Epidemic Techniques

Milo Polte

Summary of First Paper

- *Epidemic Algorithms For Replicated Database Maintenance (Demers et al. Proc. of the Sixth ACM Symp. on Principles of Distributed Computing, August 1987)*
  - *Presents randomized, epidemic algorithms for distributing updates in a replicated database to approach consistency*
  - *Analyses performance of two random epidemic algorithms (anti-entropy and rumor mongering)*
  - *Implements algorithms in simulation and on Xerox Corporate Internet to measure rate of database consistency and network traffic*
  - *Emphasizes importance of spatial distributions for efficiency*

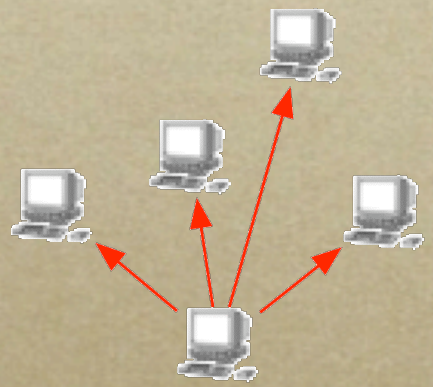
Summary of Second Paper

- *Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining (Van Renesse et al. ACM TOCS, May 2003)*
  - *Describes the distributed hierarchical database system, Astrolabe*
  - *Uses epidemic techniques to efficiently propagate through the hierarchy and achieve consistency*
  - *Presents an SQL-like language for complicated aggregation of data*
  - *Incorporates a certificate authority based security model*

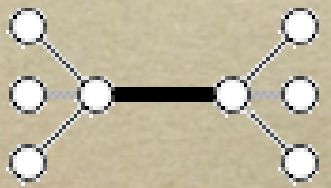
Problem:

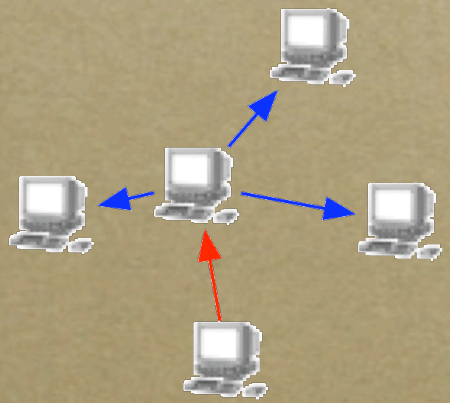
- *How do we replicate a database across many sites while maintaining consistency?*
  - *Many different hosts may have write access to the database*
  - *Underlying network is unreliable*
  - *We want to avoid unnecessary network traffic*

Two unsuccessful approaches:



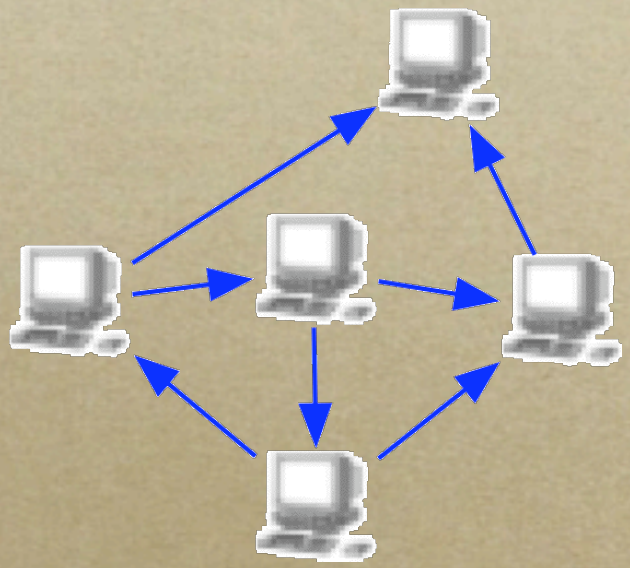
- *Each host responsible for propagating their updates directly to all other hosts*
  - \+ *Updates propagated immediately*
  - \+ *No redundant messages sent*
  - \- *Each host must know full membership -- Difficult with churn*
  - \- *Messages may be lost*
  - \- *May saturate critical links*
  - \- *Forces updating node to make O(n) connections*

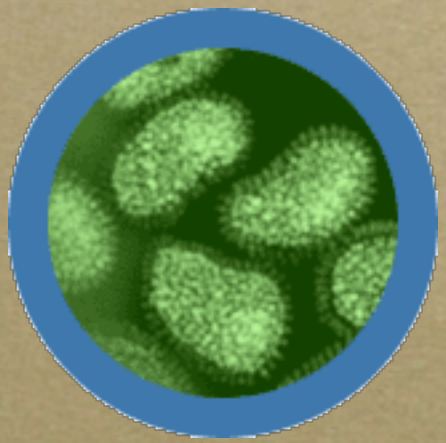




- *Use primary site update*
  - \+ *Simplifies update distribution*
  - \- *Single point of failure/Bottleneck*

An alternative approach:





- *Use peer-to-peer randomized algorithms to disseminate updates in the network like an epidemic*
  - + *Does not require full knowledge of network at any single host*
  - + *Works well with unreliable message delivery*
  - + *Updates spread rapidly as more sites become "infected"*
  - - *Harder to achieve consistency with randomized algorithm*
  - - *Reoccurring question: How do we avoid generating tremendous network traffic?*

Epidemic Methods

The first paper describes three techniques for update propagation:

1. *Direct mail - Each host sends all updates to every other host. Has same pros/cons of the first unsuccessful approach. Not epidemic.*

2. *Anti-entropy - Sites periodically contact other sites and reconcile database with them.*

3. *Rumor mongering - When a site encounters a new update, it begins to gossip it to random sites until the rumor becomes "cold" by some measurement (e.g. many sites contacted already knew rumor).*

Anti-Entropy

- *Sites pick random partner and exchange database content and resolve differences*
- *Operations referred to as "push", "pull", or "push-pull" depending on which direction updates flow*
- *Expected time for update to propagate to n hosts using push is logarithmic:* $\log_2(n) + \ln(n) + c$ (Pittel, 87)
- *push seems to be used more in practice (e.g. USENET) but pull will propagate updates more rapidly in settings where only a few sites initially do not have the update*
- *To keep deleted entries from re-propagating through the network, Death Certificates must be distributed and stored*

Compare Traffic

- *A naive anti-entropy algorithm exchanges entire databases to find differences, generating a prohibitive amount of "compare traffic"*
- *Solutions:*
  1. *Checksums*
     - *still exchanges entire databases when checksums differ)*
  2. *Maintaining window of recent updates which are always exchanged.*
     - *use checksums to compare databases after applying recent updates*
     - *Sensitive to choice of window size*
  3. *Exchange updates in reverse chronological order until checksums agree*
  4. *Other possibilities include recursive, hierarchical checksums of database or version vectors (e.g. Bayou)*

Rumor Mongering (Complex Epidemics)

- *A node that hears about an update considers it a "hot rumor"*
- *Nodes spread hot rumors to other random nodes*
- *At some point nodes consider rumor cold and stop spreading it*
- Problem: *Not all nodes may have heard rumor by the time it is considered cold.*
  - *Can be backed up with anti-entropy to achieve eventual consistency*

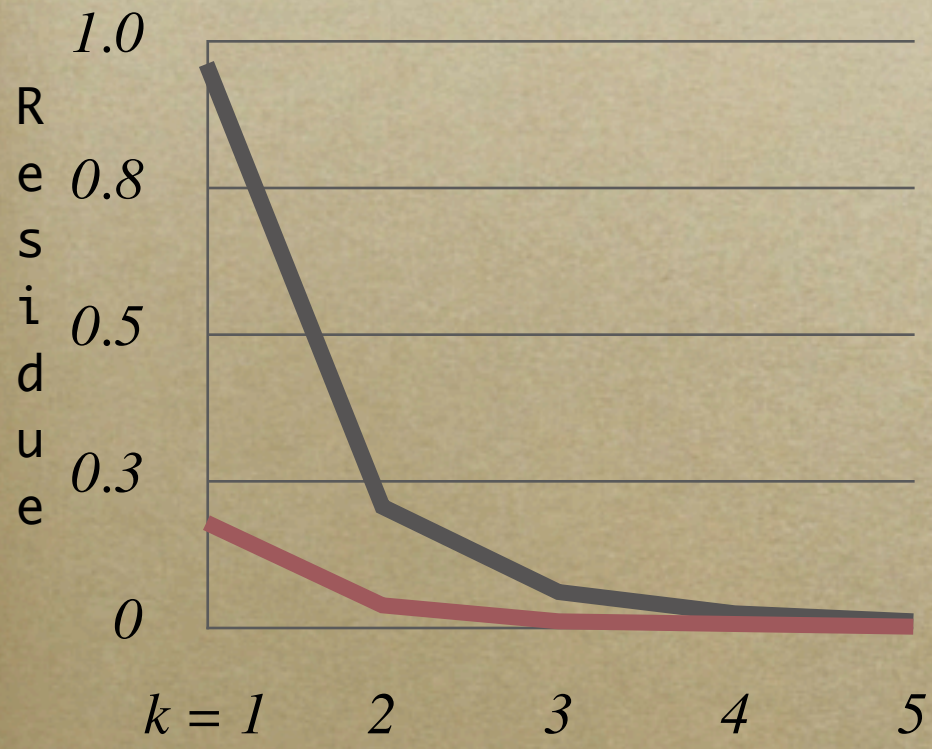
Deciding When to Stop

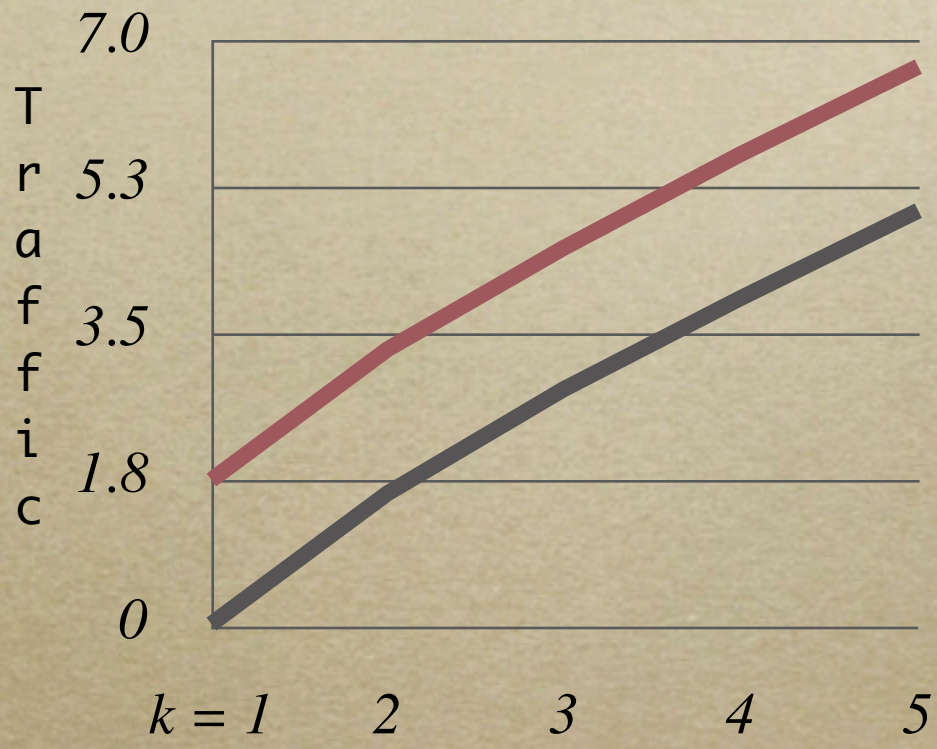
- *We want to design an epidemic which minimizes:*
  1. **Residue**, *the ratio of nodes susceptible at the end of the epidemic*
  2. **Traffic**
  3. **Delay** *until most sites know the rumor*

The first and third of these desires are in conflict with the second

Two different stopping policies compared:

Simulations on 1000 nodes

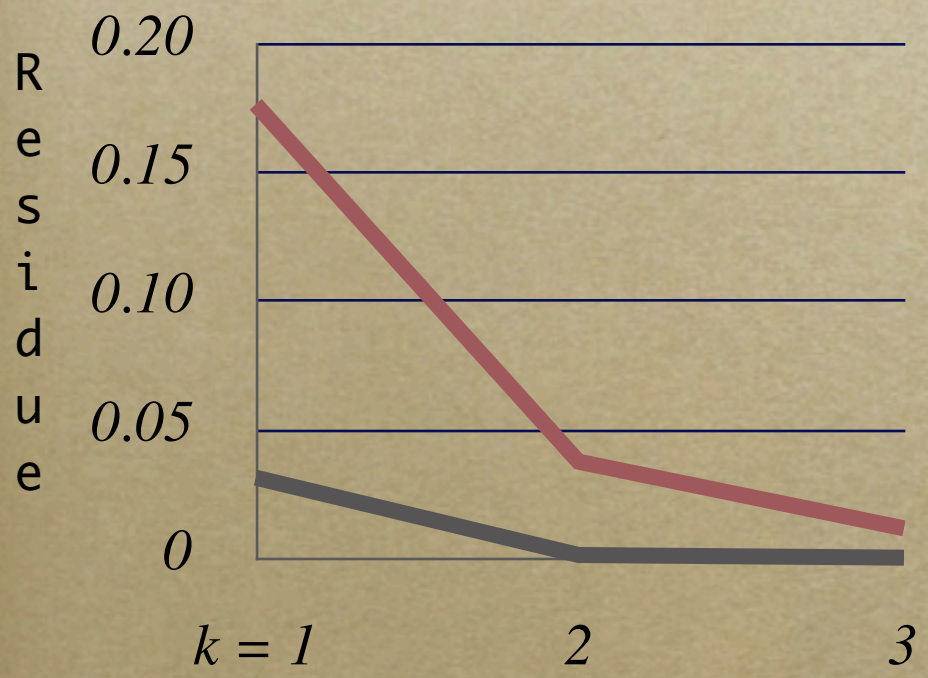


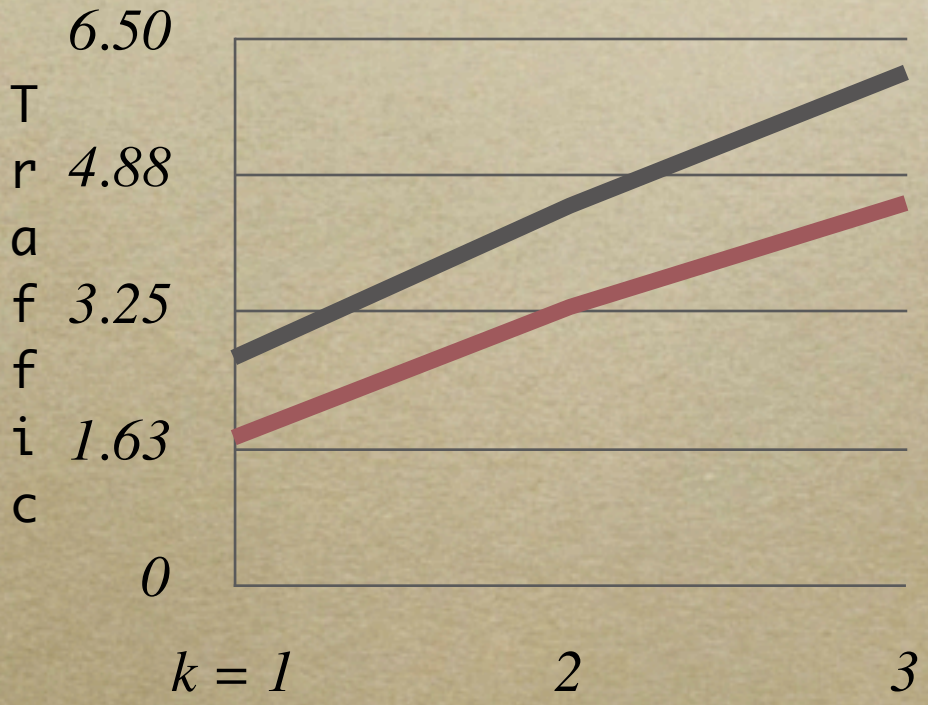


- Losing interest after contacting k recipients who already knew the rumor
- Losing interest with probability 1/k after ever cycle

Pulling Rumors

In a system with enough update traffic, it might be worthwhile to pull rumors instead for lower residue:





Pushing rumors

Pulling rumors

Motivation for Spatial Awareness

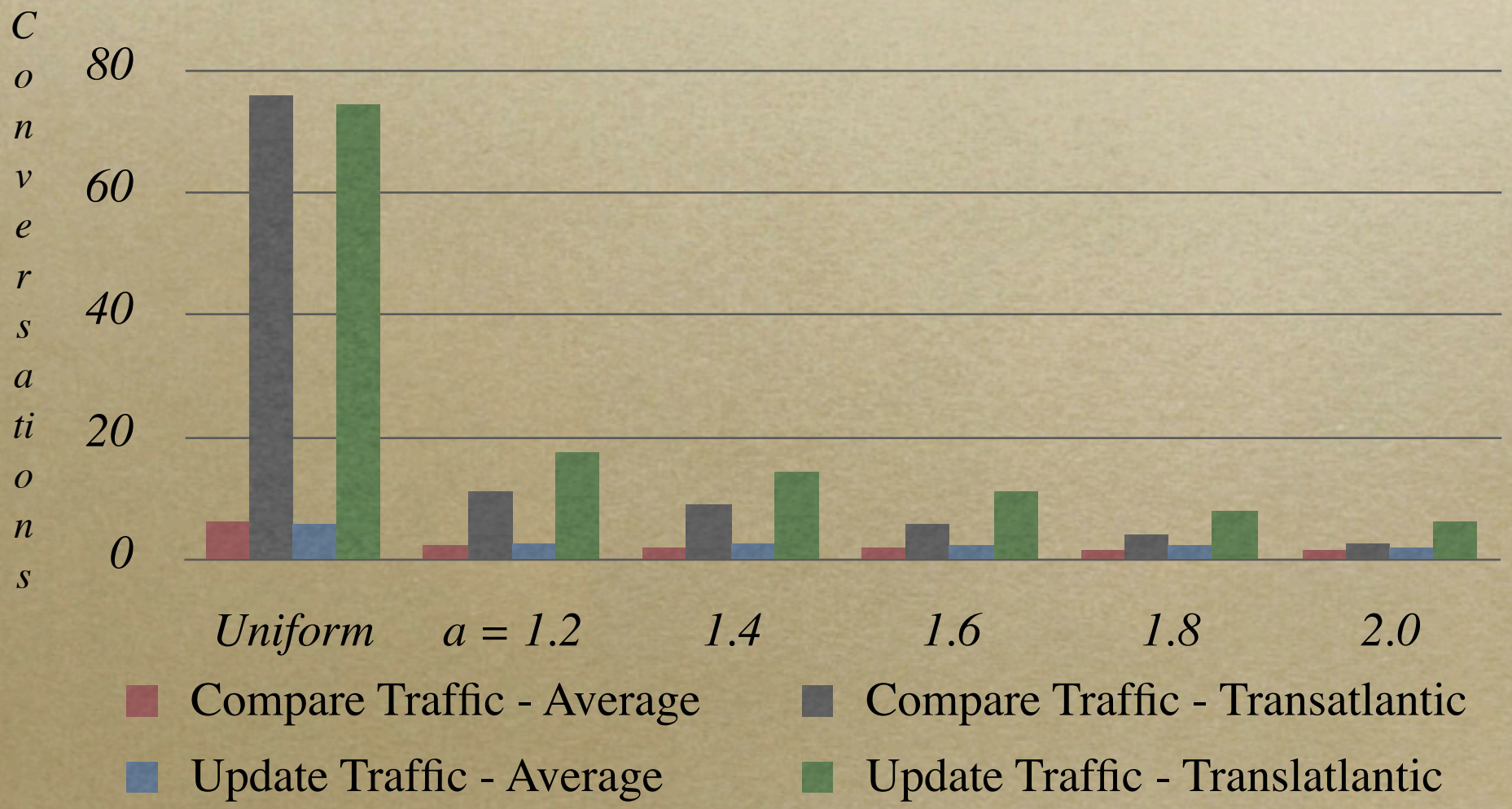
- *Clearinghouse name service*
  - *A translation database replicated on hundreds of servers on the Xerox Corporate Internet, a world wide network of thousands of hosts*
  - *Relied on anti-entropy with uniform host selection and direct mail to propagate updates*
  - *Found direct mailing was flooding the network but...*
  - ***Even without direct mailing, anti-entropy would saturate key links***

Spatial Distributions

- *Too much randomness seems unwise. We want nodes to infect nodes nearby them.*
- *Uniform selection of gossiping partners undesirable. Critical links in the network will face large traffic.*
  - *In CIN, key transatlantic links would have 80 conversations/round compared to the link average of 6 conversations/round*

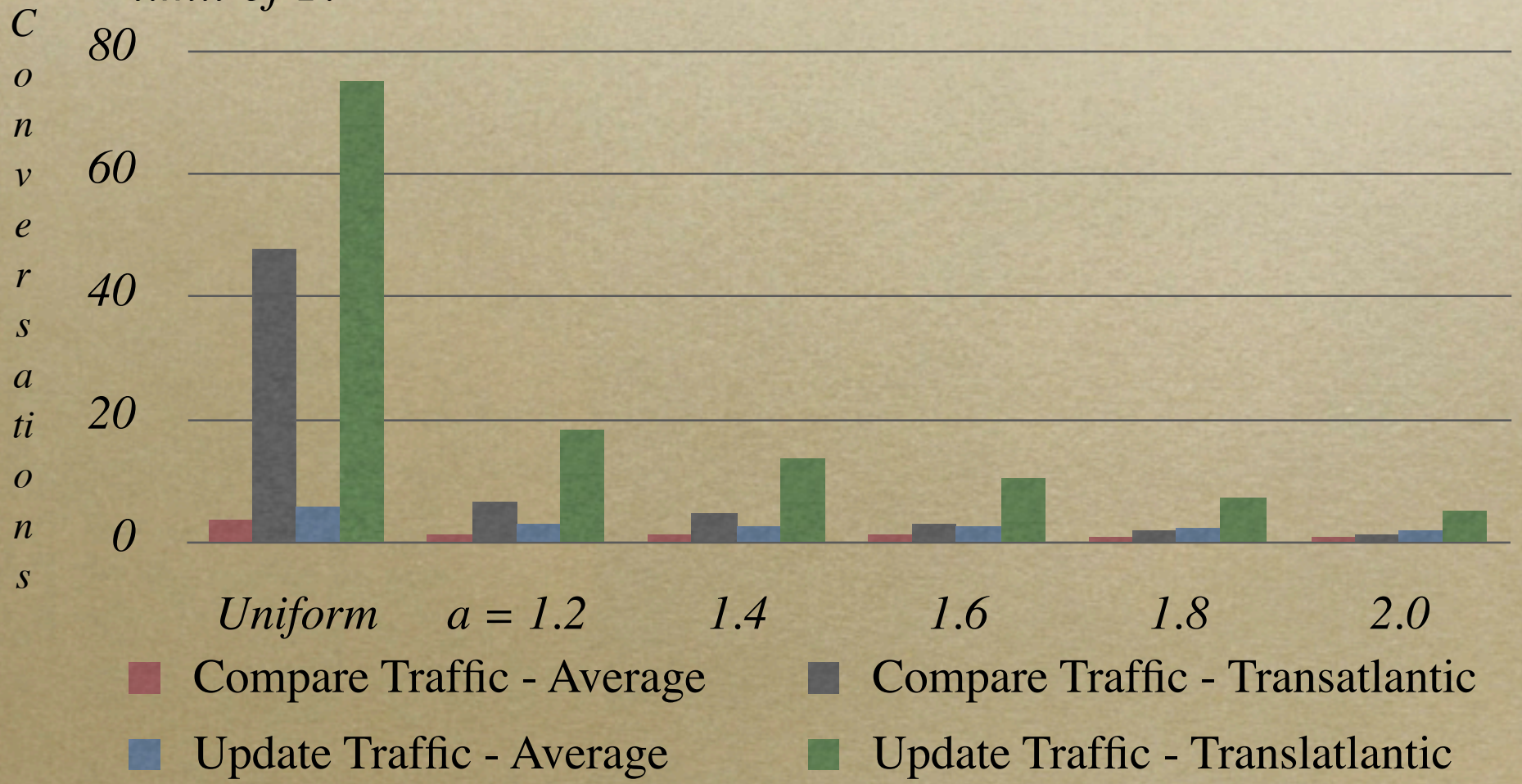
Incorporating Distance

- *Sites select gossiping partners with probability determined by the distance rank of the nodes and a parameter* a.



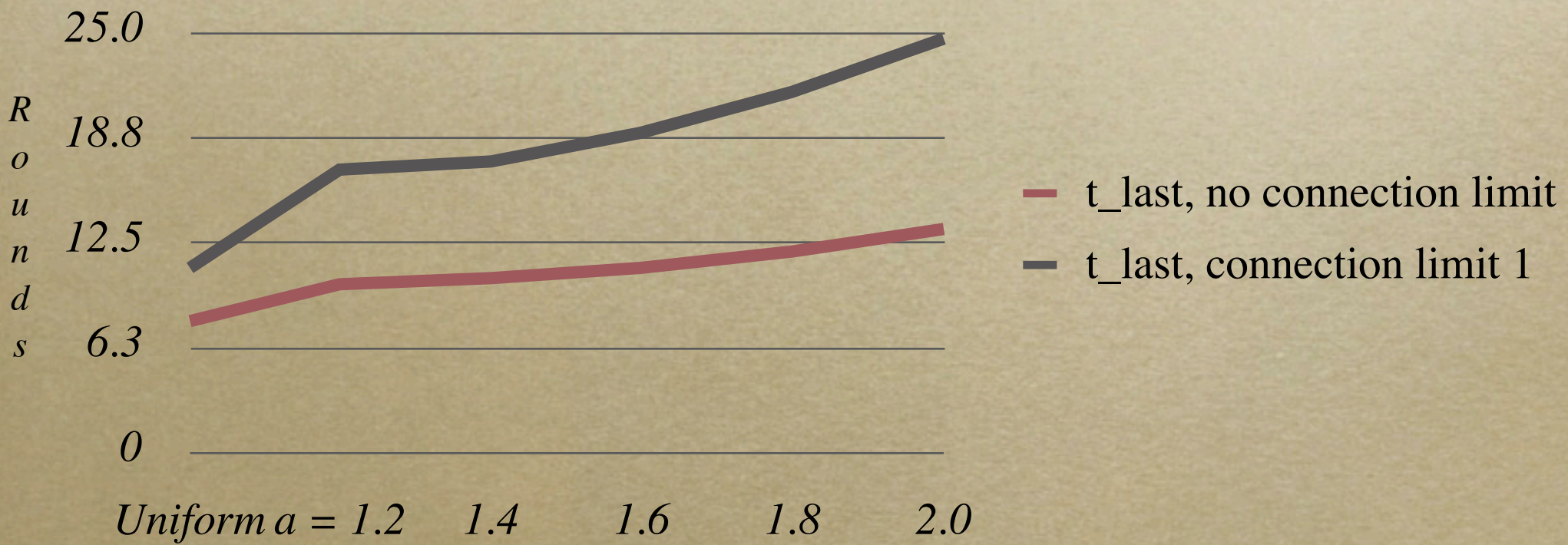
Incorporating Distance (cont.)

- *Sites select gossiping partners with probability determined by the distance rank of the nodes and a parameter* a *with connection limit of 1:*



Incorporating Distance (cont.)

- *While it seems that spatial information is critical for network load balancing, it does mean consistency takes longer to reach outer nodes:*



We have not escaped the trade-off between efficiency and consistency

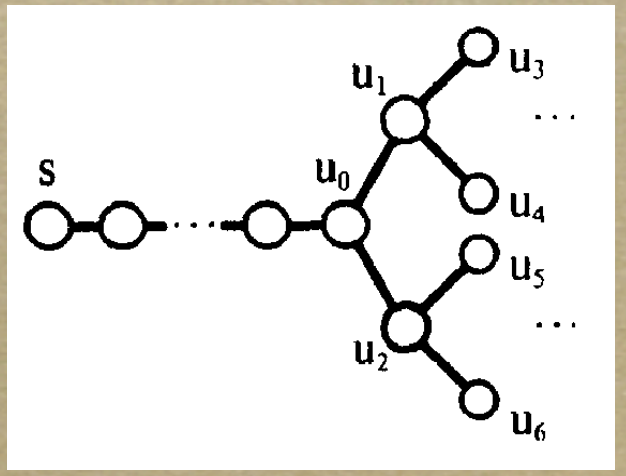
Rumor mongering sensitive to spatial parameters

Rumors may become cold everywhere before reaching distant nodes

Spatial Dist	k	t_{last}	t_{ave}	Compare Traffic		Update Traffic	
				Avg	Bushey	Avg	Bushey
uniform	4	7.83	5.32	8.87	114.0	5.84	75.87
$a = 1.2$	6	10.14	6.33	3.20	18.0	2.60	17.25
$a = 1.4$	5	10.27	6.31	2.86	13.0	2.49	14.05
$a = 1.6$	8	11.24	6.90	2.94	9.80	2.27	10.54
$a = 1.8$	7	12.04	7.24	2.40	5.91	2.08	7.69
$a = 2.0$	6	13.09	7.74	1.99	3.44	1.90	5.94

k *must be tweaked to reach consistency with tighter locality.*

The situation can become arbitrarily bad:



Summary of First Paper

- *The epidemic system we just examined replicates the entire database at all nodes*
- *Requires spatial distributions to efficiently spread information*
- *Spatial locality interferes with the rate at which we achieve consistency*
- *How does the systems scale as more more hosts (and updates) enter the system?*
- *Perhaps we can achieve better performance by replicating only summaries of data and propagating updates in a hierarchical manner....*

Astrolabe

A hierarchical, scalable distributed data storage and mining system

Four design goals:

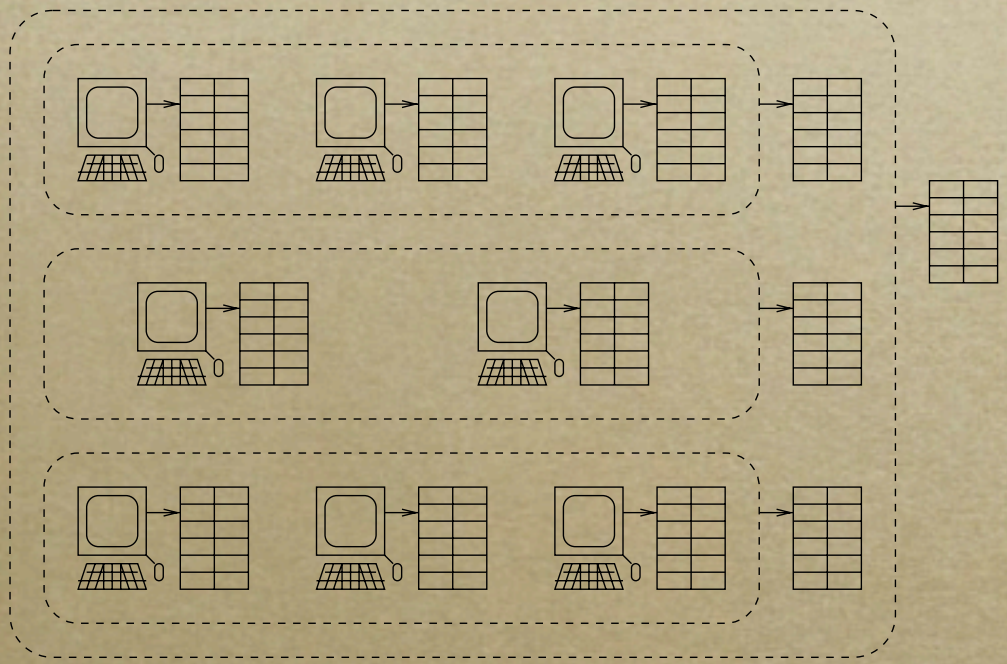
1. *Scalable* - Designed to be organized into hierarchical zones of close nodes. Data is summarized before moving between zones.

2. *Flexible* - Presents SQL-like code for aggregation functions. New functions may be added dynamically.

3. *Robust* - Hosts exchange information with a peer-to-peer epidemic algorithm resistant to host failures.

4. *Secure* - Certificate Authorities are used at each zone to control access to information and resources.

Turtles all the way down...



Individual zones maintain a management information base (MIB). Leaf nodes maintain local host information in 'virtual child zones'

Internal zones use aggregate function certificates to combine MIBs of their children into their MIB, and so forth, up to the root.

Smart Aggregation functions may propagate information through Astrolabe without replicating the entire database.

Aggregation -- The key to scalability

- *As more hosts are added to the system, and more information is stored, the number of updates and amount of data grows.*

- *In order to scale, this data must be locally aggregated before being propagated through the network.*

Aggregation (cont.)

- *Aggregation Function Certificates contain information on how to collect and aggregate attributes of child zone MIBs into entries for inner zone MIBs*
  - *Programmed in SQL-like language*
  - *Contain information on how the AFC should be propagated through the hierarchy*

Aggregation (cont.)

- *While computation and the inputs of an AFC may be complicated, it is important that their output is simple and scales. Sample AFC questions (taken from van Renesse, Birman, and Vogels's presentation):*

  1. Which are the three lowest loaded hosts?
  2. Which domains contain hosts with an out-of date virus database?
  3. Do >30% of hosts measure elevated radiation?
  4. Where is the nearest logging server?
  5.

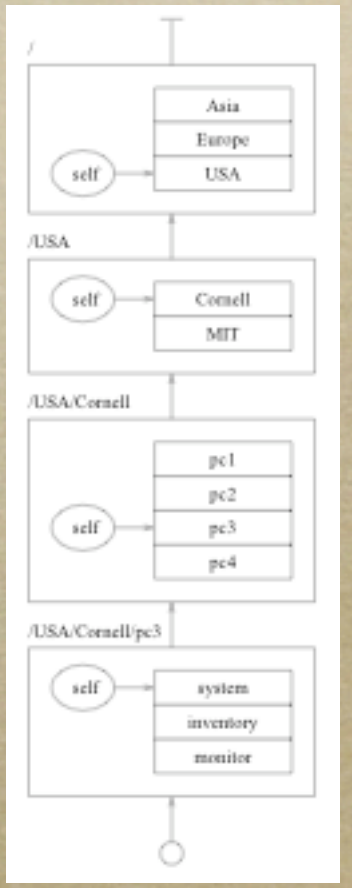
- *An invalid AFC question is something like* Which hosts have an nethack.rc file? *The result of this query increases with the number of hosts in the system. But we can be clever...*

Aggregate Propagation

- *An AFC may output itself. Since parents generate their MIBs by evaluating AFCs found in their children, the AFC will propagate up the hierarchy, so long as it is signed by an authorized client or an ancestor zone (see Security below)*
- *Children adopt new AFCs found in their ancestor zones*
- *The propagation of AFCs relies on timely gossip consistency of ancestor zones*

Robustness: Gossip

- *Each zone's MIB contains a list of contact agents for the zone, aggregated arbitrarily or though some voting mechanism*
- *Each agent periodically gossips for every zone for which it is a contact:*
  - *The agent picks a contact for another child zone and executes an anti-entropy protocol with them*
  - *The two agents push-pull to synchronize the contents of all child zone MIBs up to the root*
  - *The creator and time of creation of the MIBs recorded by the participating agents are exchanged to determine what records need to be updated.*



More Robustness

- *Astrolabe must also be robust in the face of network churn*
  - *Agents remove MIBs when they are not updated by the representative within some timeout*
    - *Entire zones are removed after all their MIBs are removed*
  - *New machines and split trees must be integrated*
    - *Trees must find each other using local broadcast, or IP multicast, or using a list of relatives :(*
  - *The administrator is responsible for assigning spatial significant zone names and relative lists.*
    - *Perhaps some other technique is possible. Perhaps relative lists and spatial information could even be gossiped.*
  - *Not designed for churn--Astrolabe processes can run on stable hosts. Other hosts can use RPC to talk to them.*

Security

- *Each zone has its own certificate authority:*
  - *Parent zones' CAs sign the public zone keys of child zones*
  - *MIBs are signed with the private key of the corresponding zone when gossiped*
  - *Client certificates specifying capabilities and are signed by a CA of any ancestor zone*
  - *AFCs are signed by ancestor zones or valid children to determine trustworthiness*

Security Issues

- *CAs must be well known and trustworthy. Compromising the CA for a zone lets a client write a client certificate for all descendent zones*
- *Nodes can lie about their values. Contacts for a zone (who know the zone's public/private keys) can lie about MIBs when gossiping. Depending on the election algorithm nodes can lie about their values in order to be elected as contacts. :)*
- *Certificates are hard to revoke, and with appropriate certificates, a client may install potentially expensive AFCs.*
- *What else?*
- *Has hierarchical design introduced more tiers for failure?*

Performance

- *For the data in Astrolabe to be timely, AFCs and MIBs must be propagated quickly.*
- *Higher branching factor = Faster propagation (fewer levels) and more overhead (more siblings mean more MIBs to gossip)*
- *More representatives at each level = Faster propagation (more gossipers) but also more traffic*

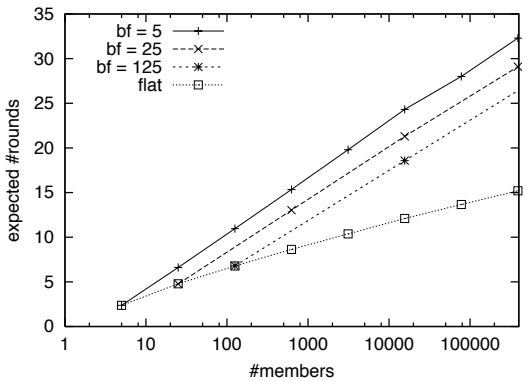


Fig. 6. The average number of rounds necessary to infect all participants, using different branching factors. In all these measurements, the number of representatives is 1, and there are no failures.

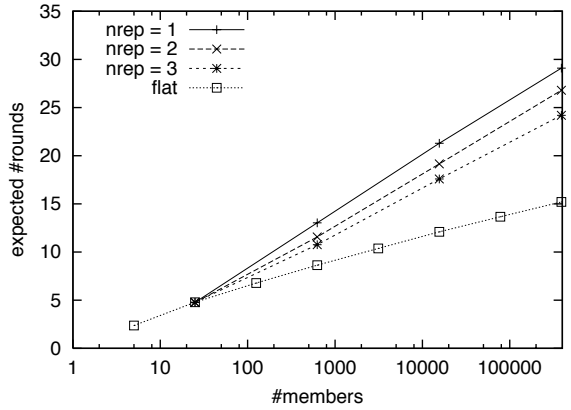


Fig. 7. The average number of rounds necessary to infect all participants, using a different number of representatives. In all these measurements, the branching factor is 25, and there are no failures.

Performance/Robustness

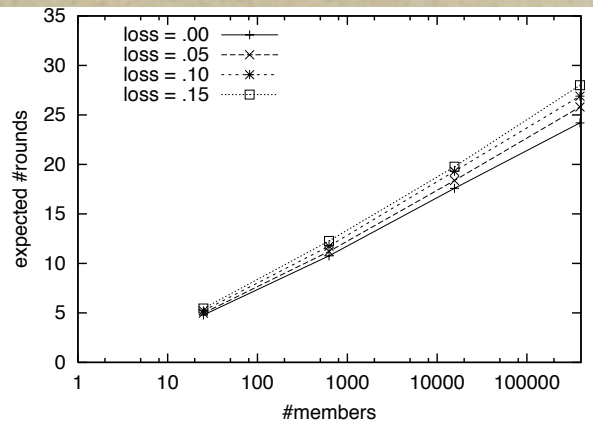


Fig. 8. The average number of rounds necessary to infect all participants, using different message loss probabilities. In all these measurements, the branching factor is 25, and the number of representatives is three.

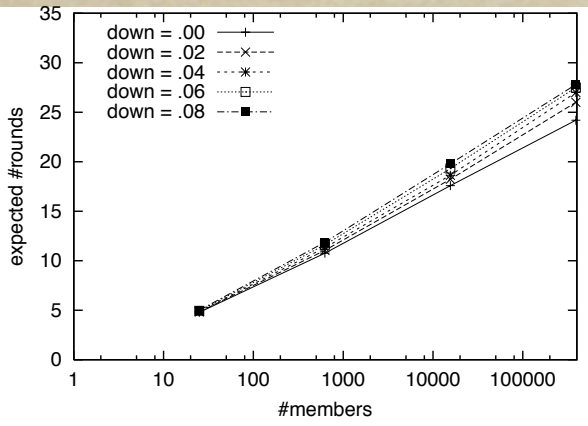


Fig. 9. The average number of rounds necessary to infect all participants, using different probabilities of a host being down. In all these measurements, the branching factor is 25, and the number of representatives is three.

- *In a gossiping protocol, the amount of gossip seems closely related to the robustness of the system.*
- *Need to compare effects of message lost and host failure to other Astrolabe configurations to be sure.*

Performance/Node Load

- *A higher branching factor means fewer messages/host (because they are representatives for fewer zones):*

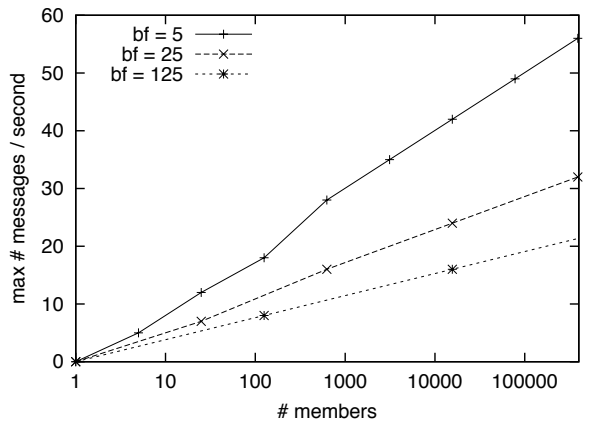


Fig. 10. The maximum load in terms of number of messages per round as a function of number of participants and branching factor.

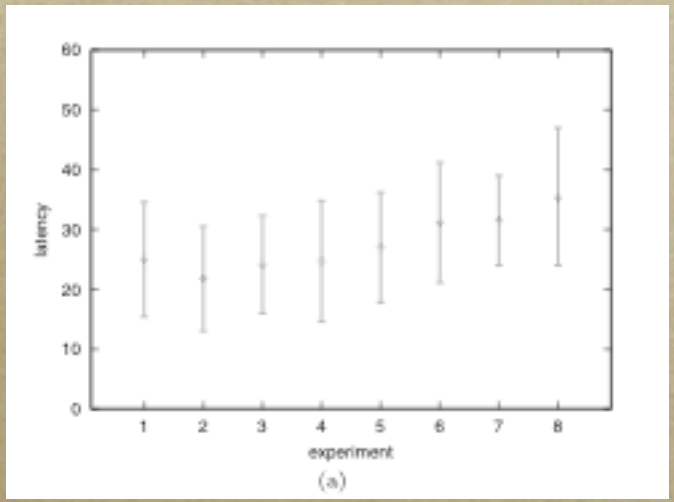
- *Unfortunately, the greater number of MIBs means more signatures must be checked. Can be done in the background, but PKC is computationally expensive*

Experimental and Simulated Latency

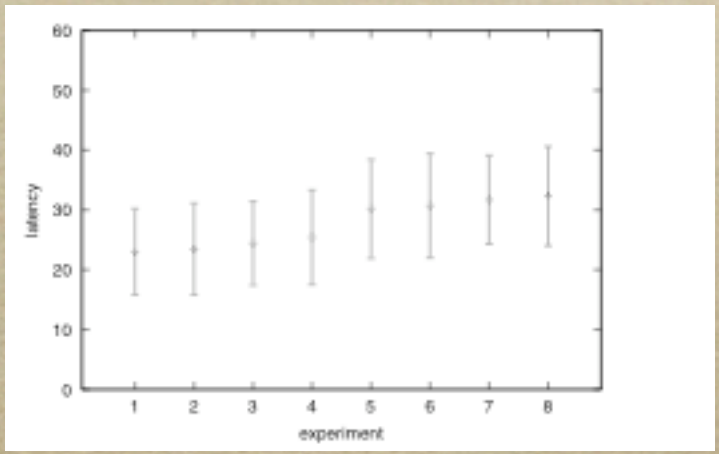
Are the previous measurements meaningful?

Compare time to propagate simple aggregate function through network:

Real:



Simulated:



Experimental results often slightly better *than simulated (gossiping works better when not synchronized). Do we buy it?*

Conclusions and Lessons

- *Randomized, epidemic algorithms are a useful tool for building information systems resistant to faults*
- *The spread of rumors, however, has scaling problems unless we organize the communication of hosts*
- *Organization of hosts can, however, begin to slow the rate at which the system achieves consistency and introduce new points of failure (i.e. Certificate Authorities, zone representatives)*
- *The key to epidemic algorithms then is not that they eliminate the problems of robustness versus efficiency, but that they provide us with many more points at which to tweak*