Gossip Algorithms

Seif Haridi
Ali Ghodsi
Peter Van Roy

Gossip is a general technique

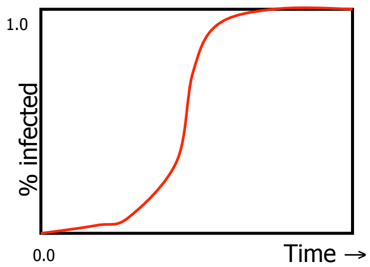
- Gossip algorithms: also called epidemic algorithms
- Important technique to solve problems in dynamic large scale systems
  - Scalable
  - Simple
  - Robust to node failures, message loss and transient network disruptions (network partitions, …)

Introduction to Gossip

- Suppose that I know something
- I'm sitting next to Ali, and I tell him
  - Now 2 of us "know"
- Later, he tells Cosmin and I tell Tallat
  - Now 4
- This is an example of a *push* epidemic
- Pull happens if Ali asks me instead
- *Push-pull* occurs if we exchange data

Gossip scales very nicely

- Participants' loads independent of size
- Network load linear in system size
- Information spreads in log(system size) time



What is a gossip protocol (1/2)?

- Cyclic/Periodic, pair-wise interaction between peers
- The amount of information exchanged is of (small) bounded size per cycle
- The state of each peer is bounded (small)
- During interaction the state of one of both peers changes in a way that reflects the state of the other peer

4/26/11 5

What is a gossip protocol (2/2)?

- Random peer selection
  - The full peer set, or
  - Small set of neighbors
- Reliable communication is not assumed
- The protocol cost is negligible
  - The frequency of interaction is much lower than message round-trip times

4/26/11 6

Gossip for dissemination

- Information Dissemination Protocols: gossip to spread information in a manner that produces bounded worst-case loads
  - Event dissemination protocols use gossip to perform multicast. They report events periodically.
  - Background data dissemination protocols gossip about information associated with nodes

Gossip for repairing

- *Anti-entropy protocols for repairing replicated data,* which operate by comparing replicas and reconciling differences
  - "I have 6 updates from Cosmin"
- If we aren't in a hurry, gossip to replicate data too
- Typical use (bimodal Multicast)
  - Use a best effort multicast
  - Then gossip to fill the gaps

Gossip for membership

- Start with a *bootstrap protocol*
  - For example, processes go to some web site and it lists a dozen nodes where the system has been stable for a long time
  - Pick one at random
- Then track "processes I've heard from recently" and "processes other people have heard from recently"
- Use push gossip to spread the word

Gossip for aggregates

- *Protocols that compute aggregates*
- These compute a network-wide aggregate by:
  - Sampling information at the nodes in the network
  - Combining the values to arrive at a system-wide value
  - Like wave algorithms computing average, max, min, …
- Example: the number of nodes in the system

Gossip for network topology

- *Protocols that arrange network topology*
- Example: *rings (T-man algorithm)*
  - Starting from local view of fixed size in a random network
  - Do bidirectional exchange of the view with a random peer ⇒ 2 views
  - Keep peers with ids near to you ⇒ 1 view
  - repeat

Gossip for many other tasks…

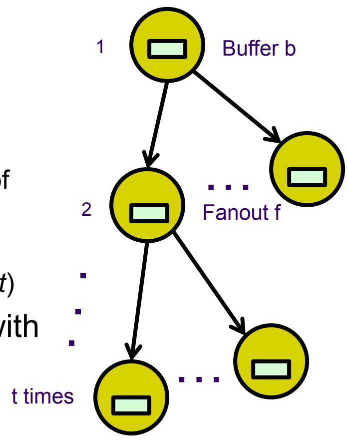
- Gossip has been used for many other things
  - Global failure detection
  - Global clock synchronization
  - Reputation dissemination
  - …

Information dissemination

4/26/11 13

Information dissemination

- Start with one peer that wants to disseminate some message
- Every peer does the following:
  - Buffers every message (information unit) it receives up to a certain *buffer capacity* ***b***
  - Forwards that message a limited number of *hops* or *time steps* ***t***
  - Forwards the message each time to ***f*** randomly selected set of processes (*fanout*)
- Many variations of this scheme exist with different values of ***b***, ***t***, ***f***
  - We will look at a few interesting ones



4/26/11 14

Information dissemination?

- Dissemination is like a disease epidemic
- Given a system size ***n*** (population size)
  - What is reliability of information delivery given ***b***, ***t***, ***f***, ***n***?
  - Does it depend on ***n***?
  - How many cycles do we need to infect all peers?
- Let us answer a few of these questions
  - N.T.J. Bailey, "The Mathematical Theory of Infectious Diseases and Its Applications", 2nd ed., Hafner Press, 1975.

Infect-forever model

- Fixed size population ***n***
- One infectious individual at round 1
- Infected individuals remain infectious throughout
- Y_r is the fraction of individuals infected at round ***r***
- Assume that infectious individuals try to contaminate ***f*** other members in each round:

$$Y_r \approx \frac{1}{1 + n \cdot e^{-f \cdot r}}$$

- The ratio of number of infected individuals to number of uninfected ones increases exponentially fast on average, by a factor of e^f in each round.

Latency in infect-forever model

- The number ***R*** of rounds necessary to infect the entire system respects the following equation:

$$R = \log_{f+1}(n) + \frac{1}{f}\log(n) + O(1)$$

- For f = 2:
  - Round 1: 1
  - Round 2: 3
  - Round 3: 3+6=9

Infect-and-die model

- Each process will take action to communicate a message exactly once, namely after receiving that message for the first time
  - Infectious process "remains infectious" for just one round
- No further action is taken, even if copies of the same message are received again
- The proportion π of processes eventually contaminated satisfies the following fixpoint equation:

$$\pi = 1 - e^{-\pi \cdot f}$$

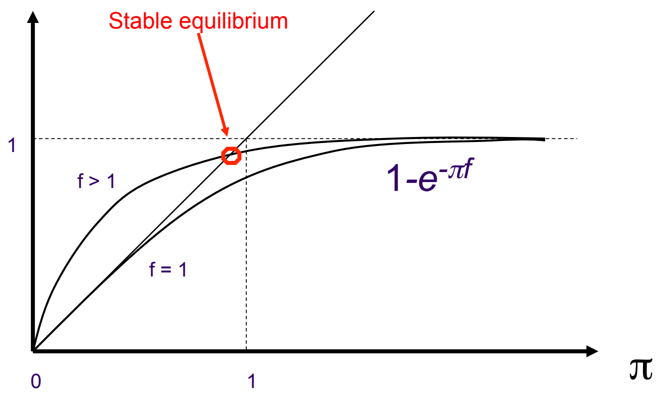
Infect-and-die model

- If $f = 1$, then $\pi = 0$ satisfies the equation
- When $f > 1$, π becomes positive
  - f=1 is a critical value
  - Example f = 2, π = 0.5
- π approaches 1 for very large populations

π is the proportion of processes eventually contaminated

$$\pi = 1 - e^{-\pi \cdot f}$$

Infect-and-die model



Latency in infect-and-die

- For population size ***n***, ***f*** must be O(log(***n***)) for the infection to reach the whole system
- If this condition is satisfied, then the number ***R*** of rounds necessary to infect the entire system respects the following equation:

$$R = \frac{\log(n)}{\log(\log(n))} + O(1)$$

4/26/11 21

Issues in info dissemination

- *Membership*
  - How peers get to know each other, and how many do they need to know
    - Assumption that all peers know each other does not hold in large systems
    - Trade-off: scalability against reliability (provide a partial view of the peers)
    - Piggyback membership information on messages
- *Network awareness*
  - How to make the connections between peers reflect the actual network topology (typically, use a hierarchy)
- *Buffer management*
  - Which information to drop at a process when its storage buffer is full
  - Prioritize messages, drop low-priority ones (e.g., older ones, or ones that are subsumed by others)

4/26/11 22

Membership

- Each process has a partial view
- The view should have a random sample of node, even under churn
- Whenever a process forwards a message, it also includes in this message a set of processes it knows
- Hence, the process that receives the message can enhance the list of processes it knows by adding new processes

Membership protocols Requirements

- Uniformity
  - All nodes play the same role, no biases
- Adaptivity under churn
  - The parameters have to be tuned (t and f)
- Bootstrapping
  - How do nodes enter and leave, how to start
- Several protocols designed for this
  - Cyclon, Newscast, SCAMP

Buffer management

- Depending on the broadcast rate, the buffer capacity of every process may be insufficient to ensure that every message is buffered long enough
- Messages are classified according to their *age* (the number of processes the message went though)
- Replace old messages
- Replace subsumed messages (depends on application semantics)

4/26/11 25

Small-world networks

4/26/11 26

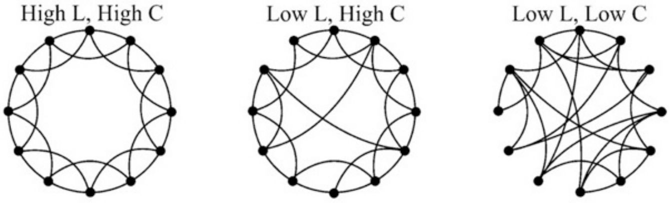
Small-world networks (1)

- There is a spectrum of how processes choose infection targets
  - Nearest-neighbor network: processes choose targets only from neighbors (number of rounds is $O(\boldsymbol{n}^{1/D})$ for D-dimensional grid)
  - Random network: processes choose targets randomly from all the nodes (what we saw so far) (number of rounds is $O(\log(\boldsymbol{n}))$)
  - There is a third, in-between case that is very interesting: small-world networks (number of rounds is also $O(\log(\boldsymbol{n}))$)
- Many real-world social networks are small-world networks

Small-world networks (2)

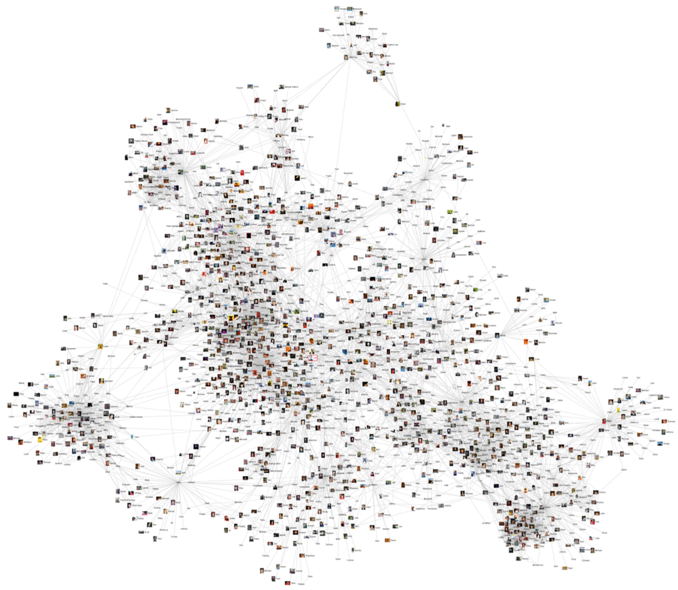
- A small-world network has both nearest neighbor connections as well as long-range connections
  - Most nodes can be reached with a small number of hops
  - Social networks (Facebook, movie casts, …), Internet connectivity, Web, gene networks are all small-world networks
- A small-world network is defined by two properties: (1) a small average shortest path length with (2) a high clustering coefficient
  - Random graphs have a **low** CC → SWN cluster more
  - Neighbor graphs have a **large** path length → SWN have shorter paths
- *Clustering coefficient c* measures degree of clustering ($0 \leq c \leq 1$)
  - For each node, count the number of edges between neighboring nodes and divide by the maximum possible (n neighbors give n(n-1)/2 max)
  - Clustering coefficient = average of this number for all nodes

Small-world networks (3)



- Nearest-neighbor graphs: High L, High C
- Small-world graphs: Low L, High C
- Random graphs: Low L, Low C

Example of a real SWN



Gossip Framework of Jelasity and Babaoglu

Proactive gossip framework

```
// active thread
do forever
    wait(T time units)
    q = SelectPeer()
    push S to q
    pull Sq from q
    S = Update(S,Sq)

// passive thread
do forever
    (p,Sp) = pull * from *
    push S to p
    S = Update(S,Sp)
```

Proactive gossip framework

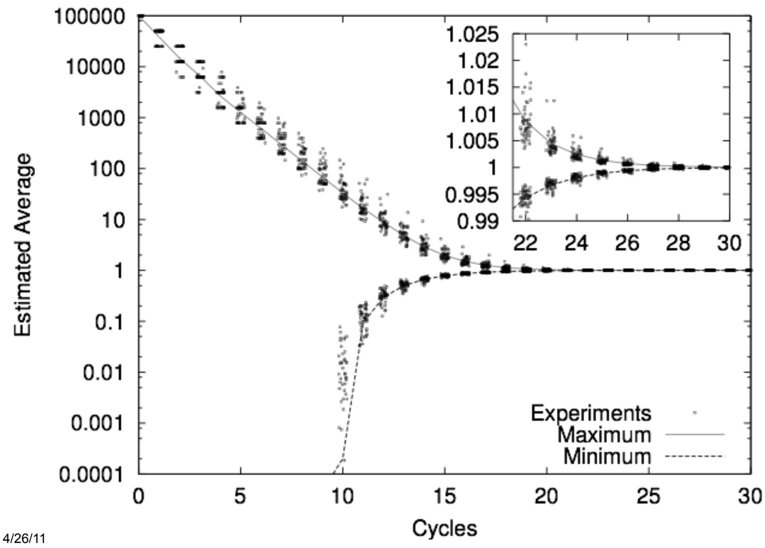
- To instantiate the framework, define:
  - Local state ***S***
  - Method **SelectPeer()**
  - Style of interaction
    - **push-pull**
    - **push**
    - **pull**
  - Method **Update()**

1: Aggregation

Gossip framework instantiation

- Style of interaction: push-pull
- Local state ***S***: Current estimate of global aggregate
- Method **SelectPeer()**: Single random neighbor
- Method **Update()**: Numerical function defined according to desired global aggregate (arithmetic/geometric mean, min, max, etc.)

Exponential convergence



Properties of gossip-based aggregation

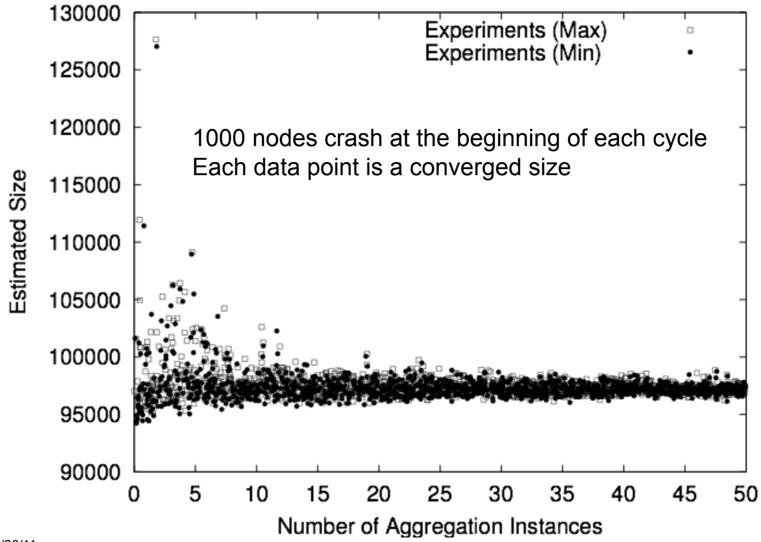
- In gossip-based averaging, if the selected peer is a globally random sample, then the variance of the set of estimates decreases exponentially
- Convergence factor:

$$\rho = \frac{E(\sigma_{i+1}^2)}{E(\sigma_i^2)} \approx \frac{1}{2\sqrt{e}} \approx 0.303$$

Network-size estimation

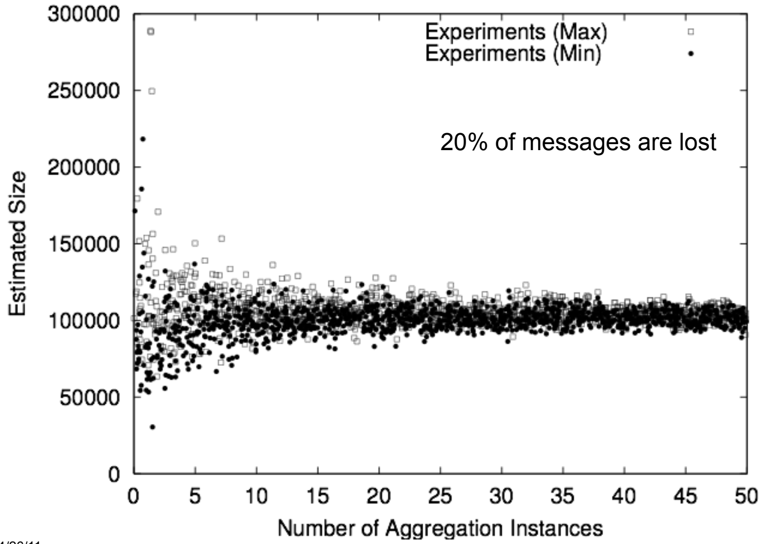
- Basic idea: if exactly one node has value 1 and all others have value 0, then the global average is 1/N, from which we deduce the size N
  - How can we guarantee that exactly one node has value 1? [d]
- We don't actually need leader election
  - We allow multiple nodes to randomly start concurrent instances of the basic protocol
  - Each concurrent instance is tagged with a unique identifier
  - We use a simple scheme to bound the number of concurrent instances
  - We periodically restart the algorithm (new epoch) and use the median values from the previous epoch to start the new epoch

Network size estimation



4/26/11 39

Network size estimation



4/26/11 40

2: Topology management

Topology management

- Topology management can be provided as an abstract service
  - Gossip-based scheme called T-Man
  - General scheme based on a ranking function
    - Paper shows results for ring, torus, binary tree
  - Topology gradually appears as a result of a ranking function
- Gossip is a very useful framework for large distributed systems
  - Failure detection, resource monitoring, data aggregation, database replication
  - T-Man can jumpstart other protocols (like DHTs)

Problem definition

- Given a set of *N* nodes
  - Each node has a view of size *c* containing node descriptors (address,profile)
  - Profile is used to calculate ranking
- Given a ranking function *R*
  - $R(x,\{y_1,\ldots, y_m\})$ = {all orderings of $\{y_1,\ldots, y_m\}$}
  - *R* can be defined through a distance function $d(x,y)$
- Problem: For all nodes *x*, construct view_x such that R(*x*,{all nodes except *x*}) contains a ranking that starts with view_x

T-Man algorithm (1)

```
view ← initialView()
do at a random time once in each
consecutive interval of T time units
  p ← selectPeer()
  myDescriptor ← (myAddress,myProfile)
  buffer ← merge(view,{myDescriptor})
  send buffer to p
  receive view_p from p
  buffer ← merge(view_p,view)
  view ← selectView(buffer)
```

(a) active thread

```
do forever
  (q, view_q) ← waitMessage()
  myDescriptor ← (myAddress,myprofile)
  buffer ← merge(view,{myDescriptor})
  send buffer to q
  buffer ← merge(view_q,view)
  view ← selectView(buffer)
```

(b) passive thread

- Each node has two threads: an active thread initiating communication with other nodes and a passive thread waiting for incoming messages
- Still need to define initialView(), selectPeer(*v*), merge(v_1,v_2), selectView(b)

T-Man algorithm (2)

- InitialView() = random sample of nodes
- Merge(v_1,v_2) = $v_1 \cup v_2$
- SelectPeer(v) = (rank current view v according to R and return random sample from first half)
- SelectView(b) = (rank b according to R and return first c elements)

Some ranking functions

- Line and ring (profiles are real numbers)
  - Line: $d(a,b) = |a-b|$
  - Ring: $d(a,b) = \min(N-|a-b|, |a-b|)$
- Mesh and torus (profiles are 2D real vectors)
  - Manhattan distance, with boundary conditions
- Binary tree
  - Profiles are binary strings giving path on the tree
  - Ranking is according to number of steps (routing on the binary tree)

Example: Torus

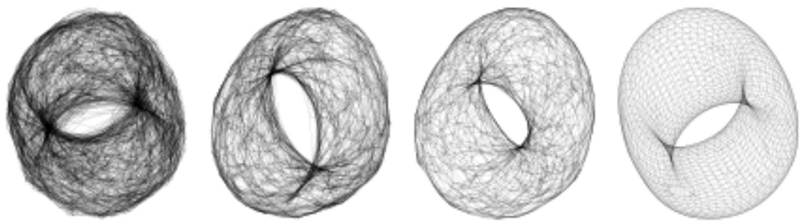


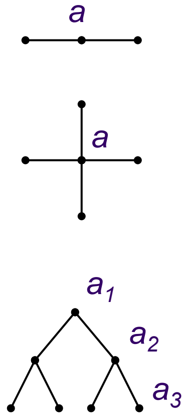
Figure 2. Illustrative example of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random topology with $c = 20$. For clarity, only the nearest 4 neighbors (out of 20) of each node are displayed.

- Define one cycle as the time interval during which *N* view updates happen
  - On average, one cycle is *T/2* time units

Analysis (1)

- Two phases: Rapid convergence phase followed by endgame phase
- We use the following approximate model
  - After *1* cycle, view is closest *c* out of *2c*
  - After *i* cycles, view is closest *c* out of $2^i c$
- Two sources of error
  - Unbalanced contact rate: Nodes may converge faster or slower than the average (both are bad)
  - Distribution skew: When nodes exchange views, they are from different distributions

Analysis (2)



- Define maximal rank $r_{a,b}$ of node b with respect to base node a: largest rank b that can possibly be assigned
  - Ring: 2, torus: 4
  - Binary tree: 2 (root), 3 (internal node), 1 (leaf)
- Define $p_{a,b}(i)$ probability that b is present in the view of a after cycle i
  - $p_{a,b}(0) = c/(N-1)$
  - If $r_{a,b}<c$ then $p_{a,b}(l) = 2p_{a,b}(l-1) = 2^i p_{a,b}(0)$
- Since $p<1$ we have $i<\log_2(N-1)-\log_2(c)$

Simulation results

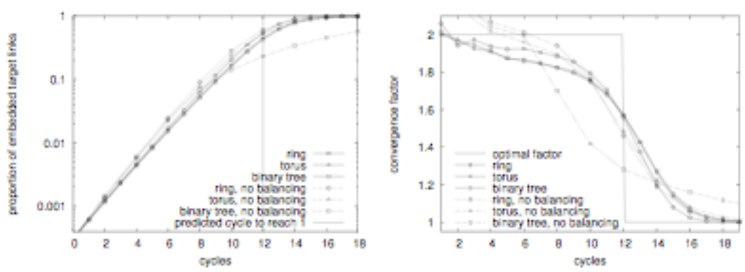


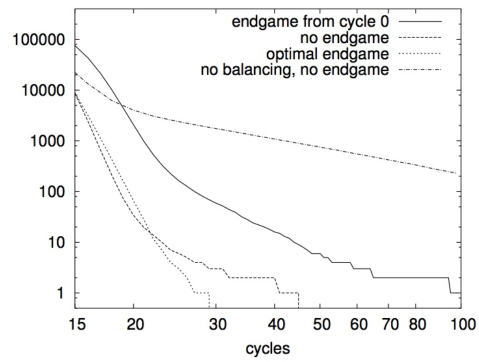
Figure 3. Comparison of convergence speed for network sizes $N = 2^{17}$ and $c = 40$, with and without balancing of the number of contacts per cycle per node. Averages from 20 runs are shown.

- Convergence factor = increase in embedded links per cycle
  - It approximates the predicted factor 2 until the cycle where convergence is predicted
- Binary tree is bad without "balancing" optimization (see later)

Handling the endgame

- Near the end, a few nodes are left behind
  - The final convergence time heavily depends on the actual topology (ring is worst!)
- Two important improvements
  - Balancing: during rapid convergence, receiving node refuses contact if there are already too many
    - Sending node searches for a willing receiving node (can be done with 1-bit ping messages)
  - Routing: during endgame, instead of random selection of a peer node, select closest one on the topology (with exponentially decreasing probabilities) (i.e., do routing!)
    - Change SelectPeer() at predicted convergence (start of endgame)

Ring endgame convergence

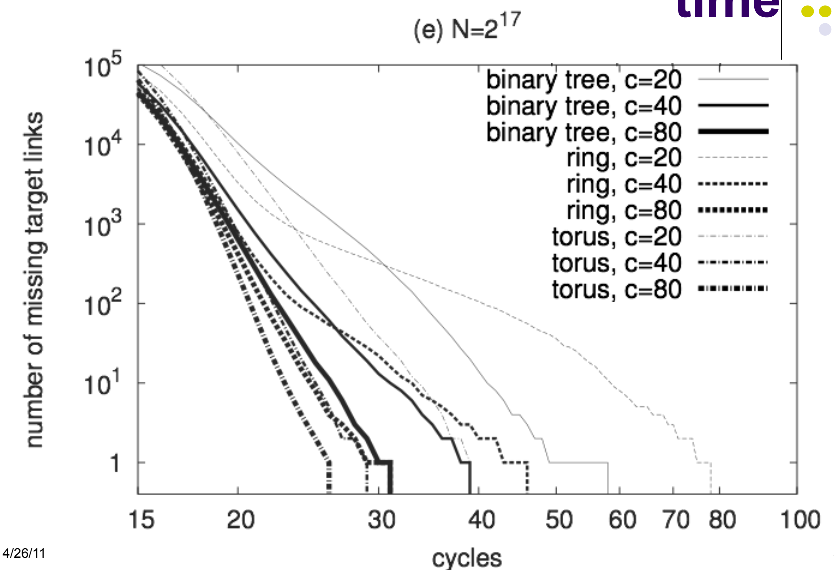


- Comparison of convergence speed for the ring topology in the end phase for network sizes N = 2^{17} and c = 40 for different versions of T-Man (shows averages of 20 runs)
- Best is with balancing and routing (optimal endgame)
- Worst is with no balancing and no routing
- Both balancing and routing are essential

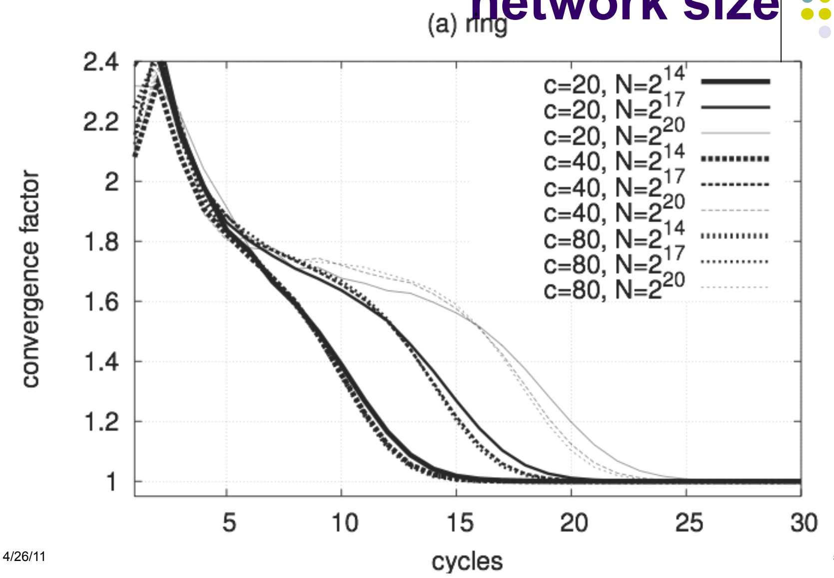
Conclusions

- T-Man is a general topology management service that converges quickly (in logarithmic time), scales well (simulated to 2^{20} nodes), and is robust
- T-Man is based on a gossip framework
  - It runs in two phases: a rapid convergence phase and an endgame phase
  - We give two optimizations that allow good performance in the endgame

Exponential convergence - time



Exponential convergence - network size



3: Heartbeat synchronization

Synchrony in nature

- Nature displays astonishing cases of synchrony among independent actors
  - Heart pacemaker cells
  - Chirping crickets
  - Menstrual cycles
  - Flashing of fireflies
- Actors may belong to the same organism or they may be parts of different organisms

Coupled oscillators

- The “Coupled oscillator” model can be used to explain the phenomenon of “self-synchronization”
- Each actor is an independent “oscillator”, like a pendulum
- Oscillators coupled through their environment
  - Mechanical vibrations
  - Air pressure
  - Visual clues
  - Olfactory signals
- They influence each other, causing minor local adjustments that result in global synchrony

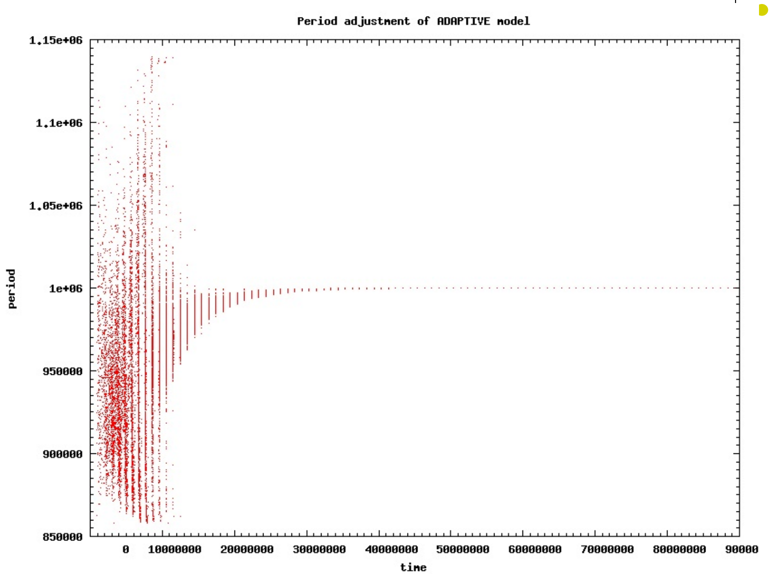
Fireflies

- Certain species of (male) fireflies (e.g., *luciola pupilla*) are known to synchronize their flashes despite:
  - Small connectivity (each firefly has a small number of "neighbors")
  - Communication not instantaneous
  - Independent local "clocks" with random initial periods

Gossip framework instantiation

- Style of interaction: push
- Local state ***S***: Current phase of local oscillator
- Method **SelectPeer()**: (small) set of random neighbors
- Method **Update()**: Function to reset the local oscillator based on the phase of arriving flash

Exponential convergence



4/26/11 61