Reputation Management Algorithms & Testing

Andrew G. West
November 3, 2008

EigenTrust

- EigenTrust (Hector Garcia-molina, et. al)
  - A normalized vector-matrix multiply based method to aggregate trust such that there is a globally convergent view

---

$$A = \begin{bmatrix} \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} & \begin{pmatrix} pos:3 \\ neg:1 \end{pmatrix} & \begin{pmatrix} pos:3 \\ neg:2 \end{pmatrix} \\ \begin{pmatrix} pos:9 \\ neg:3 \end{pmatrix} & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} & \begin{pmatrix} pos:8 \\ neg:1 \end{pmatrix} \\ \begin{pmatrix} pos:2 \\ neg:4 \end{pmatrix} & \begin{pmatrix} pos:5 \\ neg:4 \end{pmatrix} & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} \end{bmatrix}$$

Imagine past feedback between users as a matrix.

The matrix should be interpreted vector-wise. That is, column 1 is representative of user 1's experience with the other 2 users in the NW.

EigenTrust (2)

$$A = \begin{bmatrix} \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} = 0 & \begin{pmatrix} pos:3 \\ neg:1 \end{pmatrix} = 2 & \begin{pmatrix} pos:3 \\ neg:2 \end{pmatrix} = 1 \\ \begin{pmatrix} pos:9 \\ neg:3 \end{pmatrix} = 6 & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} = 0 & \begin{pmatrix} pos:8 \\ neg:1 \end{pmatrix} = 7 \\ \begin{pmatrix} pos:2 \\ neg:4 \end{pmatrix} = 0 & \begin{pmatrix} pos:5 \\ neg:4 \end{pmatrix} = 1 & \begin{pmatrix} pos:0 \\ neg:0 \end{pmatrix} = 0 \end{bmatrix}$$

At each position calculate the "feedback integer" as:

```
fback_int := pos-neg;
if(fback_int < 0)
    fback_int := 0
```

$$A' = \begin{bmatrix} 0/6 & 2/3 & 1/8 \\ 6/6 & 0/3 & 7/8 \\ 0/6 & 1/3 & 0/8 \end{bmatrix}$$

Then, normalize the matrix vector-wise.

Realize that this normalization means that our end results will only have relative interpretations, not absolute ones

EigenTrust (3)

$$p = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Next, initialize pre-trust vector `p`, which encodes a-priori notions of trust. If there are no pre-trusted users, as in this case, each entry should simply be `1/n` (n = # peers).

$$A' = \begin{bmatrix} 0/6 & 2/3 & 1/8 \\ 6/6 & 0/3 & 7/8 \\ 0/6 & 1/3 & 0/8 \end{bmatrix}$$

$$t_0 = p \qquad t_\infty = \begin{bmatrix} 0.35 \\ 0.49 \\ 0.16 \end{bmatrix}$$

$$t_{k+1} = (0.5 * A^{'T} * t_k) + 0.5 * p$$

Finally, we multiply using the equations at left. We compute `t_k` for sufficiently high `k` (usually k=7-10) such that `t` converges.

The resultant `t` is the 'global trust vector' a relative ranking of trust between nodes.

Note: The coefficients on the multiplication allow one to fine tune how much 'extra' influence pre-trusted users should have.

Note: Additional steps needed to distribute and secure this computation.

EigenTrust (4)

- What EigenTrust gives us
  - A global average, with more weight given to pre-trusted peers
- Advantages of EigenTrust
  - Mathematically elegant
  - Scalable computation (global nature + some tricks)
  - Trust doesn't weaken via transitivity (good for sparseness)
- Disadvantages of EigenTrust
  - Normalization leads to relative interpretation
  - No means of measuring negative trust
  - Globally agreed upon trust vector might not be the best idea in the face of very malignant networks

TNA-SL

- Trust Network Analysis w/Subjective Logic (Josang)
  - Uses Subjective Logic operators to analyze network graphs

Opinion: (b, d, u, a)

Expected Value: b+u*a

$$
\begin{aligned}
\text{belief} &= (pos/(pos+neg+2.0)) \\
\text{disbelief} &= (neg/(pos+neg+2.0)) \\
\text{uncertainty} &= (2.0/(pos+neg+2.0)) \\
\text{base-rate} &= \begin{cases} 1.0 & \text{if user is pre-trusted} \\ 0.5 & \text{otherwise} \end{cases}
\end{aligned}
$$

Trust is stored in Opinions, which are a 4-tuple (b, d, u, a):

b = belief d = disbelief
u = uncertainty a = a-priori trust

where (b+d+u)=1.0 and a=[0...1]

'Trust' in an Opinion is equal to the expected value shown at left

Table shows how to go from prior feedback -> Opinion.

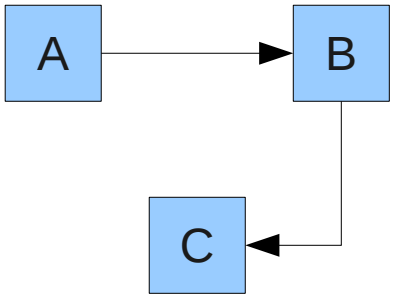
Subjective Logic

- Two Subjective Logic operators are of note:
  - 'Discount' for transitivity and 'consensus' for averaging

'Discount' encodes transitivity. Trust is weakened when a hop is taken. The discount is weaker if the intermediate node is highly trusted and vice versa.

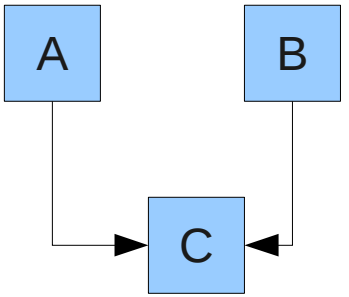
Discount: $\otimes$
$b_C^{A:B} = b_B^A b_C^B$
$d_C^{A:B} = b_B^A d_C^B$
$u_C^{A:B} = d_B^A + u_B^A + b_B^A u_C^B$
$\alpha_C^{A:B} = \alpha_C^B$

$$\omega_C^{A:B} = \omega_B^A \otimes \omega_C^B$$



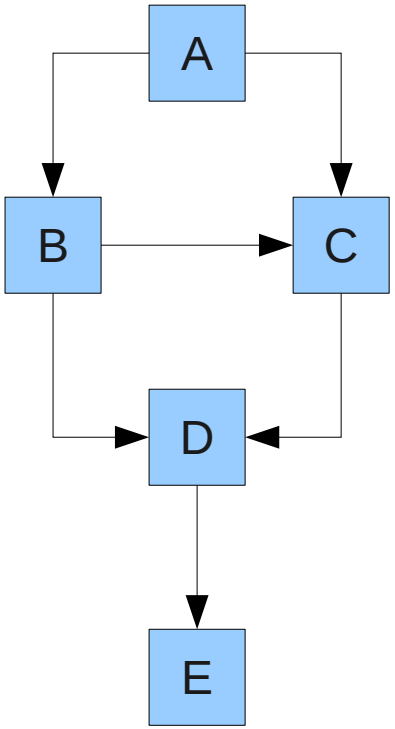
'Consensus' let's us average opinions. Averaging decreases uncertainty, while averaging belief and disbelief.

$$\omega_C^{A \diamond B} = \omega_C^A \oplus \omega_C^B$$



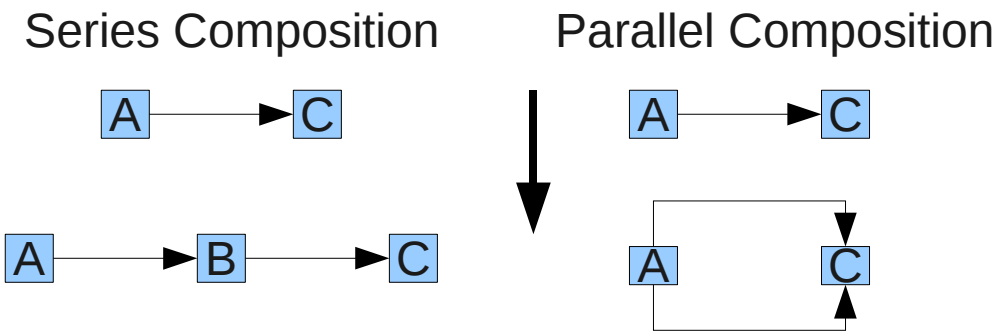
Consensus: $\oplus$
$b_C^{A \diamond B} = \frac{b_C^A u_C^B + b_C^B u_C^A}{u_C^A + u_C^B - u_C^A u_C^B}$
$d_C^{A \diamond B} = \frac{d_C^A u_C^B + d_C^B u_C^A}{u_C^A + u_C^B - u_C^A u_C^B}$
$u_C^{A \diamond B} = \frac{u_C^A u_C^B}{u_C^A + u_C^B - u_C^A u_C^B}$
$\alpha_C^{A \diamond B} = \alpha_C^A$

- We're given the graph below (with Opinions on edges) and want to determine the trust A has in E. How?



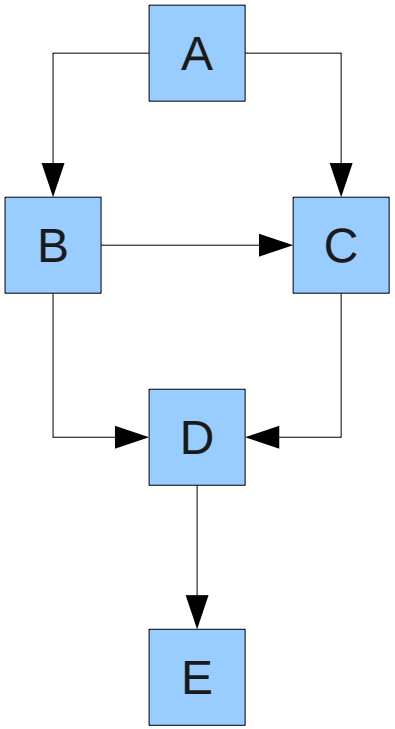
We could enumerate all paths, use discount to calculate trust of each path, then consensus them all together? But what about cycles? And this is a computational nightmare, especially for well connected graphs!

Instead, we derive a Direct Series Parallel Graph (DSPG). A DSPG is a graph created using the two operations below.



DSPG (2)

Why do we want a DSPG? Notice the construction grammar can be represented precisely by the discount and consensus operators. Thus, if our trust dependency graph is a DSPG, a trust expression with SL-operators will be a terse canonical expression



The graph at left cannot be constructed as a DSPG. So how do we go about getting one?

1. Enumerate all paths
2. Rank that set of paths by confidence
3. Add paths (or branches of paths) to the DSPG assuming their addition does not violate DSPG constraints, per the order determined in (2).
4. Once done, calculate trust via the canonical expression that can be derived for our DSPG.

The result? The DSPG that maximizes confidence (limits uncertainty), and therefore is the most accurate aggregation possible

TNA-SL

- What TNA-SL gives us
  - A user-centric calculation with transitive discount
- Advantages of TNA-SLA
  - Trust values have absolute interpretation on [0...1] (beta-PDF)
  - Negative trust is representable via the 'disbelief' field
  - User-centric views allow us to disregard malicious opinions
- Disadvantages of TNA-SL
  - Lacks scalability. Connected? 2^N paths to enumerate!
    - Plus these SL-ops are hefty operators.
  - Transitivity weakens trust; could be bad in very sparse NW
    - Good info several hops away could be discounted to oblivion

P2P-Sim Intro

- So how to test and compare and these systems?
  - EigenTrust runs simulations, using a closed-source sim, and the TNA-SL paper is too theoretical to do anything of the sort.
- We propose (and have implemented) a trace-driven simulator built in the P2P-style (C & Java)
  - New trust/reputation management algorithms can be included via a simple calling interface
  - Traces can be-run using different TMs allowing comparison

Traces

- Traces contain 4 different data sets:
  - 1. The header – Contains command-line arguments like # users, # transactions, and other sizing parameters.
  - 2. User initializations – Triples *(u, c, h)*, stating user *u* cleans up invalid files *c%* of the time, and provides honest feedback *h%* of the time. See Table 1 for user types.
  - 3. Library initializations – Triples *(u, f, v)*, stating user *u* has file *f* in his/her initial lib with validity *v* (a Boolean).
  - 4. Static transactions – Pairs *(u, f)* stating user *u* wishes to obtain file *f*.

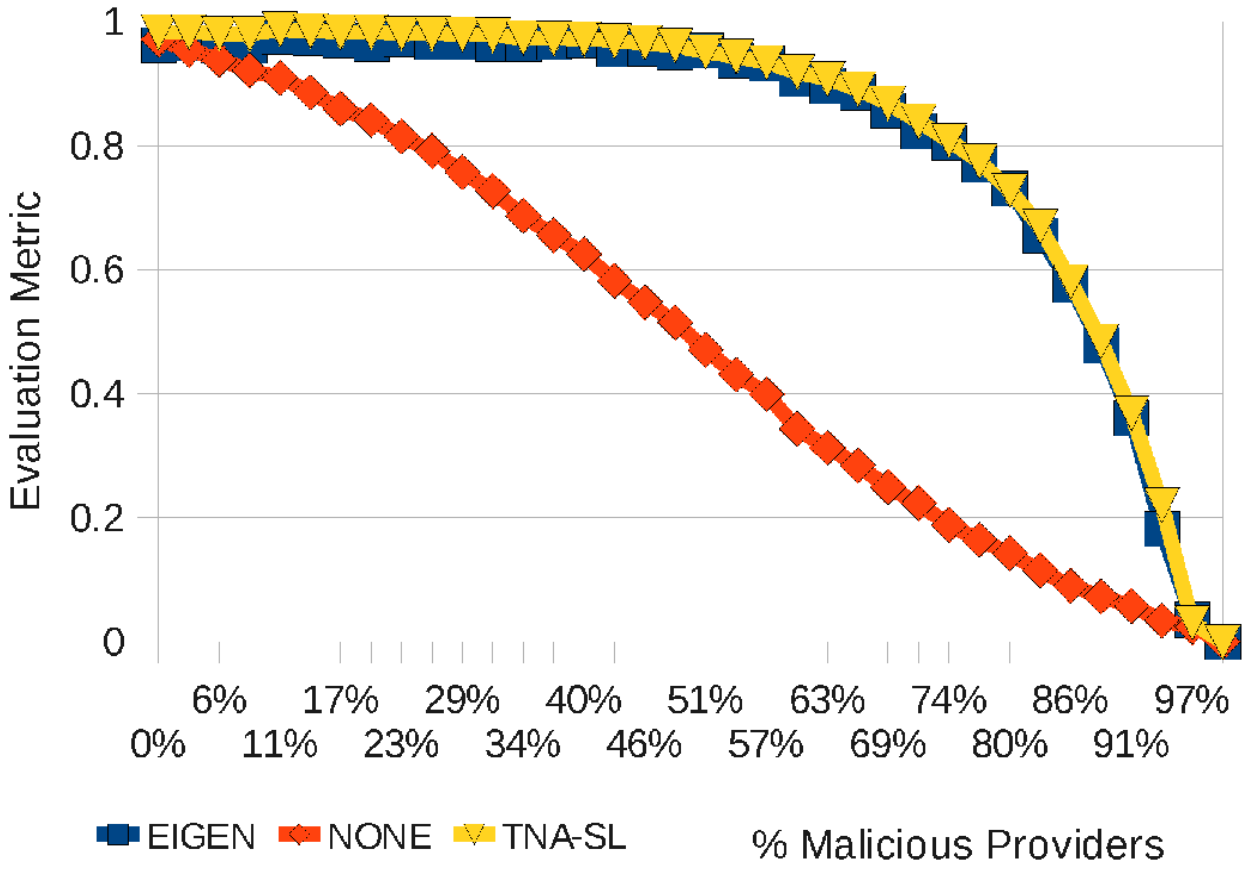
User Type	Cleanup%	Honesty%
Good	90-100%	100%
Purely Malicous	0-10%	0%
Malicious Provider	0-10%	100%
Feedback Malicous	90-100%	0%
Disguised Malicous	50-100%	50-100%
Sybil Attacker [2]	0-10%	Irrelevant

Table 1: User initialization parameters

Simulator

- While there are more transactions:
  - 1. Parse a file request from trace file.
  - 2. If requester has available DL bandwidth, proceed...
  - 3. Requester computes trust values for other NW peers
  - 4. Requester publishes file query to NW
  - 5. Using trust-values, requester source-selects from UL-bandwidth available peers who answered query
  - 6. Copy source file to requester library
  - 7. Requestee provides feedback on source

Sanity Check



$$Metric = \frac{\#\text{ valid files received by 'good' users}}{\#\text{ transactions attempted by 'good' users.}}$$

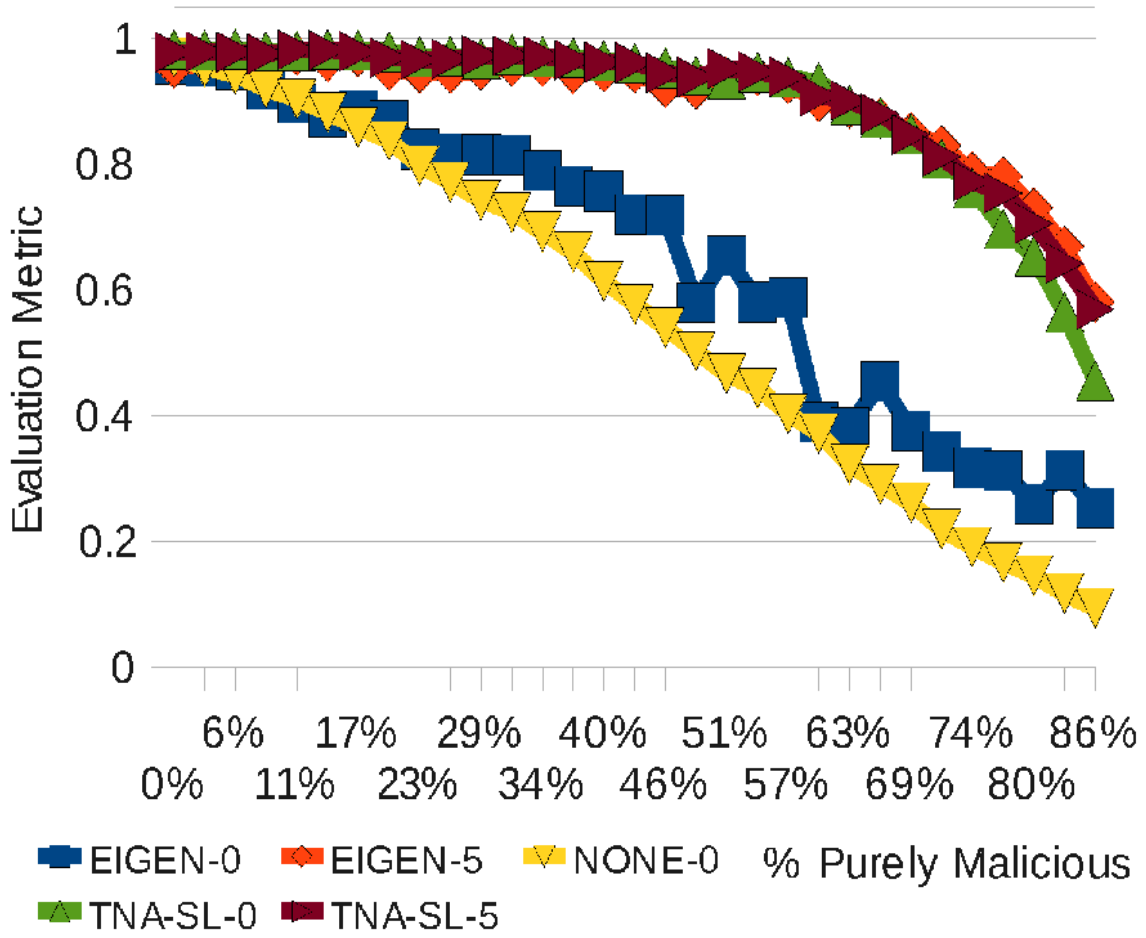
In this example...

Malicious providers keep invalid files, but they are completely honest about their malignancy, and that of others.

This is a trivial system to manage. Really just a sanity check on implementation.

Note: "None" is a control line demonstrating the lack of a trust system.

Malicious & Pre-Trust

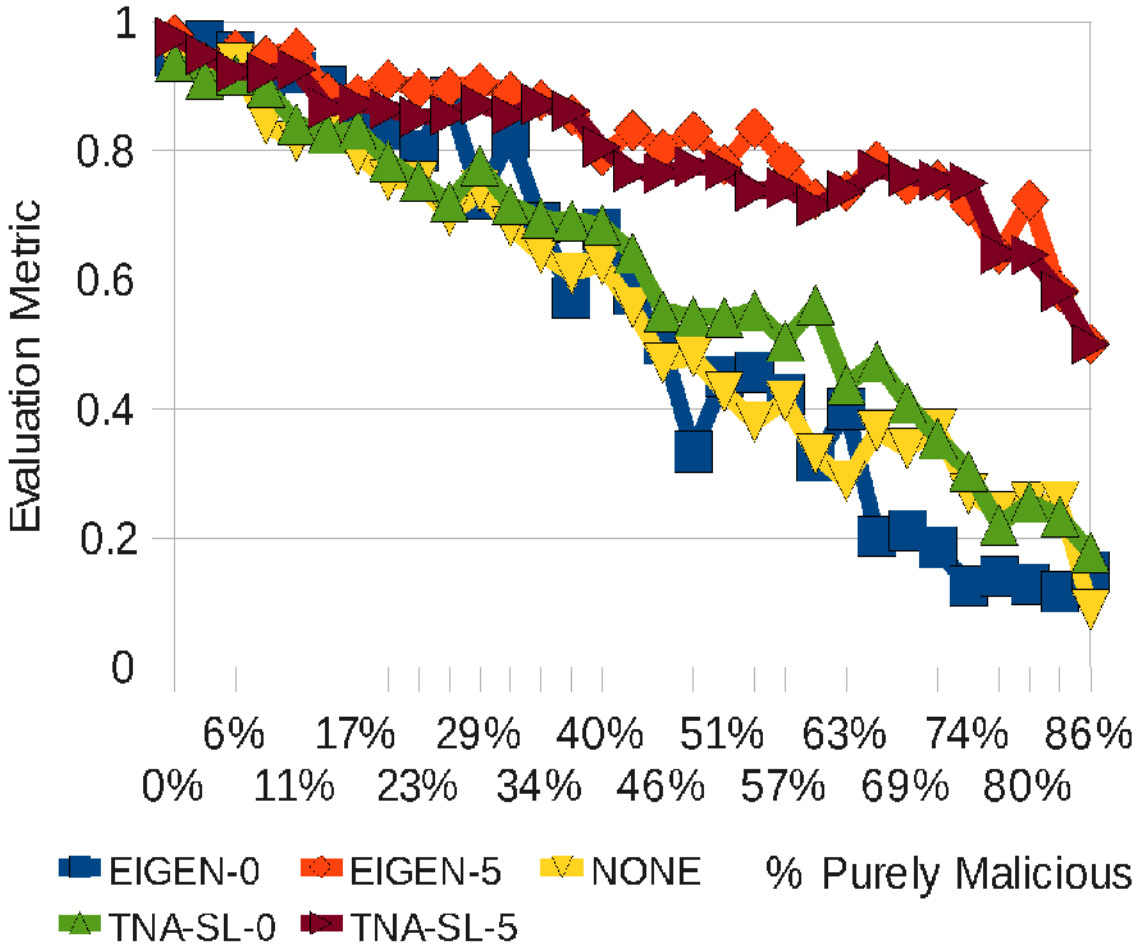


In this example...

We see “purely malicious” users, they keep invalid files (try to get them), and consistently provide dishonest feedback.

TNA-SL (with its user centric views) passes with flying colors on all accounts. EigenTrust requires the inclusion of pre-trusted users to get good performance

Very Sparse!

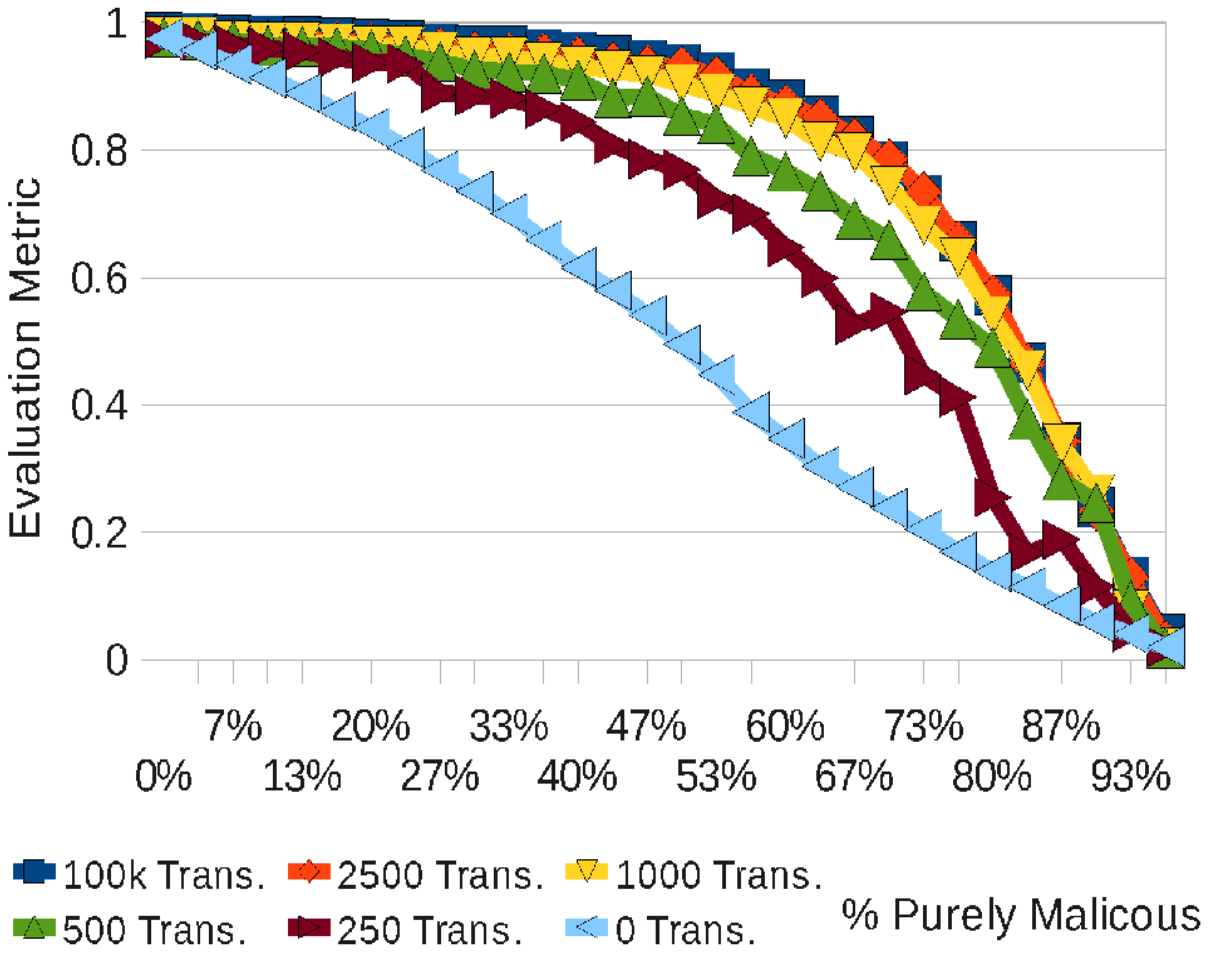


In this example...

So far we've been seeing graphs with LOTS of transactions; showing convergent behavior.

Real P2P situations are much sparser. We see that here. With less data, both systems are reliant on the notion of pre-trust to attain good performance

Quick Convergence



In this example...

We see that because users generally (definitely, in the case of our simulator) behave consistently, trust values converge very quickly.

We've used this fact to create speedup strategies so we can overcome complexity issues (and thus experiment with larger networks).

Conclusions

- Concerning the P2P-Simulator
  - Solid useful tool. Implementation increased understanding.
  - Simulator weaknesses:
    - Closed world – Users don't come and go. New files don't appear.
    - Uniform distro – Would be nice to fit distributions to actual P2P data
    - Weaknesses? We wanted to minimalize parameterization, keep static
  - Malicious model weaknesses:
    - Does it make sense a NW with 75% bad users can still be managed?
    - To test systems more effectively, malicious users need more power
    - Foremost, bad users should act collectively, not just in isolation
- Trust/Reputation Systems
  - There are bunches. We'll never say which is “best” but perhaps we can examine on a situation basis