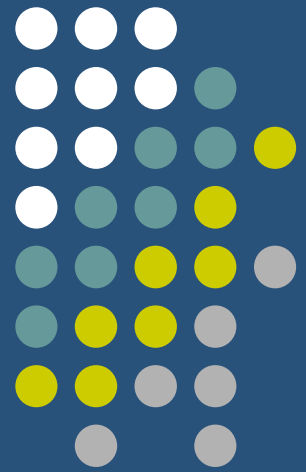
EigenTrust Algorithm for Reputation Management in P2P Networks

Authors: Kamvar, Schlosser, Garcia

Andrew G. West

QTM: 11/16/2007



Presentation Outline

- P2P network introduction
  - Ideal environment for malicious behavior and spread of replicating inauthentic files
- EigenTrust algorithm
  - Method to decrease inauthentic downloads
  - Based on the *distributed* and *secure* calculation of global trust values
  - Identifies and isolates malicious peers
- EigenTrust effectiveness via simulations

Introduction to P2P

- Peer-to-Peer networks have advantages over Client-Server model such as robustness, scalability, and diversity of data.
- A complete lack of accountability opens door to abuse. For example, *VBS.Gnutella*
  - More common (less dangerous) are inauthentic file attacks; mislabeled or non-working files.
- Literature suggests future of P2P depends on peers knowing about quality of resources

P2P Design Considerations

- Issues important to all P2P reputation sys:
  - (1): Self-policing. Peers must be enforcers, having central authority defies P2P ideology.
  - (2): Anonymity. Opaque identifiers, not IP.
  - (3): No profit to newcomers. Reputation obtained through good behavior. Malicious peers should gain no advantage reverting to newcomer status.
  - (4): Minimal overhead: Computation, storage, and maybe most important, message size/frequency.
  - (5): Robust to malicious collectives.

EigenTrust (ET) Summary

- Goal is to identify malicious peers
  - Not files. Lack of removal mechanism? ∞ files
- Each peer *i* given *global trust value* that reflects experiences of all other peers with *i*.
- All peers participate in calculating this value with goal being minimal overhead.
- Strives to be secure, minimizing ability of peers to lie for their own benefit.
- Identify and isolate malicious peers.

Reputation Systems

- Initially similar to eBay system (centralized)
  - Sidenote: eBay gets a ton of attention.
- Each time peer *i* downloads from peer *j*
  - The transaction gets rated:
  - Positively (*tr(i,j)* = 1), or negatively (*tr(i,j)* = -1)
- Transactions form local trust value $s_{i,j}$
  - $s_{i,j} = \Sigma(tr_{i,j})$
  - Alternatively, $s_{i,j} = sat(i,j) – unsat(I,j)$

Problems w/ Distributed RSYS

- Previous slide characteristic of most RSYS
  - Existing (dist) systems suffer from two problems
    - (1): Aggregation too small, relatively few peers queried and a wide enough view of reputation not achieved
    - (2): Wide view obtained, but at the cost of network congestion b/c of many messages
- ET attacks this issue with *transitive trust*
  - Peer *i* will have good opinion of those peers it has had positive transactions with AND is likely to trust the opinions of those peers about others.

ET: Normalizing LTV

- Before aggregating local trust values, they are normalized:
  - Otherwise, malicious peers could present arbitrarily low/high values to subvert system.
- Normalized local trust value:
  - $c_{i,j} = max(s_{i,j},0) / (\Sigma_j \ max(s_{i,j},0))$
- This normalization has admitted drawbacks:
  - Values are relative, no absolute interpretation
  - If $c_{i,j} = c_{i,k}$ then j has same reputation as k, but past interaction history could be very different
  - Only considers peers with a positive history

A. West Running Example

- We'll pretend that we are peer *i.*
  - And we've rated some past interactions.
    - j_0: +1, +1, +1, -1, +1, +1, -1, -1, +1 : $s_{i,j0} = +3$
    - j_1: +1, +1, +1, +1, +1, +1, +1 : $s_{i,j1} = +7$
    - j_2: -1, -1, +1, -1, +1, +1, -1, -1 : $s_{i,j2} = -2$
    - j_3: -1, +1, +1, +1, -1, +1, +1, +1 : $s_{i,j3} = +4$
  - And its no big deal to normalize them:
    - $c_{i,j0} = (3/14) = 0.21$ $c_{i,j1} = (7/14) = 0.50$
    - $c_{i,j2} = (0/14) = 0.00$ $c_{i,j3} = (4/14) = 0.29$

ET: Aggregating LTV

- To aggregate, peer *i* should ask his acquaintances about another peer, *k*. These opinions should be weighted by the trust *i* has in the acquaintances.
  - $t_{i,k} = \Sigma_j (c_{i,j} * c_{j,k})$
  - That is $t_{i,k}$ represents the trust peer *i* places in peer *k* based on asking his friends.
  - Think about this in linear algebra terms…

A. West Running Example(2)

- If C = *matrix* $[c_{i,j}]$, and $t_i^{\rightarrow}$ is the vector containing the values $t_{i,k}$ then $t_i^{\rightarrow} = C^T \times c_i^{\rightarrow}$
- Um… what?

	i	j0	j1	j2	j3
i	0.	0.33	0.25	0.	0.50
j0	0.21	0.	0.25	1.00	0.13
j1	0.50	0.33	0.	0.	0.25
j2	0.	0.33	0.25	0.	0.12
j3	0.29	0.	0.25	0.	0.

• Notice that the first column of the matrix at left corresponds to what we previously calculated as peer *i*'s assumption about other peer nodes

A. West Running Example(3)

- If C = *matrix* $[c_{i,j}]$, and $t_i^{\rightarrow}$ is the vector containing the values $t_{i,k}$ then $t_i^{\rightarrow} = C^T \times c_i^{\rightarrow}$
- Um… what?

	i	j0	j1	j2	j3
i	0.	0.33	0.25	0.	0.50
j0	0.21	0.	0.25	1.00	0.13
j1	0.50	0.33	0.	0.	0.25
j2	0.	0.33	0.25	0.	0.12
j3	0.29	0.	0.25	0.	0.

• I made up values for the other columns, however notice that all columns sum to 1, because each underwent local normalization.

A. West Running Example(4)

- If C = *matrix* $[c_{i,j}]$, and $t_i^{\rightarrow}$ is the vector containing the values $t_{i,k}$ then $t_i^{\rightarrow} = C^T \times c_i^{\rightarrow}$
- Um… what?

	i	j0	j1	j2	j3
i	0.	0.33	0.25	0.	0.50
j0	0.21	0.	0.25	1.00	0.13
j1	0.50	0.33	0.	0.	0.25
j2	0.	0.33	0.25	0.	0.12
j3	0.29	0.	0.25	0.	0.

- Peer's shouldn't be making claims about their own trust value, so the value is initially set to zero (A. West assumption)

A. West Running Example(5)

- If C = *matrix* $[c_{i,j}]$, and $t_i^{\rightarrow}$ is the vector containing the values $t_{i,k}$ then $t_i^{\rightarrow} = C^T \times c_i^{\rightarrow}$
- Um… what?

	i	j0	j1	j2	j3
i	0.34	0.08	0.21	0.33	0.11
j0	0.16	0.48	0.34	0.	0.29
j1	0.14	0.17	0.27	0.33	0.29
j2	0.23	0.08	0.11	0.33	0.11
j3	0.12	0.18	0.07	0	0.21

- If we square the previous matrix. We get the one at left.
- This is equivalent of aggregation. If we look at the new first column, these are the adjusted trust values if we ask one level deep.
- This is what the math above is really saying

A. West Running Example(6)

	i	j0	j1	j2	j3
i	0.19	0.22	0.21	0.18	0.21
j0	0.29	0.25	0.27	0.31	0.26
j1	0.22	0.22	0.22	0.22	0.22
j2	0.15	0.17	0.16	0.14	0.17
j3	0.12	0.11	0.11	0.13	0.11

- If we take the original matrix to the 7th power, we get the one at left. This is the same data we would get if we performed aggregation operation 7 acquaintances deep.

- Notice that each column (the trust values of other nodes, as perceived by that columns label) is becoming consistent across the matrix.
- If we take the original matrix to a high enough power *N*, all such vectors will converge to a single value, the left principal eigenvector of *C*.
- We label the convergent vector the *global trust vector*.

Basic EigenTrust

- Note: In reality, we won't have information about the entire network (initially, matrix may be sparse). As we delve into acquaintances, trust-data about such peers will be realized.
- Informally, we can keep taking the matrix to a higher and higher power until the difference (norm) between the current global trust vector and the previous falls below some desired threshold.
- For now, we are assuming a central server and we'll deal with the distributed environment in a minute.

A Larger Example

$$\begin{bmatrix} 0.5 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0.8 & 0 & 0.25 & 0 & 0 & 0.5 \\ 0 & 0.3 & 0 & 0.5 & 0 & 0.12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0.1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0.1 & 0.1 & 0 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.5 & 0.75 & 0 & 0 \\ 0 & 0.1 & 0 & 0.5 & 0 & 0.13 & 0 & 0 \end{bmatrix}$$

Original Matrix Squared

$$\begin{bmatrix} 0.45 & 0.20 & 0.32 & 0 & 0.1 & 0 & 0 & 0.2 \\ 0.25 & 0.52 & 0 & 0.65 & 0 & 0.16 & 0.25 & 0 \\ 0.15 & 0.01 & 0.25 & 0 & 0.1 & 0 & 0 & 0.4 \\ 0 & 0.05 & 0 & 0.25 & 0 & 0.06 & 0 & 0 \\ 0.05 & 0 & 0.18 & 0 & 0.53 & 0.75 & 0 & 0.05 \\ 0.05 & 0.06 & 0.08 & 0.05 & 0.03 & 0.01 & 0.25 & 0.05 \\ 0 & 0.02 & 0.07 & 0.05 & 0.19 & 0.01 & 0.5 & 0 \\ 0.05 & 0.10 & 0.1 & 0 & 0.06 & 0 & 0 & 0.3 \end{bmatrix}$$

Original to 10th power

$$\begin{bmatrix} 0.21 & 0.19 & 0.20 & 0.18 & 0.18 & 0.18 & 0.16 & 0.22 \\ 0.24 & 0.27 & 0.20 & 0.30 & 0.20 & 0.21 & 0.25 & 0.19 \\ 0.10 & 0.08 & 0.11 & 0.07 & 0.10 & 0.10 & 0.07 & 0.12 \\ 0.02 & 0.03 & 0.02 & 0.03 & 0.02 & 0.02 & 0.03 & 0.02 \\ 0.17 & 0.16 & 0.19 & 0.14 & 0.21 & 0.21 & 0.17 & 0.19 \\ 0.07 & 0.08 & 0.07 & 0.08 & 0.08 & 0.08 & 0.09 & 0.07 \\ 0.14 & 0.16 & 0.14 & 0.16 & 0.17 & 0.17 & 0.18 & 0.13 \\ 0.05 & 0.04 & 0.05 & 0.03 & 0.05 & 0.05 & 0.04 & 0.06 \end{bmatrix}$$

Practical Issues

- Three issues are ignored by previous alg:
  - (1): A priori notions of trust – Some peers are known trustworthy (founders).
    - Trust data for these peers is artificially inflated.
    - These peers are important later, we identify them as *P*
  - (2): Inactive peers – If a peer doesn't interact, or has only bad interactions, its will have a 0-vector and be stuck with 0-vector.
    - To remedy this, if peer *i* knows no one, or trusts no one, then *i* should trust the pre-trusted peers (1).

Practical Issues Contd.

- (3): Malicious collectives
  - Defined: Group of peers who know each other, give each other high local trust values and everyone else low values in order to subvert system.
  - Notation gets tricky, but intuitively:
    - Force every peer to form some opinion about some of the members of *P* (the trusted set).
      - It looks suspect if a peer rates a member of *P* lowly
      - Plus, *P* will have transactions with well-behaved peers, which will begin to erode the cyclic pattern the malicious collective was depending on.
    - This constraint furthermore reduces “cycles” within the matrix and guarantees the computation will converge

Picking Pre-Trusted Peers

- Since the notion of pre-trusted peers (*P)* defends against malicious collectives, guarantees convergence, and provides a foundation of trust for inactive peers, their selection is critical.
  - If a member of *P* is malicious, things fall apart very quickly
  - Better to have few *P* with strong notion of trust
  - Still an interesting, open research area

Distributed EigenTrust

- For now let’s assume everyone is honest, and capable of storing its own local trust vector (its direct opinion of its acquaintances), and storing its own global trust value (the global normalized perception of itself).
- Then the algorithm looks something like…

Copy + Paste = Easy

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 3: Distributed EigenTrust Algorithm.

- What's important to see here is that peer *i* needs to send only to B_i and receive only from A_i.
- Probabilistically these sets should represent only a very small portion of the entire portion of the P2P network.
- Thus message passing overhead should be minimal
- If several nodes are heavily active, their transmission rates can be capped with little consequence

Distributed Alg. Complexity

- The authors actually say nothing about complexity, but are careful about bounding:
  - Probability indicates message passing shouldn't be too huge in volume
    - If it is too big for a few peers, just cap it, no big deal
  - Convergence happens very quickly
    - This is the looping part of the previous slide
    - About 10 iterations in the general case
      - Math + presumed data consistency guarantees quick convergence, it's a complex topic though

Secure EigenTrust

- It’s obviously insecure to let a peer manage its own trust data. Easily manipulated.
  - So we let another peer calculate it, instead.
  - In this case, a malicious peer might return an incorrect trust value for the peer its assigned.
    - So let’s let multiple peers compute the trust value of a single peer in the system.
    - A majority vote on the trust value can then settle conflicts arising from malicious peers being involved.
    - Those tracking the trust of *i* are its *score managers*

Assigning Trust Managers

- Naïve strategies are inappropriate, as it may allow malicious peers to still cooperate
- Instead: Use a distributed hash table (DHT) to map peers into logical space using IP addr.
  - Dynamically subdivide the space into regions
  - Every peer in a region becomes a *trust manager* for all other peers in the same region
  - By knowing another peer's IP addr (or other key), they can hash to which region they reside, and collect information about them.

Increased Security/Reliability

- A well implemented approach meeting the goals of the previous slide is advantageous:
  - (1) Anonymity: Peers don’t know precisely whose trust values they are computing. Thus, malicious peers can not help other malicious peers
  - (2) Randomization: Peers can’t choose or reason the coordinates they map too. Thus, malicious nodes can’t regionalize.
  - (3) Redundancy: Multiple users in a single region, and even multiple hash functions and multiple dimension residency, if required.

Using Global Trust Values

- So we've done a lot of work computing these values. Now how do we use them?
  - (1): Isolate malicious peers by biasing users to download from reputable peers
  - (2): Provide incentive for peers to share files by rewarding reputable peers

Isolating Malicious Peers

- We could have each peer download from the highest trusted peer that responds to query.
  - Those with high trust would be overloaded
  - Doesn’t give others chance to build reputation
- Instead: distribute downloads probabilistically based on trust values
  - Limits unsatisfactory downloads, distributes load
  - Users have option to bias selection through some combination of local and global vectors.

Incentive for Freeriders to Share

- Reputable peers may be rewarded with increased connectivity or greater bandwidth
  - There is the obvious benefit of people sharing for the above reasons.
  - Secondly, it gives non-malicious peers incentive to remove non-authentic files that they may having accidentally/unknowingly downloaded. Keeps the system as a whole tidy.
    - Assumes that users are educated about rating model

Simulations

- The authors setup a model representing a typical P2P network
  - Consists of good and adversarial nodes
  - Data distribution is concentrated on users interacting in only certain data categories
  - Simulation done in execution cycles, after each, the global trust values are updated.
  - Lots of insignificant details: All backed by literature.
  - Metric of interest: Number of inauthentic downloads vs. authentic ones

Graphs, charts, oh my!

- A lot of trees are wasted to demonstrate what intuitively is assumed to be true:
  - Probabilistic download source selection leads to much better load distribution.
  - Both consistently malicious individuals and collectives are squashed by the algorithm
  - Impressive statistics nonetheless:
    - Even with 70% of users acting (consistently) malicious (individually or collectively), authentic files can still be returned with 90% probability

A Few Deficiencies Nonetheless

- Some subtle attacks are harder to reconcile:
  - Adversaries with camouflage: Returning an inauthentic file only x% of the time makes it harder to isolate those employing this strategy
  - Malicious spies: Only downloading, and providing + feedback to those distributing inauthentic files
- Other attacks are highly effective:
  - Sybil: Create thousands of users. Use once to distribute inauthentic file, then disappear (Sol: CAPTCHA).
  - Virus-Disseminators: 1% of the time send out a virus. Solution: None, but then again, no one claims to solve this.

Conclusions

- Presented method to minimize impact of malicious peers in P2P system.
- Entire system history can efficiently be taken into consideration through calculation of matrix eigenvector.
  - Furthermore, this can be done in secure and distributed manner
- The strategy proves effective under a number of common attacks.

A. West Commentary

- Overall a pretty solid paper
- We're still looking at a "soft" feedback system
  - It is the users responsibility provide feedback
    - A client might be viewed unfavorably if its feedback retrieval mechanism is too intrusive.
    - Similarly, one that is masked might only be taken advantage of by malicious users
    - The desire to provide feedback probably isn't as strong as with monetary exchange systems (eBay).
- Furthermore we assume that "good" user has ability to identify inauthentic files? Not always the case.

A. West Commentary (2)

- “Hard” feedback in P2P systems
  - Hard feedback can always be objective, so it seems desirable to exploit it in any design.
    - Virus scan? Some P2P clients have this functionality. Users sharing infected files could have their local trust values dramatically reduced (i.e. < -1).
    - Packet completion? If a user fails to complete sharing data it was supposed to…. Punish it!
    - Downstream/upstream rate? In the general case, there is a relation. If someone is downloading at T3 rates and uploading like a 56k, maybe some action should be taken.