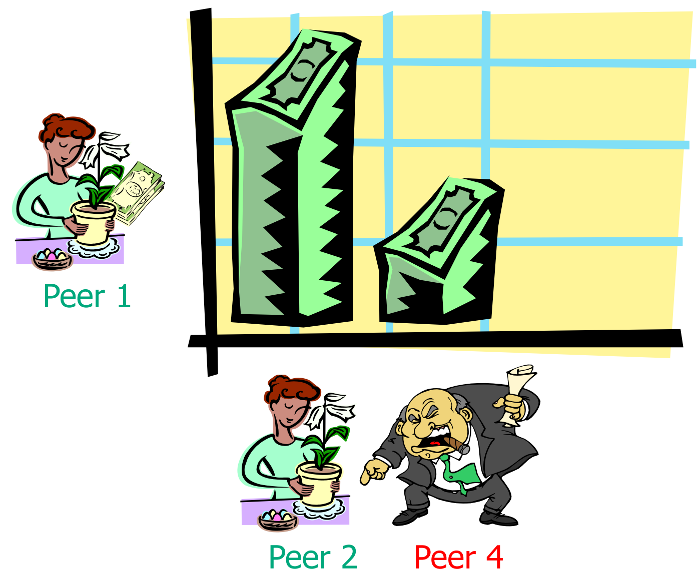
- Each time peer i downloads an authentic file from peer j, c_{ij} increases.
- Each time peer i downloads an inauthentic file from peer j, c_{ij} decreases.



Normalizing Local Trust Values

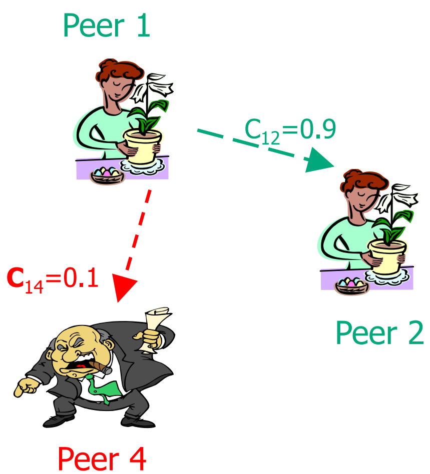
- All c_{ij} non-negative
- $c_{i1} + c_{i2} + \ldots + c_{in} = 1$





Local Trust Vector

- **Local trust vector $\mathbf{c}_i$:** contains all local trust values c_{ij} that peer i has of other peers j.



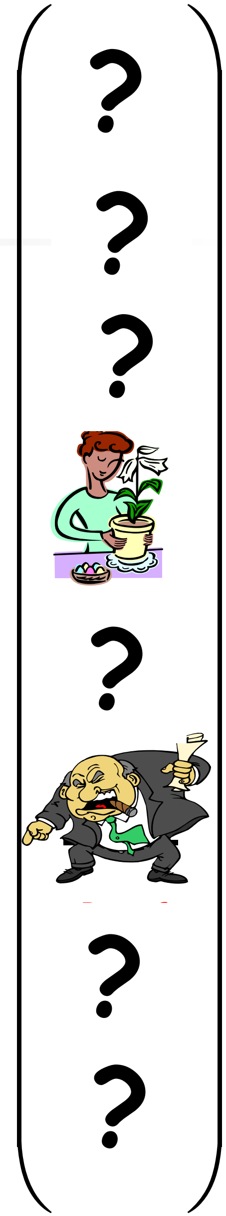
$$= \begin{pmatrix} 0 \\ 0.9 \\ 0 \\ 0.1 \end{pmatrix}$$

$\mathbf{c}_1$

Approach 1: Past history

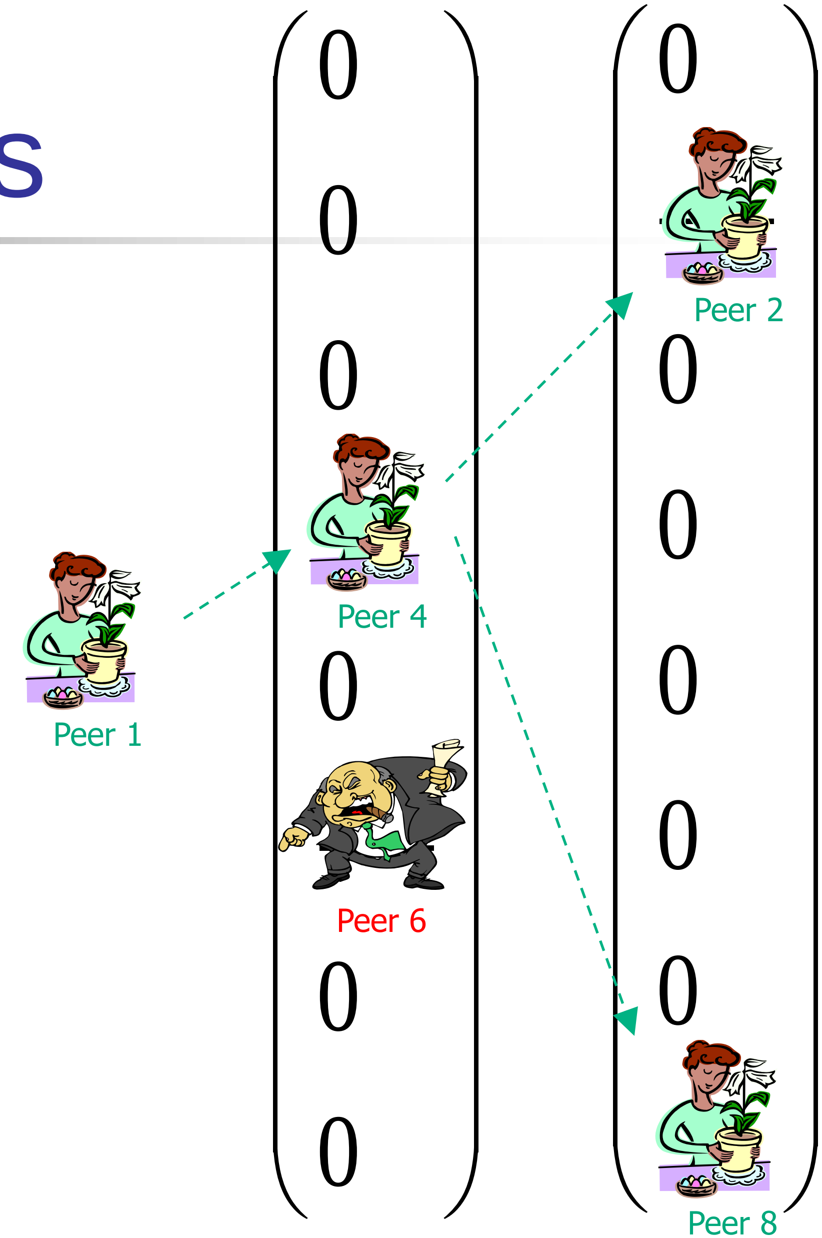
- Each peer biases its choice of downloads using its own opinion vector $\mathbf{c}_i$.
- If it has had good past experience with peer j, it will be more likely to download from that peer.
- **Problem**: Each peer has limited past experience. Knows few other peers.

Peer 1



Approach 2: Friends of Friends

- Ask for the opinions of the people who you trust.



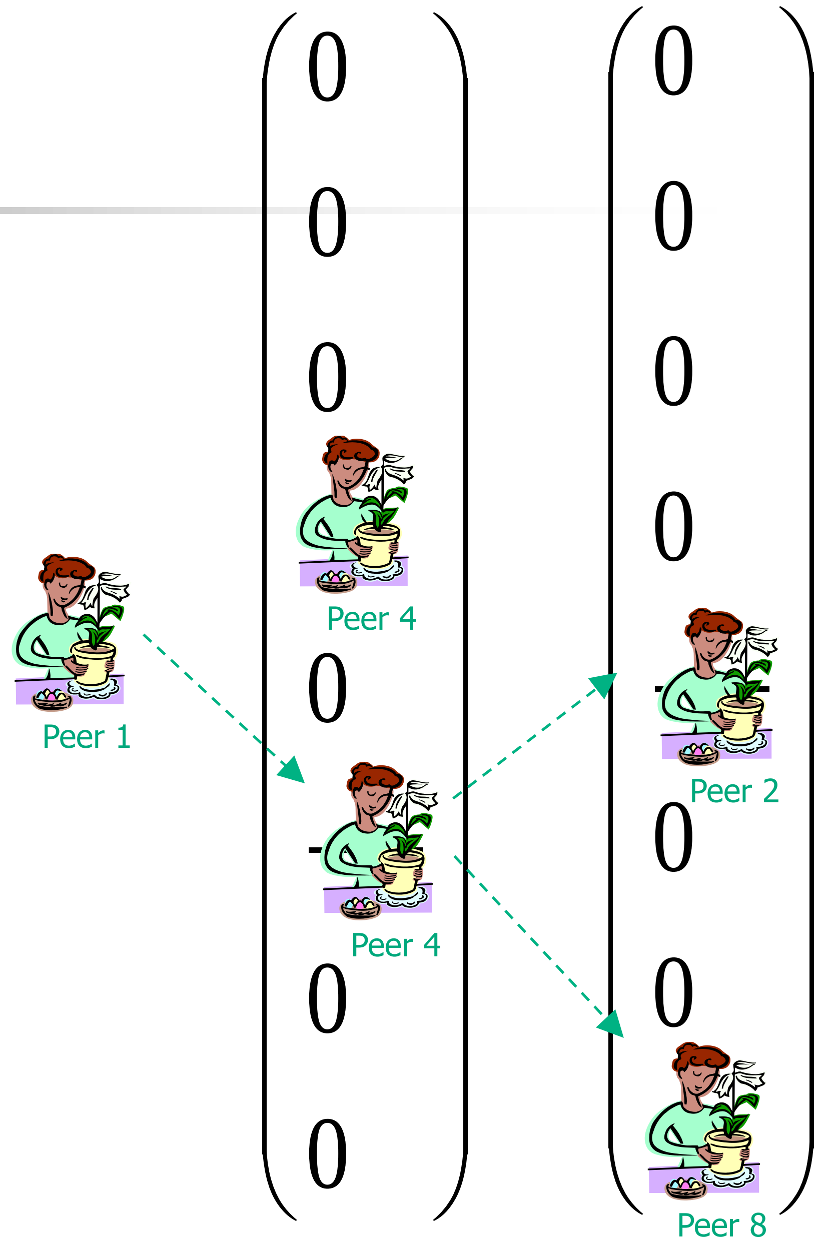
(Cf. Referral trust)

Friends of Friends

- Weight their opinions by your trust in them.

(Cf. Referral trust = Functional trust)

(Cf. Transitivity)



The Math : Transitive Trust

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

What they think of peer k.

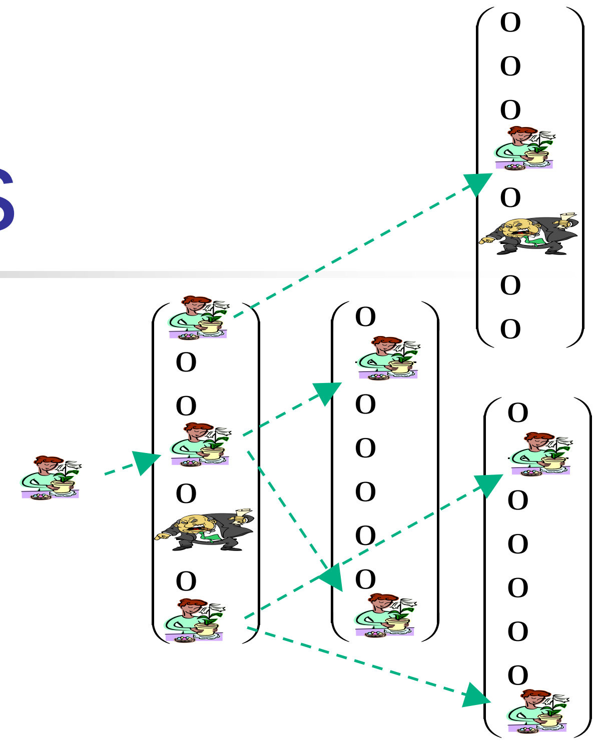
Ask your friends j

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

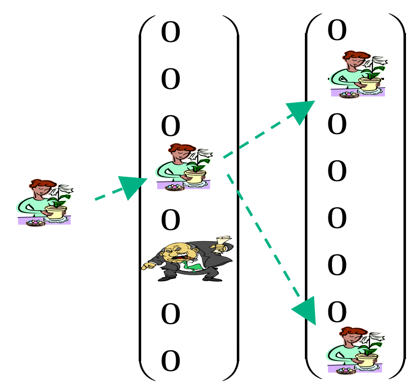
$$\begin{bmatrix}.1\\.3\\.2\\.3\\.1\\.1\end{bmatrix} = \begin{bmatrix} 0 & .2 & 0 & .3 & 0 & .5 & .1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix}.1\\.5\\0\\0\\0\\.2\end{bmatrix}$$

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

Problem with Friends



- Either you know a lot of friends, in which case, you have to communicate, compute and store many values.
- Or, you have few friends, in which case you won't know many peers, even after asking your friends.

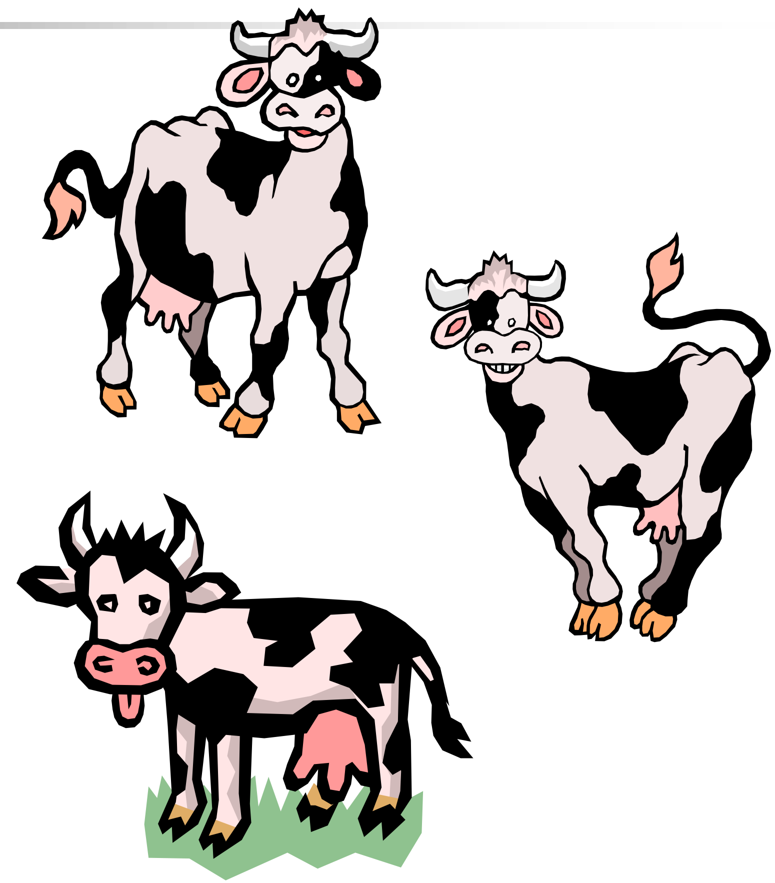


Eigen Trust: Dual Goal

- We want each peer to:
  - Know all peers.
  - Perform minimal computation (and storage).

Knowing All Peers

- Ask your friends: $t = C^T c_i$.
- Ask their friends: $t = (C^T)^2 c_i$.
- Keep asking until the cows come home: $t = (C^T)^n c_i$.



Minimal Computation

- Luckily, the *trust vector* **t**, if computed in this manner, converges to the same thing for every peer!
- Therefore, 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

- Initialize:

$$\mathbf{t}^{(0)} = \left[\frac{1}{n} \quad \cdots \quad \frac{1}{n} \right]^{\mathbf{T}}$$

- Repeat until convergence:

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

Basic EigenTrust Algorithm

- Assumption: include central server at this stage
  - A server stores all the c_{ij} values and performs the computation

$\vec{t}^{(0)} = \vec{e};$ $\quad e_i = 1/n$

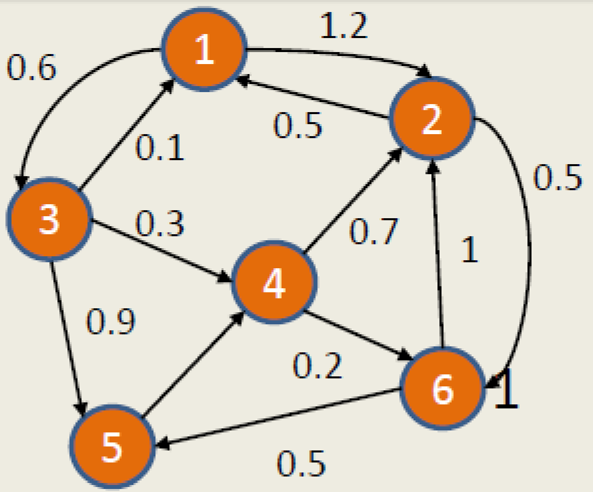
repeat

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

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

until $\delta < \epsilon;$

An Illustration Example of EigenTrust



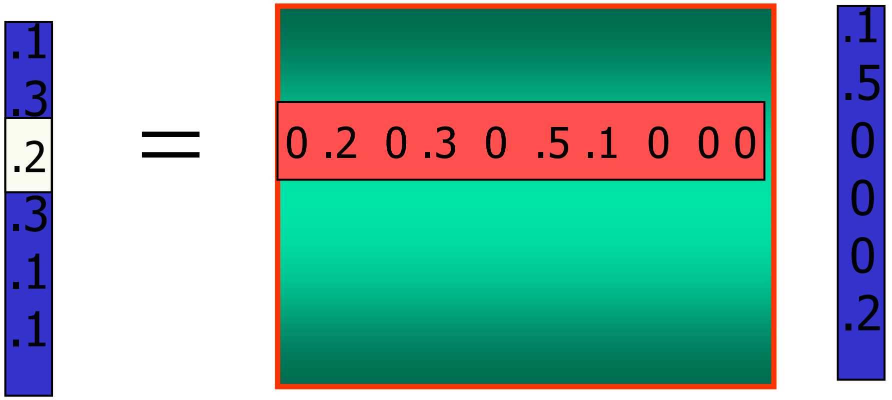
Normalization

$$C = \begin{matrix} 0 & 0.67 & 0.33 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \\ 0.08 & 0 & 0 & 0.23 & 0.69 & 0 \\ 0 & 0.78 & 0 & 0 & 0 & 0.22 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0.67 & 0 & 0 & 0.33 & 0 \end{matrix}$$

$$t^0 = \begin{matrix} 0.1667 \\ 0.1667 \\ 0.1667 \\ 0.1667 \\ 0.1667 \\ 0.1667 \end{matrix} \qquad t^1 = C^T t^0 = \begin{matrix} 0.0967 \\ 0.3534 \\ 0.0550 \\ 0.2050 \\ 0.1700 \\ 0.1200 \end{matrix} \qquad t^2 = C^T t^1 = \begin{matrix} 0.1811 \\ 0.3051 \\ 0.0319 \\ 0.1827 \\ 0.0776 \\ 0.2218 \end{matrix} \qquad \ldots\ldots\ldots \qquad \begin{matrix} 0.1764 \\ 0.3434 \\ 0.0582 \\ 0.1188 \\ 0.1055 \\ 0.1979 \end{matrix}$$

Distributed Algorithm

- No central authority to store and compute **t**.
- Each peer i holds its own opinions $\mathbf{c_i}$.
- For now, let's ignore questions of lying, and let each peer store and compute its own trust value.



$$t_i^{(k+1)} = c_{1i} t_1^{(k)} + \dots + c_{ni} t_n^{(k)}$$

Distributed Algorithm

For each peer i {

-First, ask peers who know you for their opinions of you.

-Repeat until convergence {

-Compute current trust value: $t_i^{(k+1)} = c_{1i}\, t_1^{(k)} + \ldots + c_{ni}\, t_n^{(k)}$

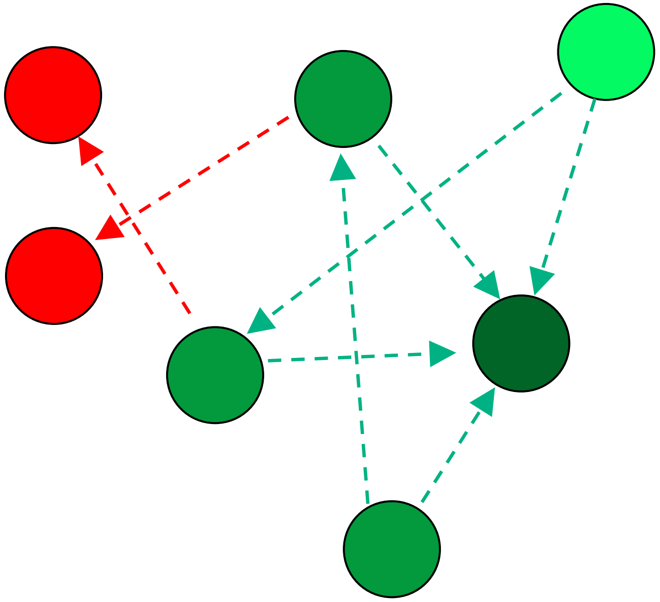
-Send your opinion c_{ij} and trust value $t_i^{(k)}$ to your acquaintances.

-Wait for the peers who know you to send you their trust values and opinions.

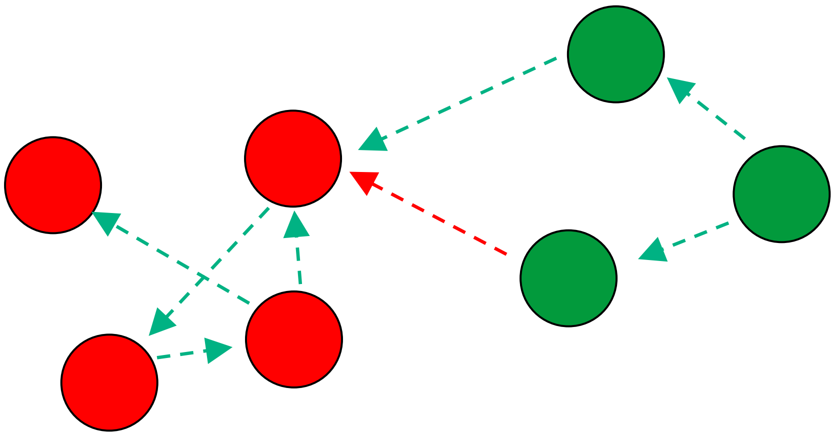
}

}

Probabilistic Interpretation : Random Surfer Model



Malicious Collectives :
Random Jumps to avoid dead ends



Revised Non-distributed Algorithm

- Initialize:

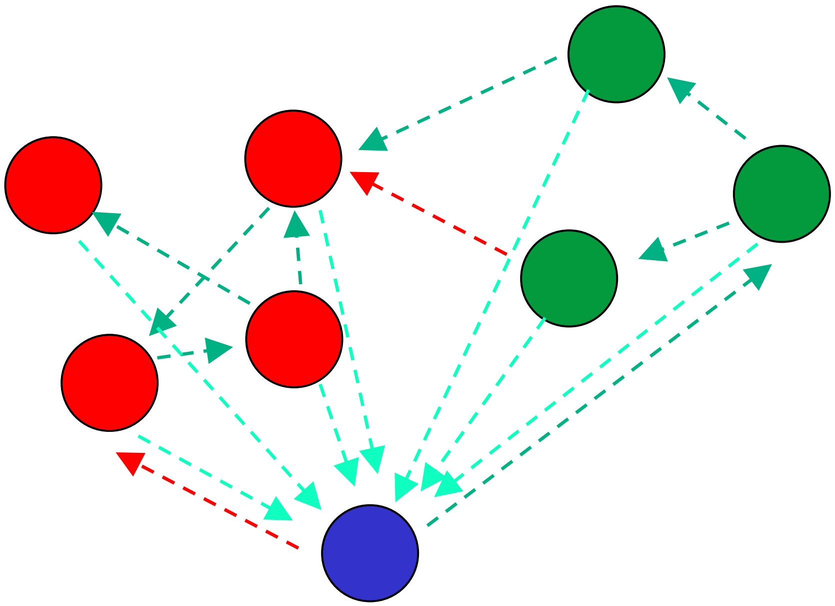
$$\mathbf{t}^{(0)} = \begin{bmatrix} \frac{1}{n} & \ldots & \frac{1}{n} \end{bmatrix}^{\mathbf{T}}$$

- Repeat until convergence:

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

Pre-trusted Peers

- Battling Malicious Collectives
- Inactive Peers
- Incorporating heuristic notions of trust
- Improving Convergence Rate



Practical Issues

- Apriori notions of trust
  - Can we assign any profit to newcomers?
  - Only the first few peers to join the network are known to be trustworthy
  - $p_i = 1/|p|$ if $i \in P$, and $p_i = 0$ otherwise
  - Use $\vec{p}$ instead of $\vec{e}$

Practical Issues(2)

- Inactive Peers
  - What happens if peer i doesn't download from anybody else?

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

  - Choose to trust the pre-trusted peers

Practical Issues(3)

- Malicious Collectives
  - a group of malicious peers who know each other
  - How to prevent them from subverting the system?

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

  - The modified algorithm:

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

repeat

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

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

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

until $\delta < \epsilon$;

Distributed EigenTrust

- Assumption: Everyone is honest
- Each peer computes its own global trust value:

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

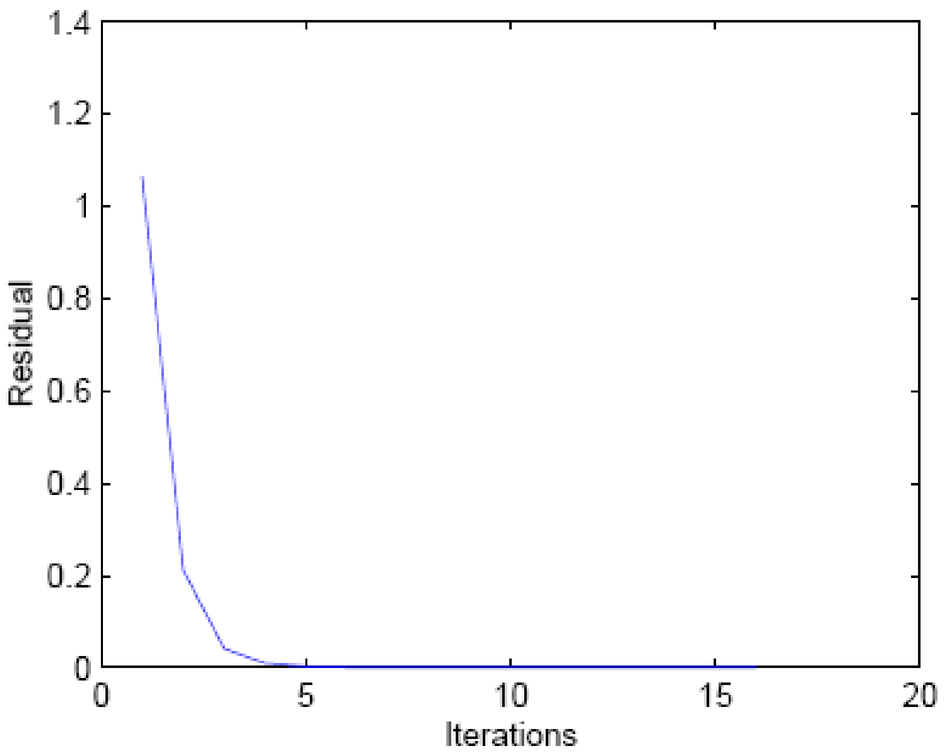
Definitions:

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

Algorithm:
Each peer i do {
Query all peers $j \in A_i$ for $t_j^{(0)} = p_j$;
repeat
  Compute $t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \ldots + c_{ni}t_n^{(k)}) + ap_i$;
  Send $c_{ij}t_i^{(k+1)}$ to all peers $j \in B_i$;
  Compute $\delta = |t_i^{(k+1)} - t_i^{(k)}|$;
  Wait for all peers $j \in A_i$ to return $c_{ji}t_j^{(k+1)}$;
until $\delta < \epsilon$.;
}

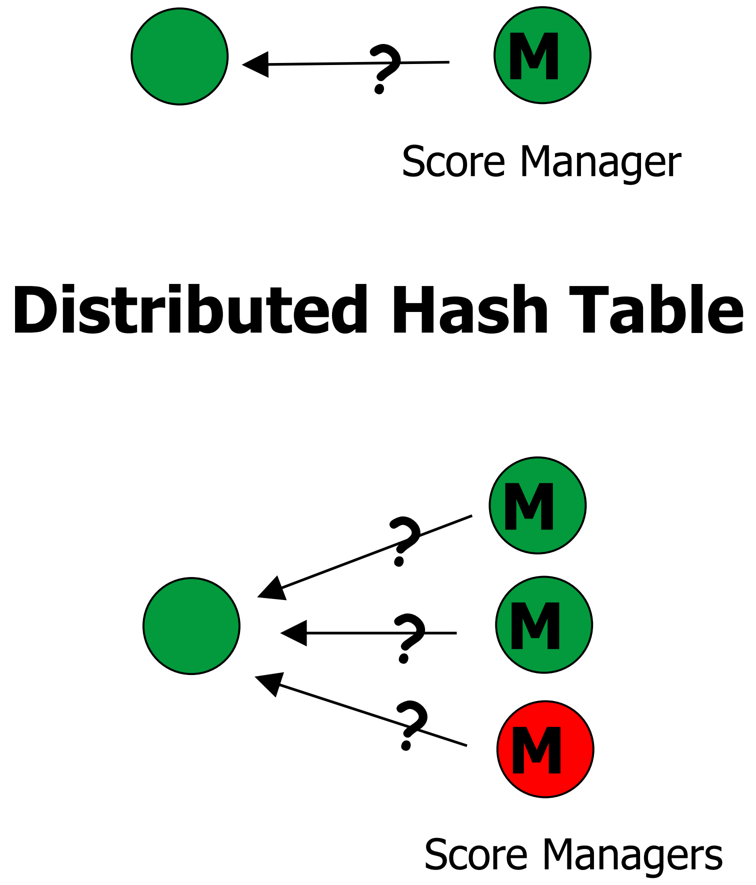
Algorithm Complexity

- The algorithm converges fast
  - A network of 100 peers after 100 query cycles



Secure Score Management

- Two basic ideas:
  - 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.

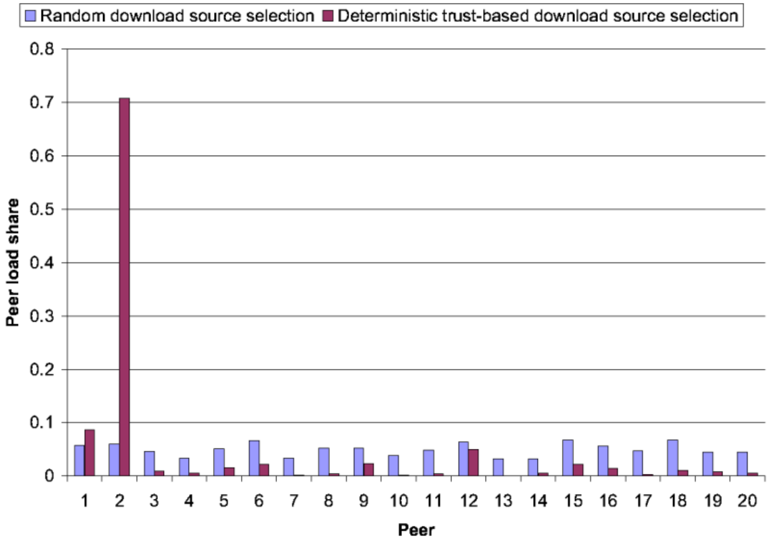


How to use the trust values t_i

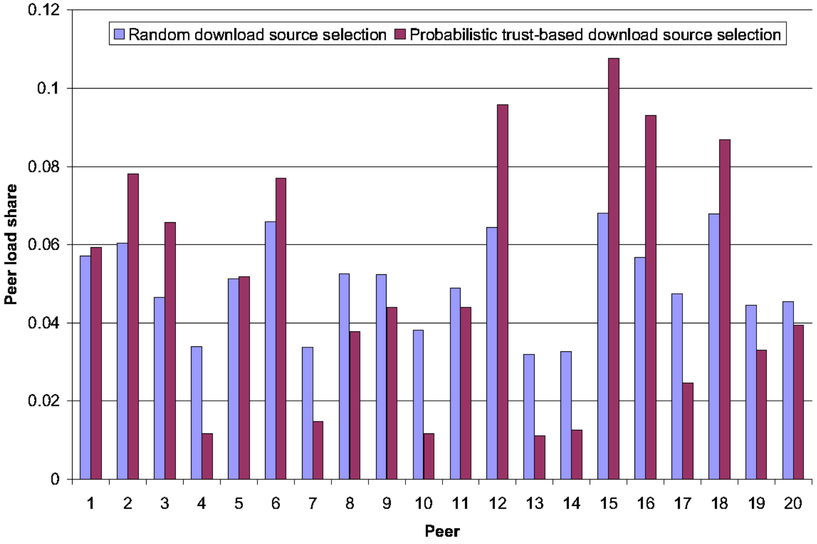
- When you get responses from multiple peers:
  - **Deterministic**: Choose the peer with highest trust value.
    - To avoid discriminating against new peer (at the risk of inviting malicious peer), occasionally use other peer.
  - **Probabilistic**: Choose a peer with probability proportional to its trust value.
    - This approach improves *load balancing*.

Load Distribution

Deterministic Download Choice

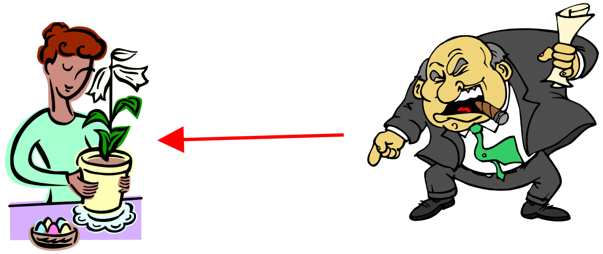


Probabilistic Download Choice

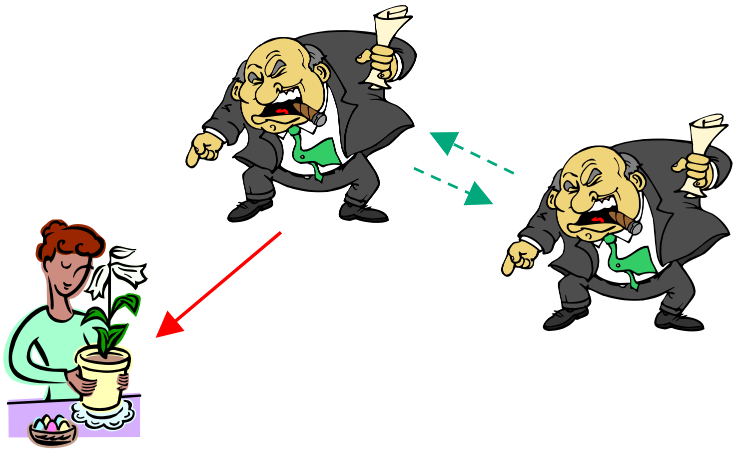


Threat Scenarios

- **Malicious Individuals**
  - Always provide inauthentic files.

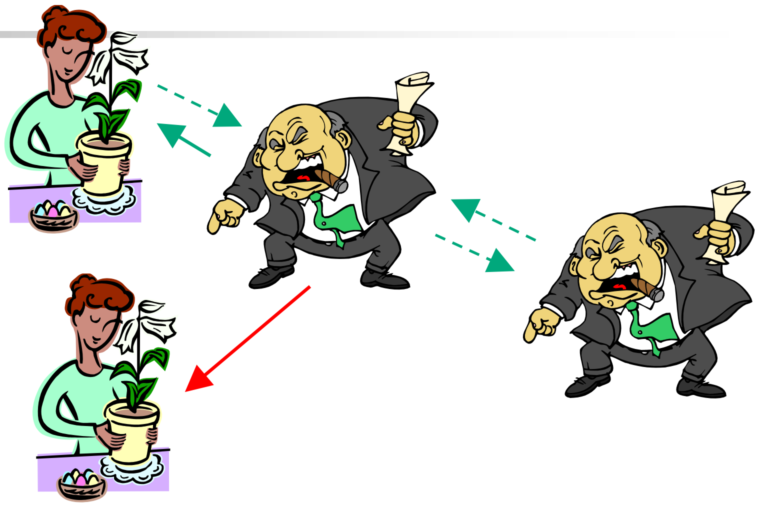


- **Malicious Collective**
  - Always provide inauthentic files.
  - Know each other. Give each other good opinions, and give other peers bad opinions.

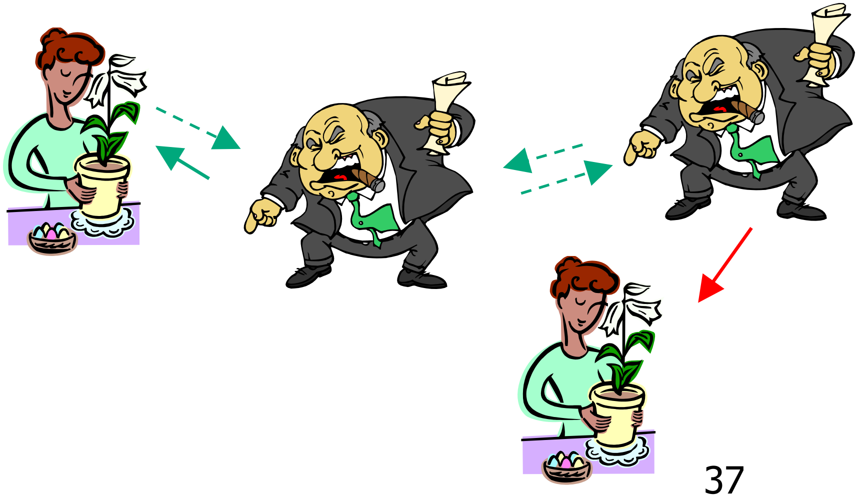


More Threat Scenarios

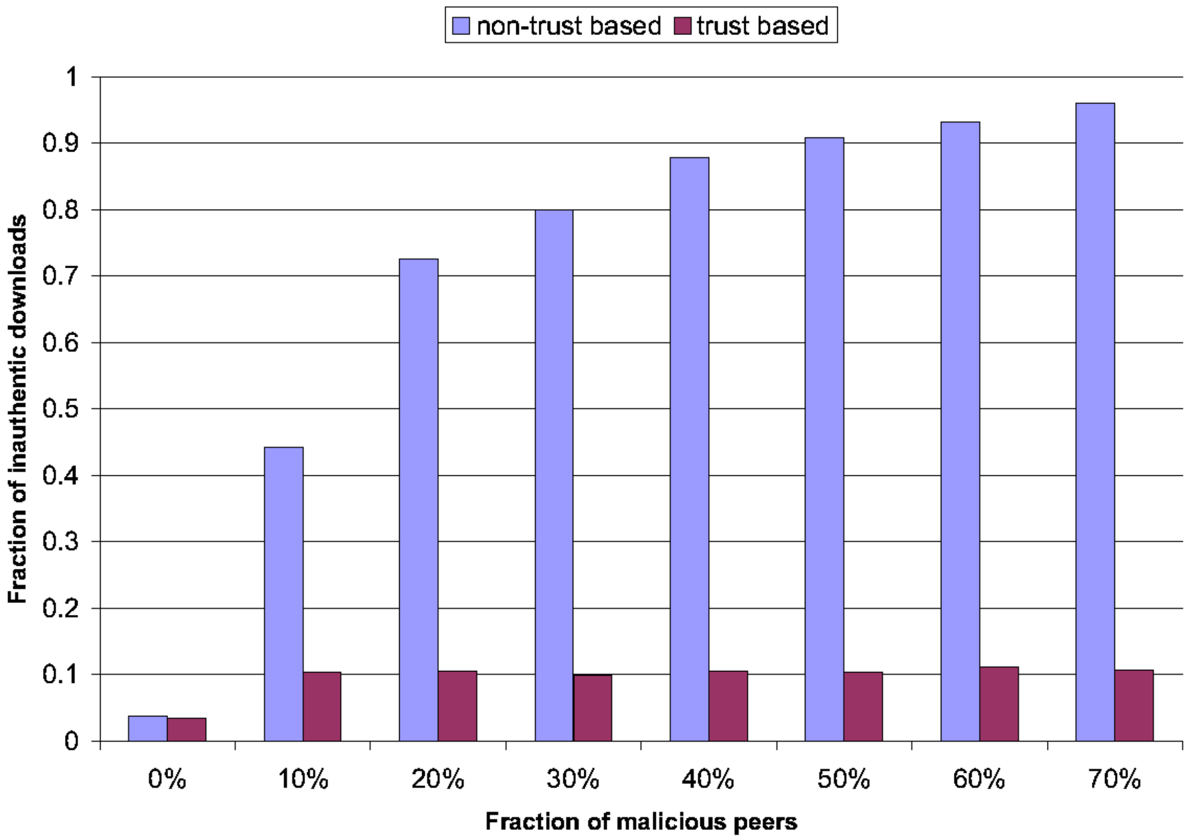
- **Camouflaged Collective**
  - Provide authentic files some of the time to trick good peers into giving them good opinions.



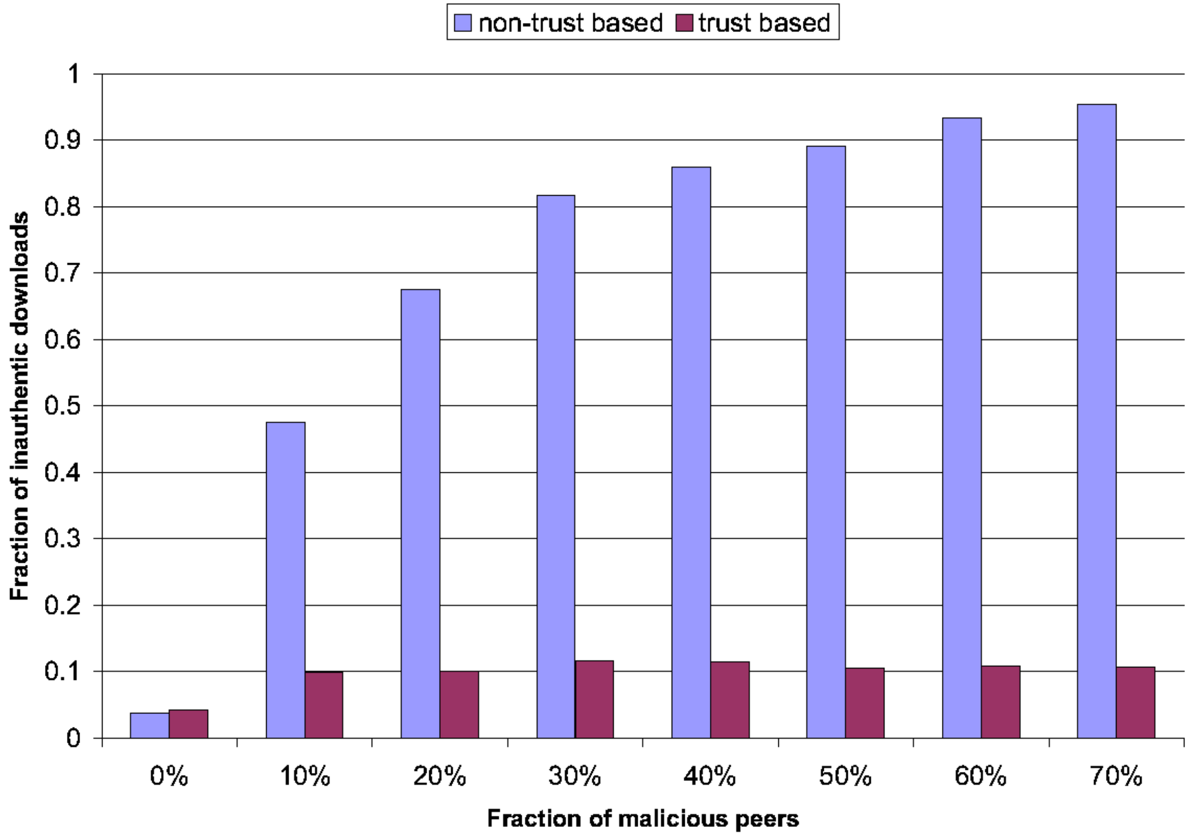
- **Malicious Spies**
  - Some members of the collective give good files all the time, but give good opinions to malicious peers.



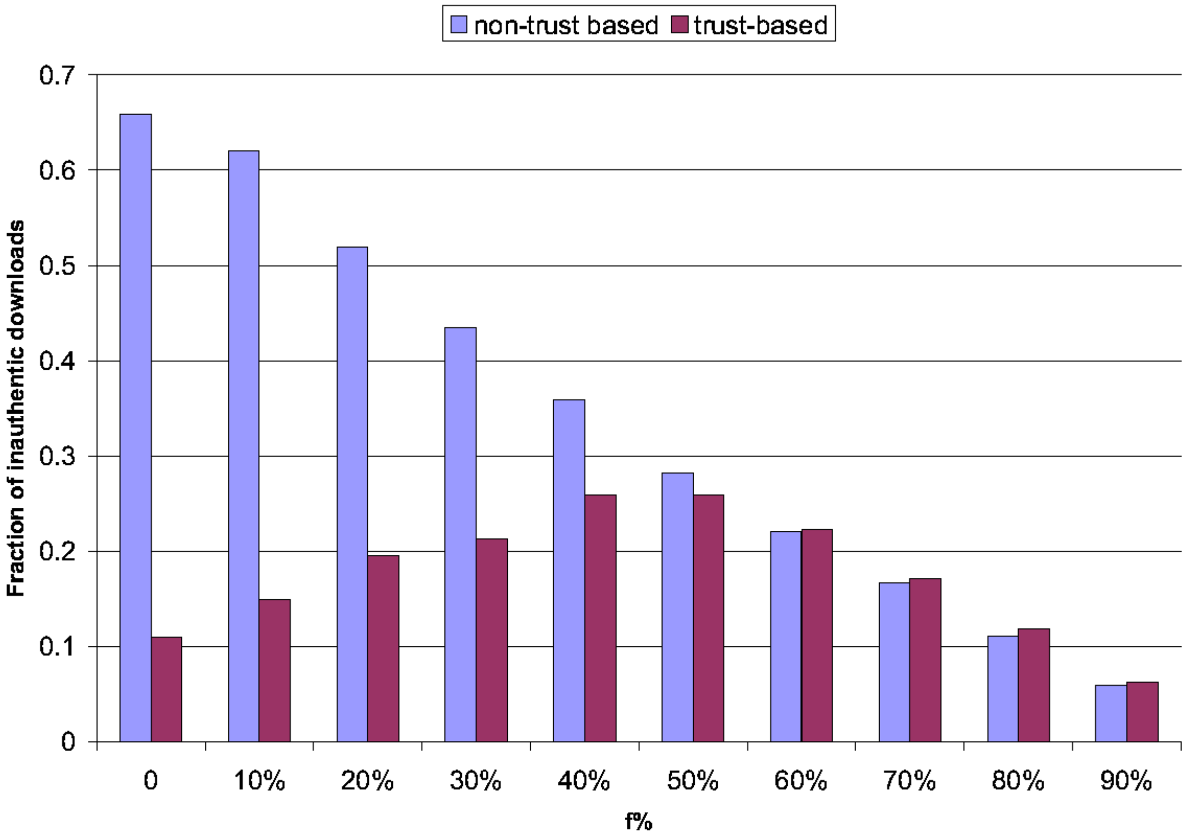
Malicious Individuals



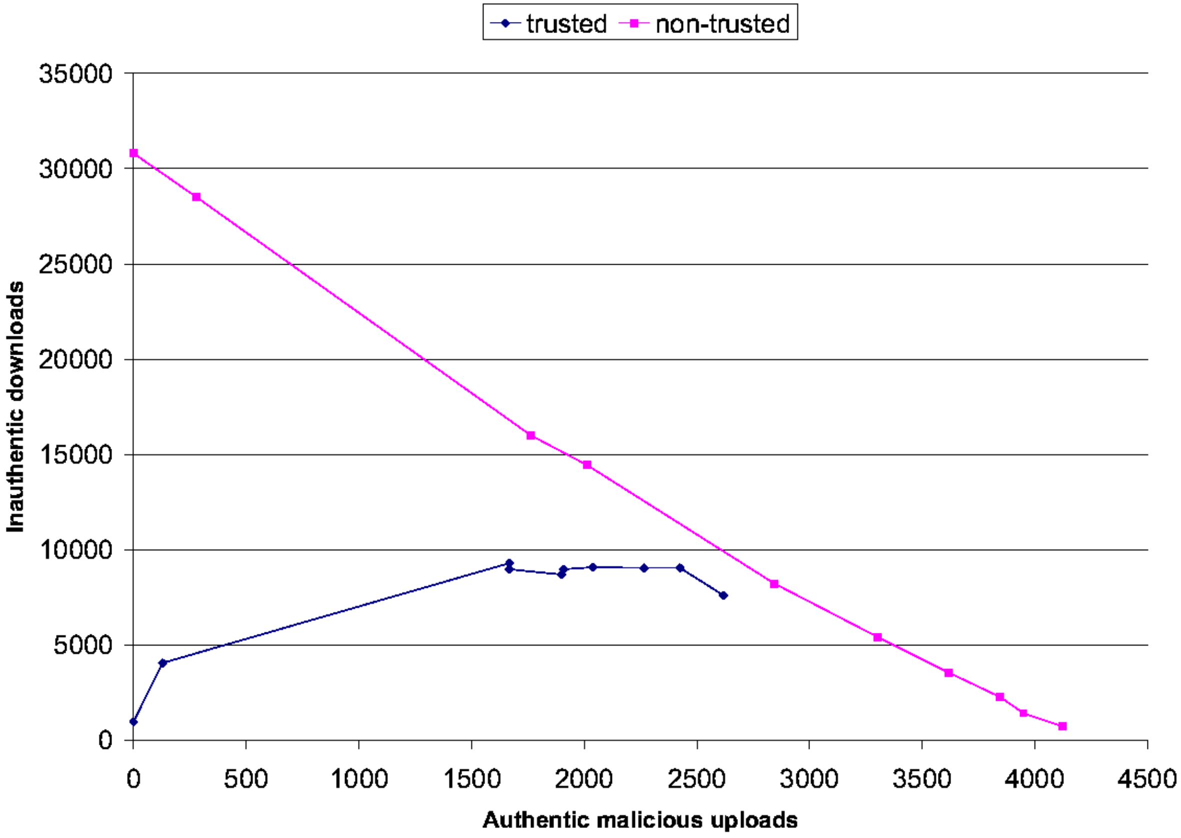
Malicious Collective



Camouflaged Collective



Malicious Spies



Conclusion

- Eigentrust
  - Dramatically reduces number of inauthentic files on the network.
  - Robust to malicious peers.
  - Low overhead.
- Paper available at http://www.stanford.edu/~sdkamvar/research.html