Metadata, Moderation & Vote Sampling for improved search

Plus the bigger picture….

The big picture

- Closed tracker sites (like TvTorrents) work really well – why?
  - Identities are not cheap (but not so expens.)
  - Ratio enforcement (but not difficult to hack)
  - Moderation with high quality meta-data on a webpage only viewable by authenticated ID’s
  - Run own tracker which only authenticated identities can use (-ish)

The big picture

- Can this be done in a fully distributed way?
- If so we need:
  - Distributed ratio enforcement
    - BarterCast (long-run), Give-to-get (short-run)
  - Distributed moderation and metadata
    - ModerationCast + VoteCast
  - Distributed tracker
    - DHT-based (Kashmir), gossip-based? (littleBird, torrentSmell)

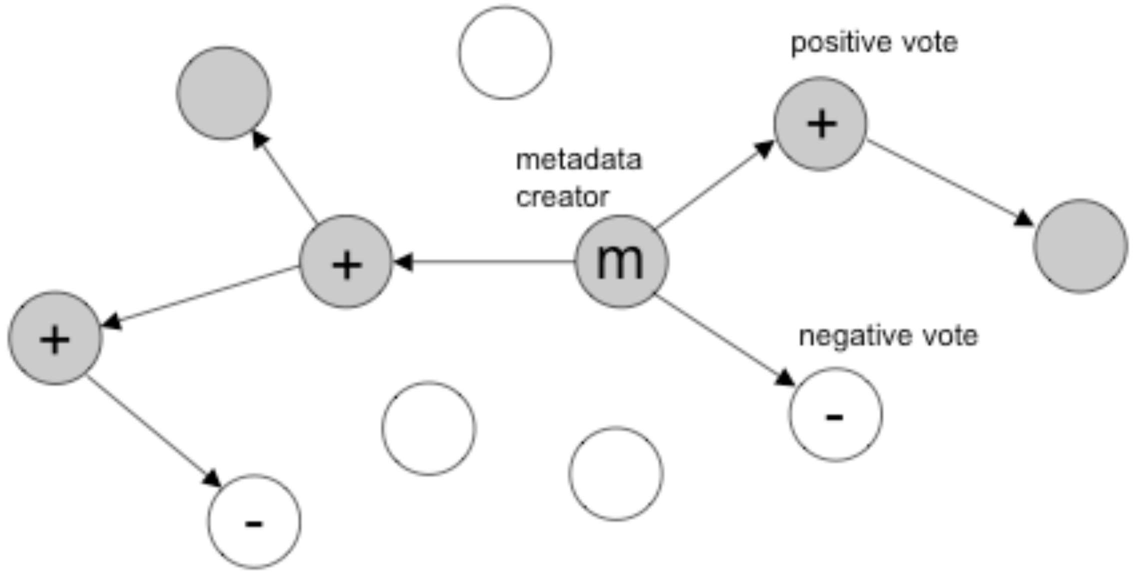
Talks

- I will talk about a design for Voting on moderations (VoteCast)
  - Already a paper on this with sim. results
  - Now in the process of implementation
  - Should be ready to deploy end of Nov.
- Victor will talk about some ideas on a gossip-based distributed tracker (TorrentSmell)
  - There are notes on the wiki

Assumed Protocols

- We assume that BuddyCast (a PSS), BarterCast (a reputation system), and ModerationCast (Meta-data spread) exist
- Currently moderationCast is not fully implemented – some code from Vincent
- I am not going to talk about BuddyCast or BarterCast

ModerationCast Overview



VoteCast problem

- Given there are moderations in my localDB
- Can I rank them in some way based on how other nodes have voted on Moderators
- Hence can I determine for a given moderator how many +votes and -negative votes there are in the population bound to them
- Need to do this in a distributed way, which is not easily (i.e. a few nodes) colludable, such that simple spam can be prevented

VoteCast

- VoteCast is composed of two subprotocols
  - BallotBox and Voxpopuli
- First we'll talk about BallotBox
  - Gossip-based – requires random pairings
  - Push gossip – spread your votes to others
  - Local state update - count of +ve and –ve votes against a list of moderators
  - One node one vote (per moderator) principle
  - Every node is conducting it's own ballot by receiving votes from each new random node it encounters. Hence sample the population

BallotBox - stopping bad guys

- But since identities are cheap in Tribler, what would stop a kind of Sybil voting attack?
  - One node could create a million identities and vote up their own spam
- Here we use BarterCast to supply us with an estimate of the upload flow to a node
- Only nodes that are above some Threshold get their votes counted
  - We define an experience function *E(i)* which will tell you if node *i* is experienced or not (binary)

BallotBox Pseudocode

```
do forever
  wait $\Delta$
  j ← GetRandomNode()
  Send $vote\_list_i$ to j
  $vote\_list_j$ ← Receive(j)
  if $E_i(j)$ = *true*
    ballot_box ← Merge(ballot_box, $vote\_list_j$)
```

(a) BallotBox

```
do forever
  $vote\_list_i$ ← receive(*)
  Send $vote\_list_j$ to i
  if $E_j(i)$ = *true*
    ballot_box ← Merge(ballot_box, $vote\_list_i$)
```

(b) BallotBox passive thread

Experience

$$E_i(j) = \begin{cases} true & \text{iff } f_{j \rightarrow i} \geq T; \\ false & \text{otherwise.} \end{cases}$$

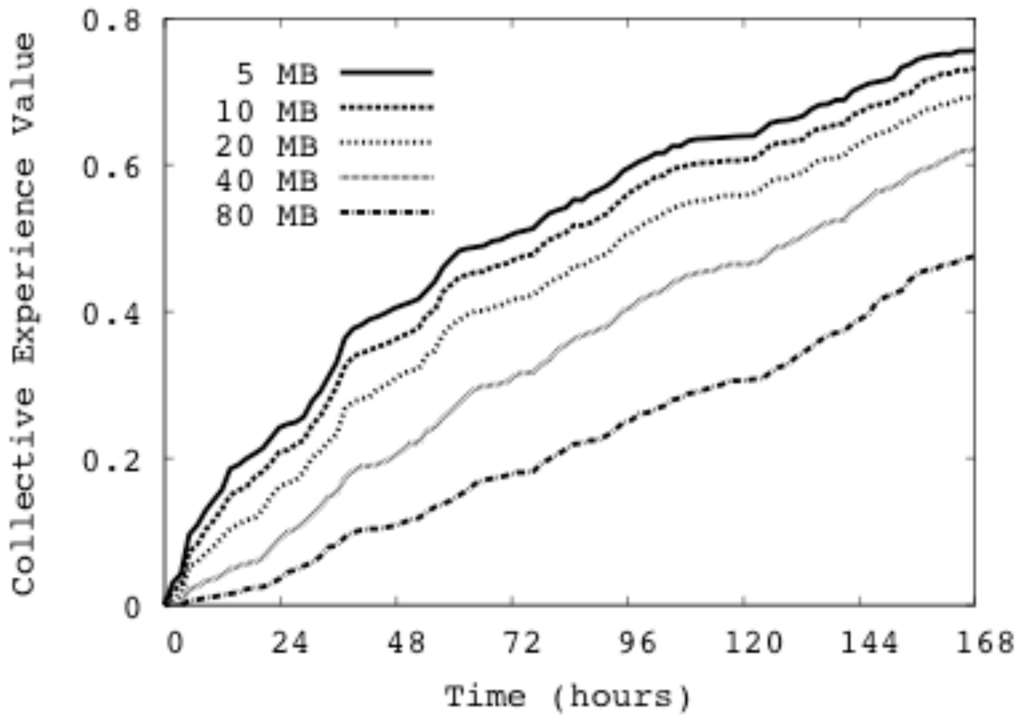
$$e_i(j) = \begin{cases} 1 & \text{iff } E_i(j) = true; \\ 0 & \text{otherwise.} \end{cases}$$

$$CEV = \frac{1}{N} \sum_{i \in N} \sum_{j \neq i} \frac{e_i(j)}{N-1}$$

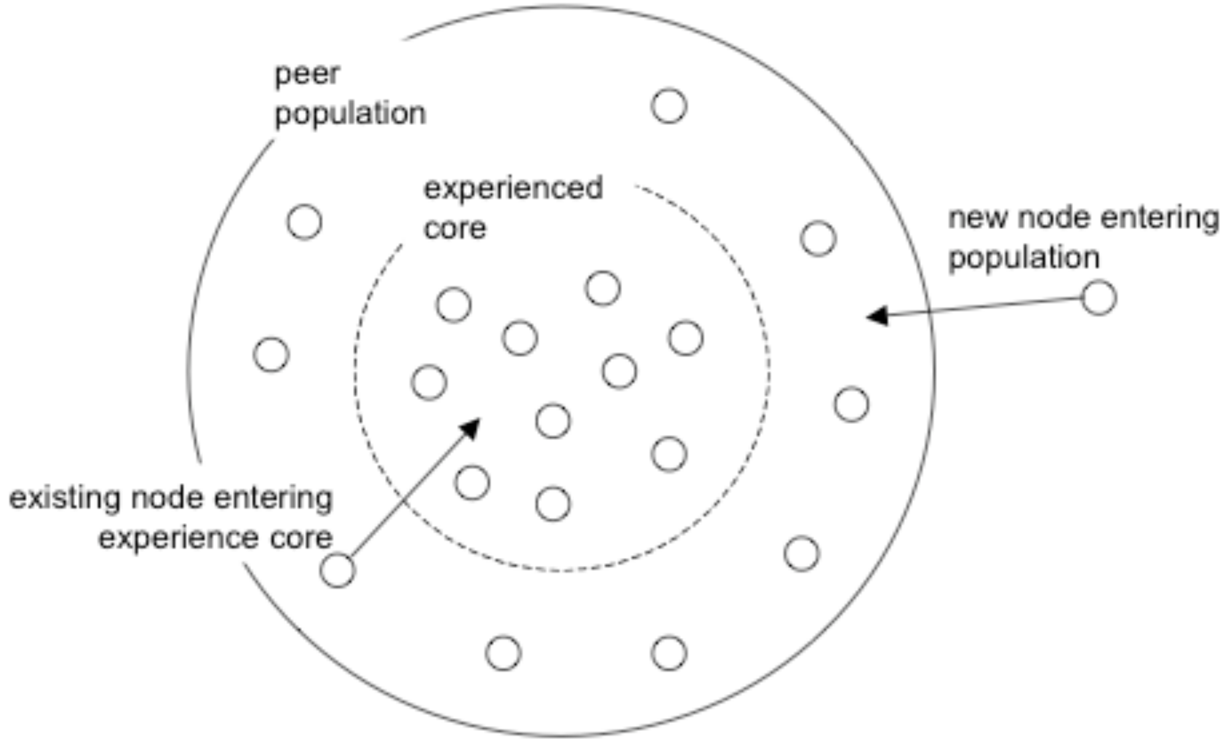
BallotBox - what should T be?

- What should our threshold value T be set to?
- We don't know – how much is a bad guy prepared to pay in upload to get an identity?
- So we just picked a few arbitrary values and ran some sims on traces
- What we wanted to see is nodes getting experienced quickly – but not too quickly
- The time to become experienced is a kind of cost controlled by T.

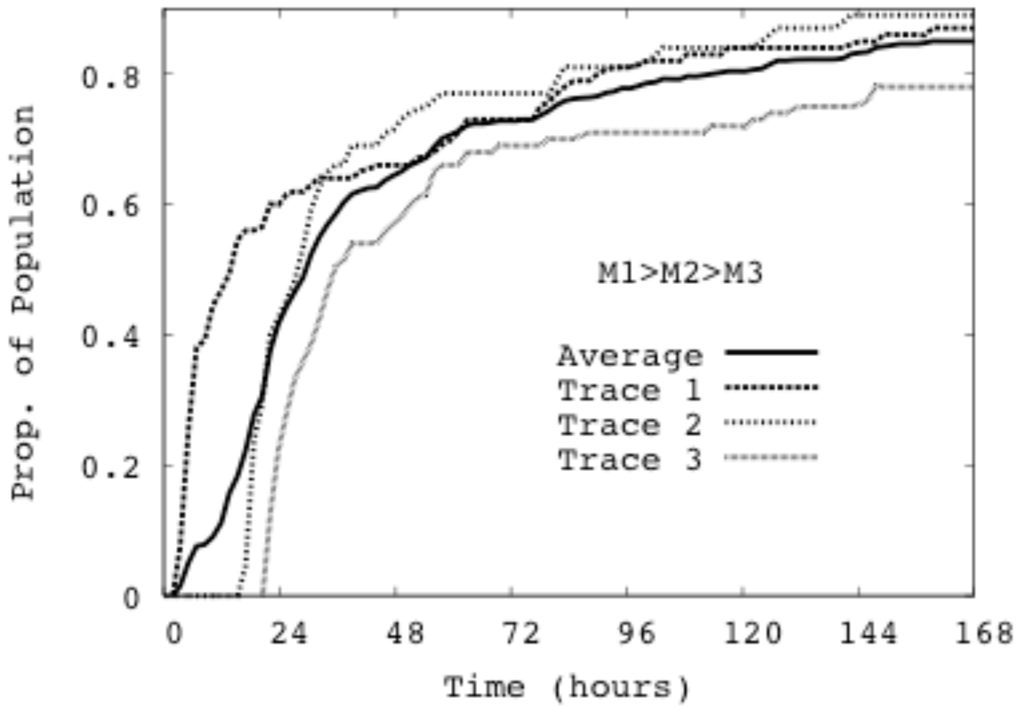
How quickly does experience grow?



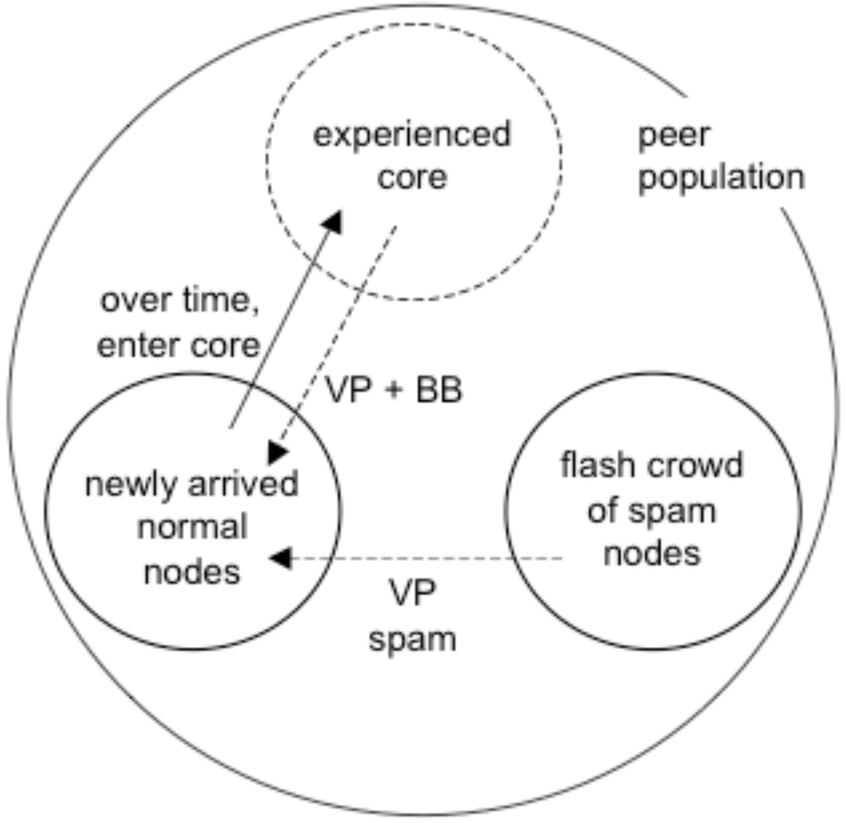
Experienced core



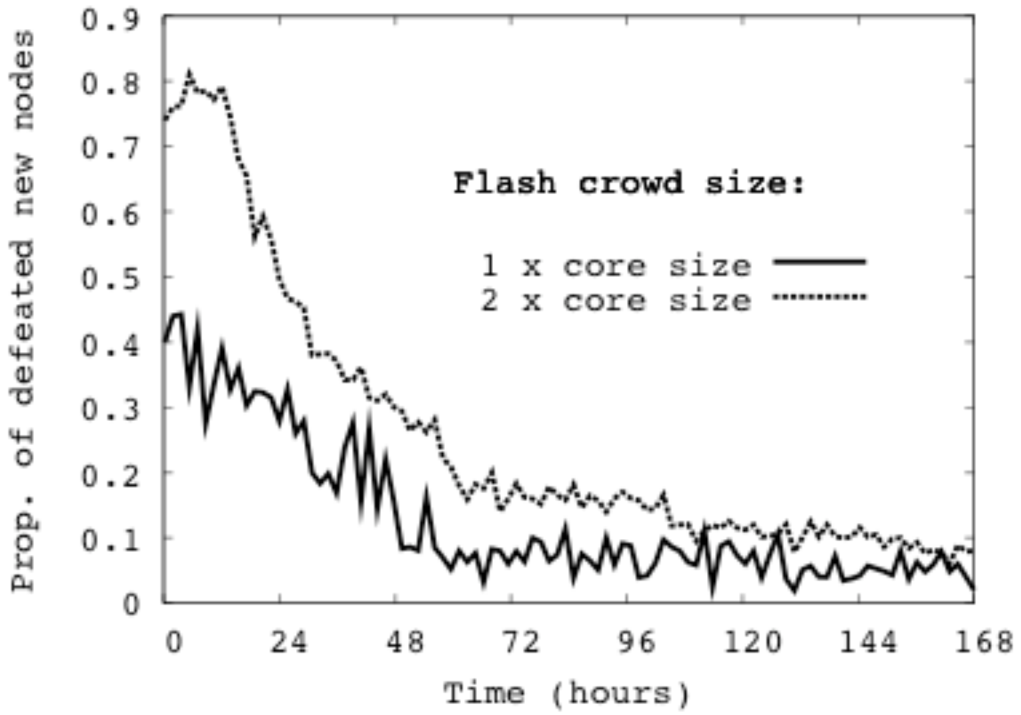
Without bad guys



Spam attack


experienced core
peer population
over time, enter core
VP + BB
newly arrived normal nodes
flash crowd of spam nodes
VP spam

Spam attack



Prop. of defeated new nodes
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
24
48
72
96
120
144
168
Time (hours)
Flash crowd size:
1 x core size
2 x core size

Vox populi

- New nodes have to wait a while to build up experience from others
- Hence during that time they they can’t count any votes
- Vox populi is an extra protocol that “fills-the-gap” while waiting for experienced votes
- Simply asks random nodes for their top-K moderators and takes average

BallotBox + VoxPop

```
do forever
  wait $\Delta$
  j $\leftarrow$ GetRandomNode()
  Send vote_list$_i$ to j
  vote_list$_j$ $\leftarrow$ Receive(j)
  if E$_i$(j) = *true*
    ballot_box $\leftarrow$ Merge(ballot_box, vote_list$_j$)
  end if
  if num_unique_users(ballot_box) < B$_{min}$
    Send VP_request to j
    topK$_j$ $\leftarrow$ Receive(j)
    topK_cache $\leftarrow$ Merge(topK_cache, topK$_j$)
  end if
```

(a) BallotBox and VoxPopli active thread

```
do forever
  vote_list$_i$ $\leftarrow$ receive(*)
  Send vote_list$_j$ to i
  if E$_j$(i) = *true*
    ballot_box $\leftarrow$ Merge(ballot_box, vote_list$_i$)
```

(b) BallotBox passive thread

```
do forever
  VP_request$_i$ $\leftarrow$ receive(*)
  if num_unique_users(ballot_box) $\geq$ B$_{min}$
    topK$_j$ $\leftarrow$ Rank(ballot_box)
    Send topK$_j$ to i
  else
    Send null to j
  end if
```

(c) VoxPopuli passive thread

Open Issues

Adaptive T?

- We selected our T in a very arbitrary way based on small old traces
- Would make sense to adapt T to the environment
- If it appears spammers are obtaining Experience E()=1 then increase T otherwise decrease T?
- How can we do this?

Adaptive T?

- First simple idea:
  - If when receive new votes the variance of votes increases then inc T
  - If it decreases then dec T
  - But how to measure variance?
  - How much to inc and dec T?
  - Do we have upper and lower thresholds too?
- Any ideas or hunches?

Stopping “Front” or “Mole” attacks

- BarterCast uses a maxflow algorithm
- This is vulnerable to a so-called “Front” attack
  - One node builds-up high experience by uploading
  - The colludes with other identities
  - Allowing those identities to appear experienced
  - A clever spam node could do this incurring only the cost of getting one identity experienced
- How to stop this?

Stopping Front attacks

- Maybe by modifying the way maxflow works we can limit such attacks (by dividing flows over siblings?)
- By using a distributed social network where new nodes are invited by friends into the system a given credit (like TvTorrents – avoid whitewashing)?
- These ideas need thinking through…