CS514: Intermediate Course in Computer Systems

Lecture 15: Oct. 18, 2003

Epidemic Protocols

(or, Gossip is Good)

Slides by Robbert Van Renesse

Up to now…

CS514

- We've looked at a few forms of replication
  - Hot standby, group communications systems, pub/sub architectures
- Or focus has been on relatively synchronized replication
  - and other strict properties, like ordering
- Its time for a change of pace!

Well, its still about replication . . .

CS514

- In fact, CS514 in a way is almost entirely about replication!
- But lets spend some time looking at weaker, less synchronous forms of "replication"
  - Perhaps better called "dissemination"

What is wrong with ISIS, Totem, Spread, etc?

CS514

- In a word, *scalability* (that is, they don't have much)
  - The lockstep nature of these protocols leads to a "weakest link" phenomenon … the slowest member dominates performance
  - Recall that ISIS deployment in French ATC was limited to groups of 5-6 machines over LANs

What is wrong with ISIS, Totem, Spread, etc?

CS514

- Furthermore, they are complex protocols, which speaks badly for fault tolerance
  - Complex software is more buggy
- And they are overkill for many applications
  - We just happen to be focusing on particular extreme requirements

So what do we want?

CS514

- Systems with simple protocols
- Systems that have "only" probabilistic guarantees
- Systems that scale to very large numbers of nodes
  - No "weakest link" phenomenon
- Systems that are relatively insensitive to "churn"
  - Nodes coming and going
- Systems that disseminate data pretty fast

"Push" versus "pull"

CS514

- Pablo showed us Content Distribution Networks (CDN)
- As used by Akamai, these are "pull" based systems
  - User requests drive the distribution of data into caches
- The pub/sub systems we looked at are "push" based systems
  - Publish events drive the distribution of data

We're interested in both

CS514

- But for now we are going to focus on push based systems
  - First gossip, then reliable multicast (of various forms)
- Later we'll look (a little more) at pull (caching) based systems

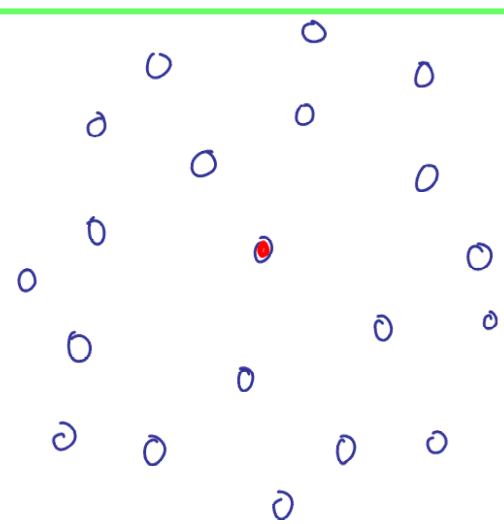
Basic goal:

CS514

- Distribute some data among a group of nodes
  - Should be fast, but no synchrony guarantees
  - Should be robust (some nodes may crash, but still works)
  - Should scale to many nodes
  - Should be efficient

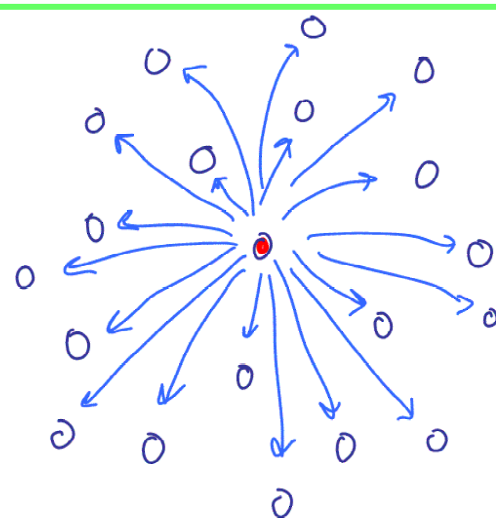
Basic goal: fast, robust data dissemination

CS514



Basic goal: fast, robust data dissemination

CS514



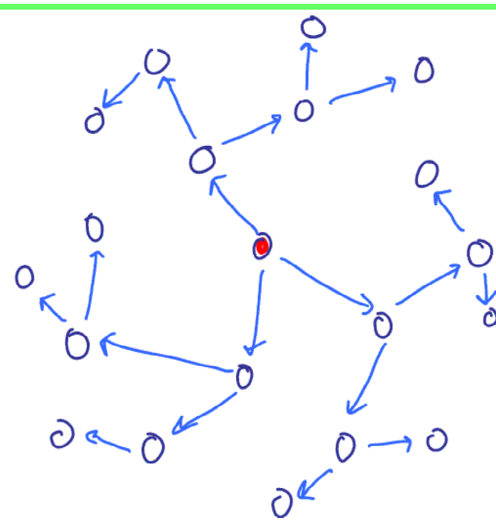
One way…sender sends message to all other nodes one at a time

CS514

- Efficient, in that exactly N-1 copies are sent
- But slow !
  - (N-1)*L, where N is the number of nodes, and L is the time it takes to send the message
- So to overcome this, we want to exploit some kind of parallelism
- (Also, how does the sender learn about all the other nodes?)

Another way…build a tree

CS514



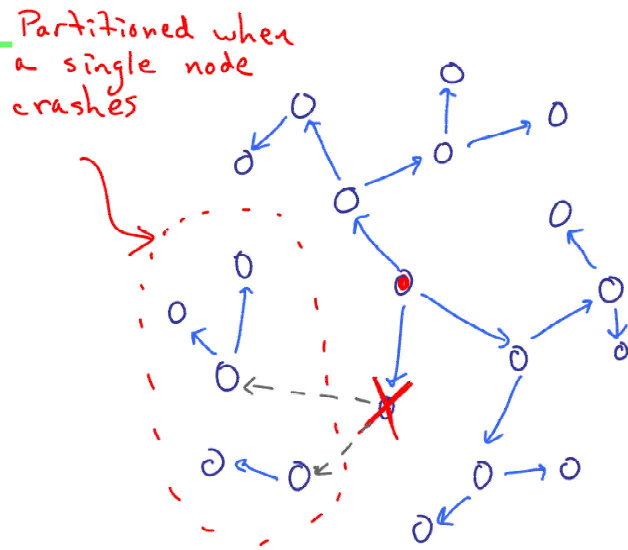
Another way…build a tree

CS514

- Very fast (lots of parallelism)
- Very efficient
  - N-1 copies sent
  - And spread over many nodes (not just one sender)
- But fragile, and complex
  - Hard to build these trees
  - If one node crashes, other nodes are partitioned, at least for a while

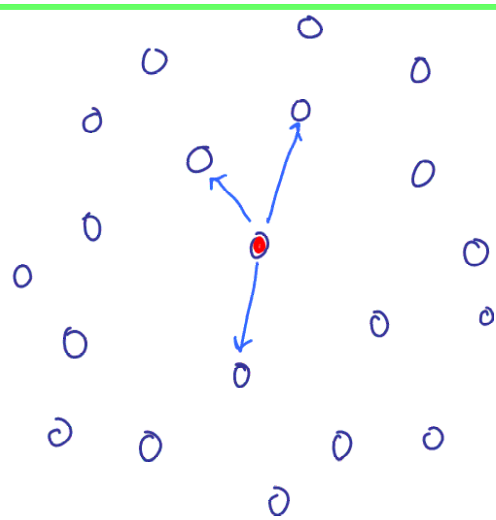
Tree partition

CS514



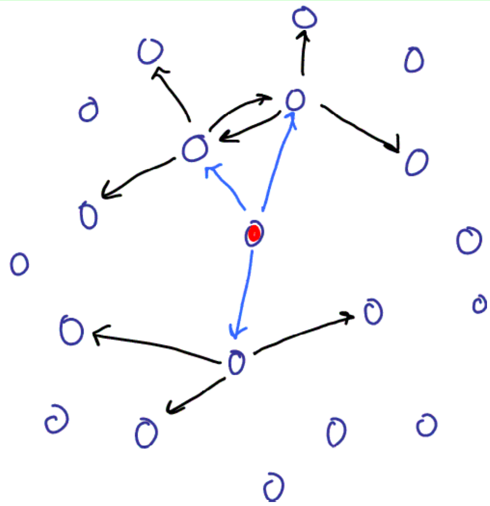
Another way…flood the data

CS514



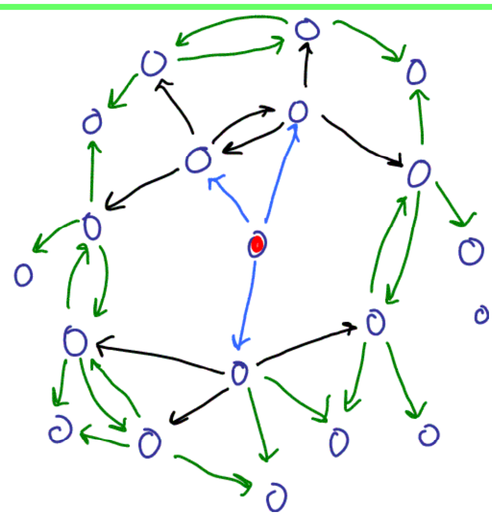
Another way…flood the data

CS514



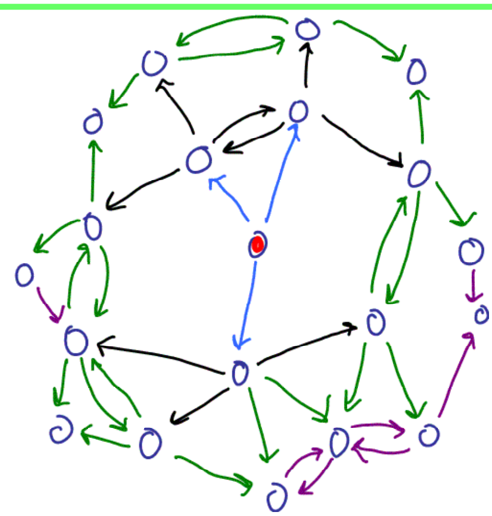
Another way…flood the data

CS514



Another way…flood the data

CS514



Another way…flood the data

CS514

- Robust, fast, and scales well
- But quite inefficient
  - Most nodes receive the message multiple times…worse with higher node degree
  - Also, each node must remember identifier of specific received messages (so that the flood can terminate)

Another way…gossip (a.k.a. epidemic algorithm)

CS514

- Gossip is something like flooding
  - Robust, not perfect efficiency
- Flooding paradigm is message forwarding
  - Gossip paradigm is state exchange
- Flooding nodes forward messages immediately
  - Gossip nodes exchange state periodically
- Flooding nodes keep list of recent message identifiers
  - Gossip nodes keep current state

Another way…gossip (a.k.a. epidemic algorithm)

CS514

- Flooding nodes talk to small number of “neighbors”
  - Gossip nodes talk at random with any other node
- Flooding is a fast burst of activity
  - Gossip is a slow persistent burn
- Ultimately gossip is more robust because it continuously tries to synchronize state
  - With flooding, if a node fails to receive a message, it doesn’t get a second chance

History of Gossip

CS514

- Grapevine/Clearinghouse Directory Service (Demers, Xerox PARC, 1987)
- Refdbms (Golding, UCSC, 1993)
- Bayou (Xerox PARC, 1995)
- Bimodal Multicast (Cornell, 1998)
- Astrolabe (Cornell, 1999)

State Monotonic Property

CS514

- A gossip message contains the state of the sender of the gossip.
- The receiver uses a Merge function to merge the received state and the sent state:
  - State' = Merge(State, Gossip)
- Need some kind of monotonicity:
  - State' $\geq$ State, State' $\geq$ Gossip
  - Without this, old "news" will constantly chase new "news"
  - Can be implemented with a per datum sequence number set by the state originator

Anti-Entropy

CS514

- This gossip scheme with monotonic merge is sometimes called anti-entropy.
- The protocol is called a simple epidemic.

How fast (and how well) does gossip spread?

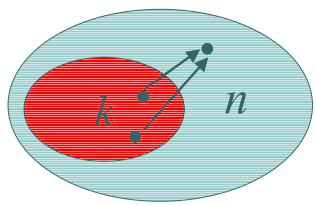
CS514

- Epidemic theory (e.g., Bailey …)
- Assume a fixed population of size *n*.
- For now, assume homogeneous spreading
  - simple epidemic: anybody can infect anyone else with equal probability
- Assume *k* members already infected.
- Assume infection occurs in *rounds*.

Probability of Infection?

CS514

- What is the probability Pinfect(k, n) that a particular uninfected member is infected in a round if k are already infected?



Pinfect(k, n)
$= 1 - P(\text{nobody infects member})$
$= 1 - (1 - 1/n)^k$

E(#newly infected members) =
$(n - k) \times \text{Pinfect}(k, n)$
(binomial distribution)

Phase 1: fast growth of infection

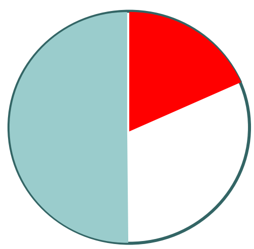
CS514

- Early on, most state exchanges result in a new infection
  - Initial rate of infection: factor of 2
- In the middle, start reaching saturation
  - Half way: factor of 1.4
- In the end, most data exchanges are redundant, but the remaining uninfected nodes are infected rapidly
  - Near end, factor ≈ 1

Intuition: 2 phases

CS514

- Phase 1: $1 \rightarrow n/2$ (first half)
- Phase 2: $n/2 \rightarrow n$ (second half)
- For large n, Pinfect($n/2$, n) $\approx 1 - (1/e)$^.5 $\approx$ .4
- Half way:



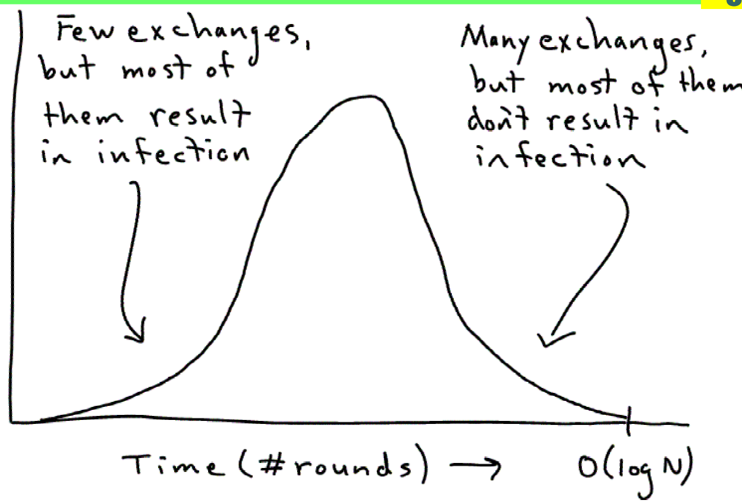
- Infection grows by factor 1.4
- Uninfection declines by factor .4

Exponential growth

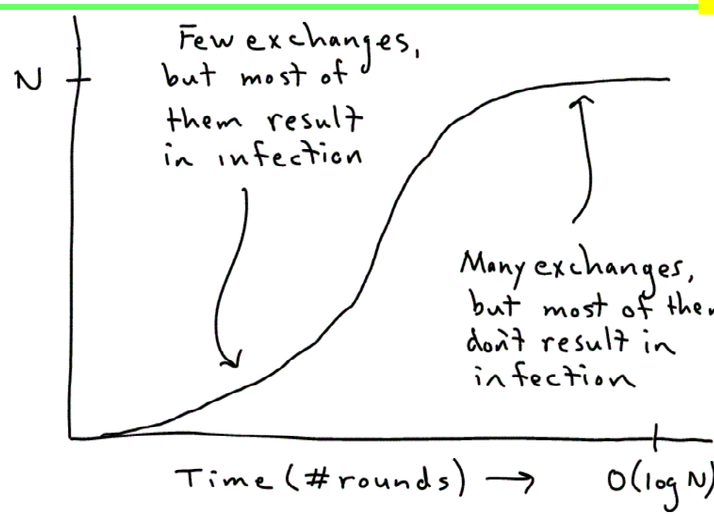
CS514

- Taken together: #rounds necessary to infect the entire population grows O(log n)
- Base of log: 1.585 (experimental)
- Even under bad conditions (see later):
  - member failures
  - message loss
  - but base of log decreases

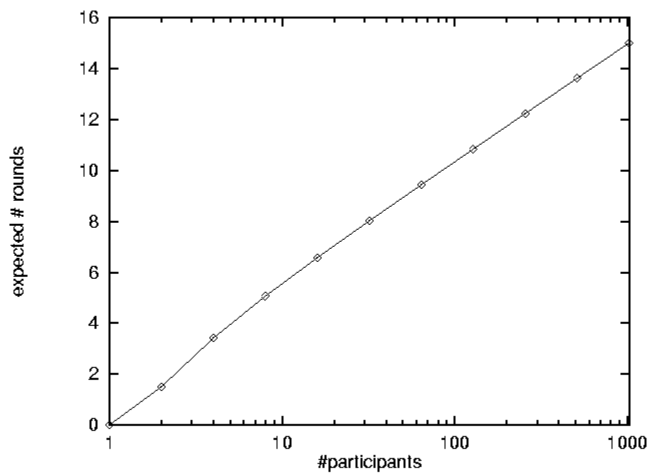
Number of new infections



Number of infected nodes



Expected #rounds



Push versus pull

- If data entries are big, it is costly to "push" complete state in each round
- Instead, send a "digest" of the state, and the recipient can request anything it doesn't already have
  - I.e. the timestamp of each data entry
- This is an optimization that doesn't change the basic concept

Case Study 1: Failure Detection

CS514

- Robust and accurate FD over a wide area is difficult
- All nodes pinging all other nodes doesn't scale
- One or a few nodes pinging all other nodes isn't robust
  - And doesn't scale for those few nodes
- What can gossip do for us here?

Informal Properties

CS514

- Mistake probability fixed
  - (independent of #members)
- Scales in #members (O(nlogn))
- Scales in bandwidth (O(n))
- Resilient against message loss
- Resilient against crashes

Environment

CS514

- Crash failures and partitions
- Unbounded message delay
- Negligible clock drift

Basic Gossip Protocol

CS514

- Each member maintains a list of (address, heartbeat) pairs
- Periodically, each member gossips:
  - increments its own heartbeat
  - sends list to randomly chosen member
- On receipt of gossip, merge lists
- Each member maintains last time heartbeat increased for each other

Linear Bandwidth

CS514

- Gossip message grows linearly with *n*
- #members grows linearly with *n*
  - Slow down gossiping linearly:

$$T_{gossip} = 8n / B$$

How long to wait before reporting failure?

Model

CS514

- Each *micro*-round one random member gossips to another random one.
- We track "infection" of one heartbeat of one member.
- Calculate probability that *k* members are infected in micro-round *i*: $P(k_i = k)$
- *f* members failed from start

Failure Caveat

CS514

- Assume initial member does not fail
  - To simplify analysis
- This affects outcome by at most one round:
  - Initially infected member would have to crash right after it gossips
  - So does the recipient of the gossip, and so on.

Analysis

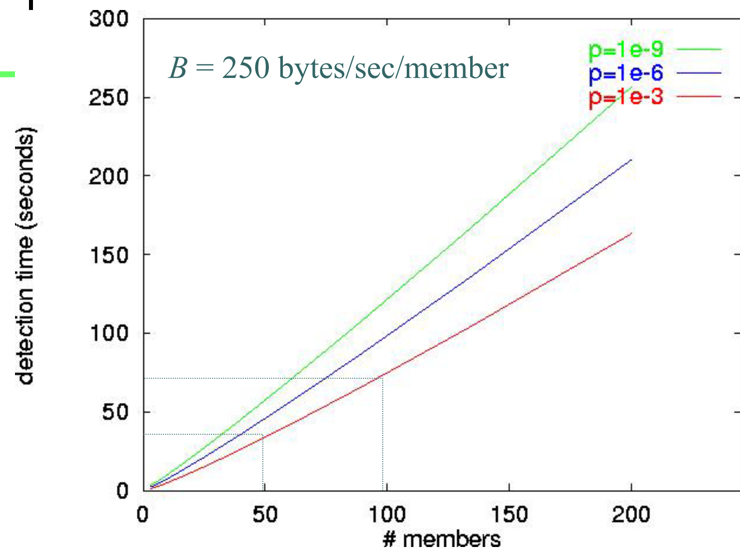
CS514

$$P_{inc}(k) = \frac{k}{n} \times \frac{n-f-k}{n-1} \times P_{arrival}$$

$$P(k_{i+1} = k) = P_{inc}(k-1) \cdot P(k_i = k-1) + (1 - P_{inc}(k)) \cdot P(k_i = k)$$

$$P_{mistake}(r) \le (n-f)(1 - P(k_r = n-f))$$

Failure Detection Time

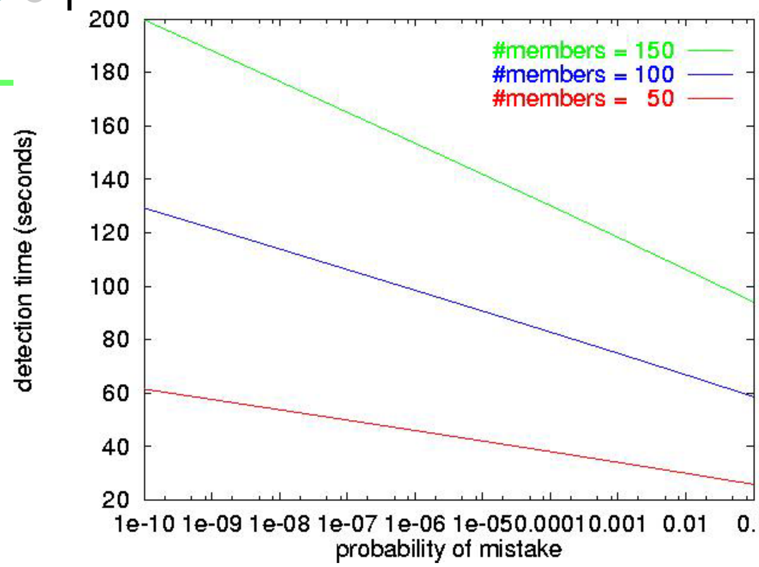


Seems slow…

CS514

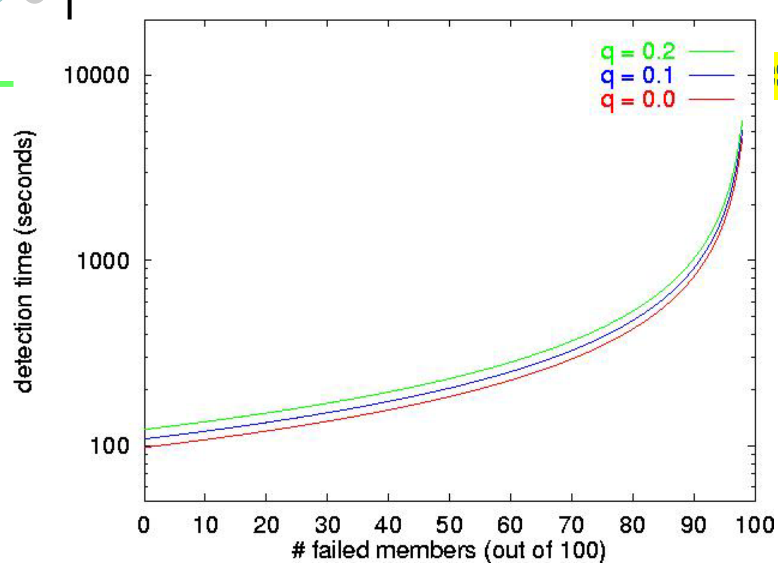
- Takes ~35sec to detect a down member with .999% correctness
  - 250 bytes/second/member
  - 50 members at 8 bytes per member
  - = 400 bytes per state transfer
  - Which means *1.6 sec per round*

Quality of Detection



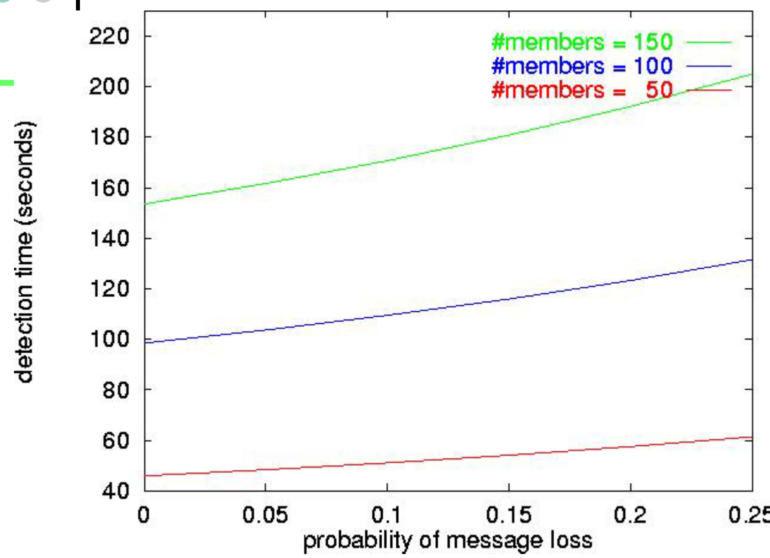
CS514

Effect of Failed Members



CS514

Effect of Message Loss



CS514

What to make of this

CS514

- The approach is very robust
  - Consider message loss, node failure
- But also slow
  - Because the whole state is exchanged each round, and bandwidth kept rather low
- Turns out an alternative approach is faster, and nearly as robust . . .

Faster approach to failure detection

CS514

- Use gossip to advertise complete set of members
  - This can be somewhat slow
  - We are interested in quickly detecting failure, not newly joined members
- Have each member ping 4-5 others
  - Use an arbitrary convention to decide which…
  - Such as, ping four members with two immediately higher and two immediately lower member IDs

Faster approach to failure detection

CS514

- Direct ping can detect crash with high probability in 10 – 30 seconds
  - Depending on quality of communications path
- When detect failure, gossip failure with very short period (100ms)
- Require multiple members to detect failure (i.e. 2 out of 4)

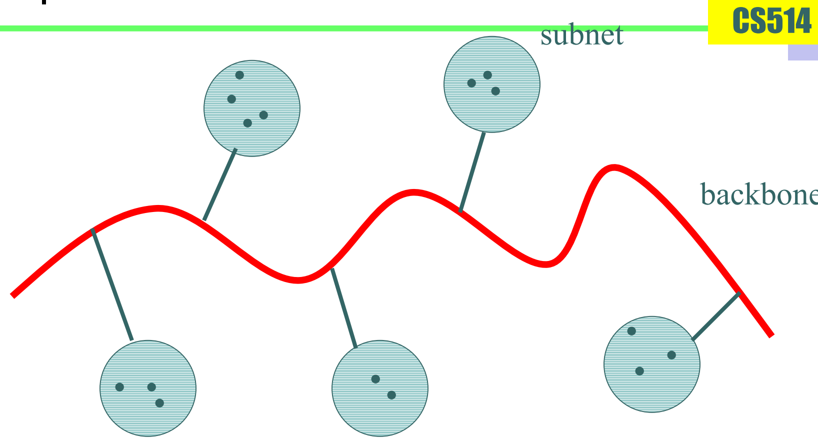
Simple gossip has some scaling issues

CS514

- Requires full membership
  - doesn't scale
- Load on network grows quickly
  - linear if one source of information
    - One source x N members
  - quadratic if all participants can contribute
    - N sources x N members
- Led to demise of Xerox Clearinghouse
  - (and the victory of DNS)

High load on routers

CS514



Idea: add locality to gossip

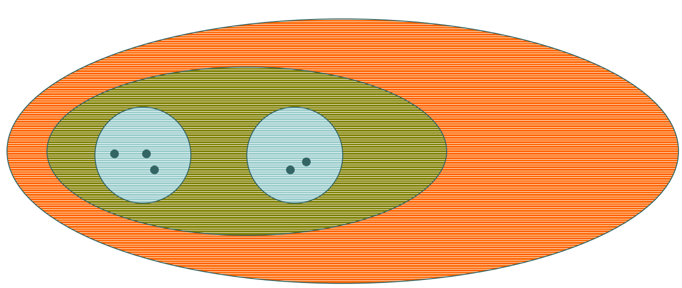
CS514

- Gossip mostly in your neighborhood
- Occasionally gossip farther away
- Generalize to multiple levels
- Resembles spread of (real) viruses

Domains

CS514

- Smallest domain: local host
- Largest domain: all hosts
- Domains are fully nested (form a tree)



Multi-level Gossip Protocol

CS514

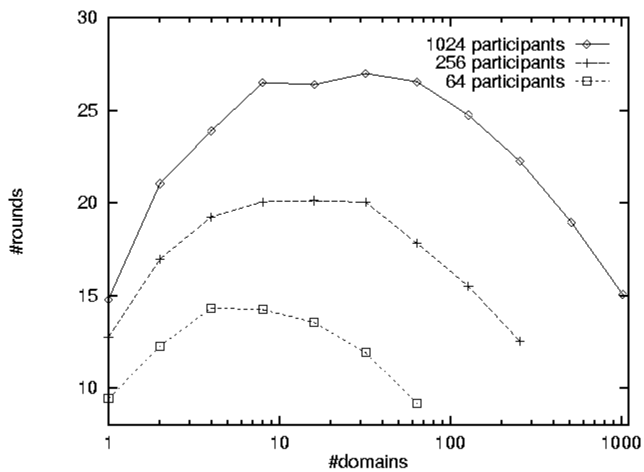
- Start with local domain
- Pick a member at random
- If picked self, go to next level up
  - If no more levels, don't gossip
- Send gossip to chosen member
  - pick random subdomain in chosen member
  - if not host-level, then descend into subdomain
  - otherwise send message

Better properties

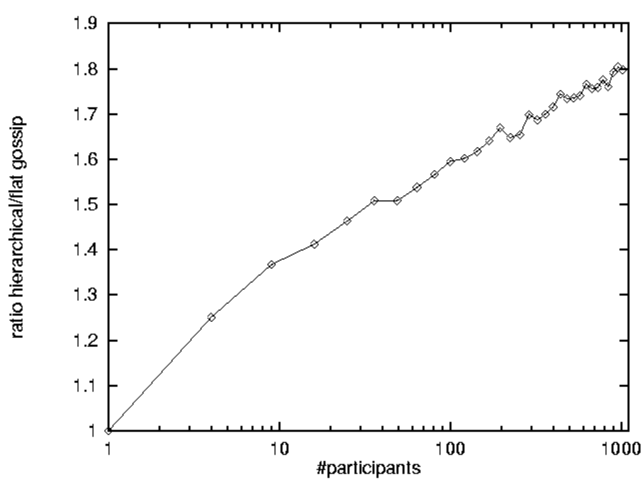
CS514

- Most gossips are local
- Fewer problems with partitioning
- At every level, about the same gossip load
- Within any domain, there is, on average, one gossip message from every node to every other node
- But, propagation is slower:

Two-level hierarchy

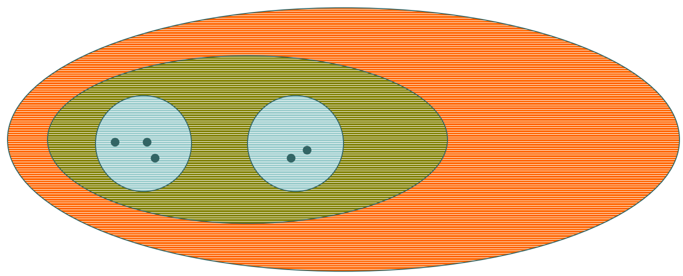


Two-level cost



No longer logarithmic...

CS514



- #levels in the domain tree is O(log *n*)
- resulting growth, log^log, is polynomial

Problems

CS514

- Polynomial growth
  - (degree is small though, like .2)
  - if n = 1,000,000,000, branching factor is 100, and gossip every second,
  - dissemination time < 10 min.
- Still requires full membership
- Message sizes may grow linearly if everybody contributes information (e.g., a sequence number for each member)

New idea (Astrolabe)

CS514

- Reduce information content with distance
  - e.g., go from exact values to average values
  - from exact membership to representatives
  - use distance metric in the domain tree

Related Literature

CS514

- The Mathematical Theory of Infectious Diseases and its Applications. N.T.J. Bailey. Hafner Press. 1975.
- Epidemic Algorithms for Replicated Database Maintenance. A. Demers et al. Proc. of the 6th ACM PODC conf. August 1987.
- A Weak-Consistency Architecture for Distributed Information Services. R.A. Golding. Computing Systems 5(4), Fall 1992.
- Flexible Update Propagation for Weakly Consistent Replication. K. Petersen et al. Proc of the 16th ACM SOSP conf. October 1997.

- My home page: http://www.cs.cornell.edu/home/rvr