Gossip Protocols

Márk Jelasity

Hungarian Academy of Sciences and
University of Szeged, Hungary

Introduction

- A few key ideas from previous class
  - Overlay networks are the key abstraction
  - Function is always implemented as a distributed algorithm that communicates over this network
  - Overlay networks can emerge or can be created
- Main points in this class
  - Gossip protocols over networks
    - **Dissemination, computation, overlay construction**
  - Applications for distributed data mining
    - **EM algorithm, clustering, collaborative filtering**

Introduction

- Gossip-like phenomena are commonplace
  - human gossip
  - epidemics (virus spreading, etc)
  - computer epidemics (malicious agents: worms, viruses)
  - phenomena such as forest fires, branching processes and diffusion are all similar mathematically
- extremely simple locally, powerful and robust globally
- In computer science, epidemics are relevant
  - for security (against worms and viruses)
  - for designing useful protocols (we look at this here)

Epidemic Database Updates

- Problem
  - Xerox corporate Internet, replicated databases
  - Each database has a set of keys that have values (along with a time stamp)
  - Goal: all databases are the same, in the face of key updates, removals and additions
  - Updates are made locally and have to be replicated at all sites (300 sites)
- Solution in 1986: emailing updates
  - problems with detecting and correcting errors (done by hand!)
  - bottleneck with the originating (updated) site
  - not scalable (slow if very large number of nodes)
  - (message complexity quite good though!)

Gossip to the rescue

- Main components are replaced by gossip
  - update spreading: rumor mongering (no bottleneck)
  - error correction: anti-entropy gossip (reliable)
- anti-entropy
  - uses “simple epidemics” with two states: infective and susceptible (a.k.a. SI model)
  - guarantees perfect dissemination
- rumor mongering
  - uses “complex epidemics” with an additional state: removed (a.k.a. SIR model)
  - certain (quite small) probability of error

SI gossip

```
1:  loop
2:      wait(Δ)
3:      p ← random peer
4:      if push and in state I then
5:          send update to p
6:      end if
7:      if pull then
8:          send update-request to p
9:      end if
10: end loop
```

```
11: procedure ONUPDATE(m)
12:     store m.update      ▷ means switching to state I
13: end procedure
14:
15: procedure ONUPDATEREQUEST(m)
16:     if in state I then
17:         send update to m.sender
18:     end if
19: end procedure
```

What overlay network is assumed here?

Some Properties of the SI model

- the push model
  - N nodes communicate in rounds (cycles)
  - in each cycle, a node that has the update (infected) sends it to a random other node, that becomes infected too
- In anti-entropy
  - nodes send the (hash of) the entire database (not only a single update)
    - **as a side effect, all new updates are spread according to the SI model**
  - receiving nodes update their own database via merging the unseen updates

Mean-field model of push SI

- Let p_i be the proportion of *not* infected nodes in cycle i
- $1-p_0=1/N$
- Pittel (1987) shows that the model below is quite accurate for predicting p_i

$$E(p_{i+1}) = p_i\left(1-\frac{1}{N}\right)^{N(1-p_i)} \approx p_i e^{-(1-p_i)}$$

Speed and cost of push SI

- Let S_N be the first cycle where $p_i=0$
- Pittel (1987) shows that in probability

$$S_N = \log(N) + \ln(N) + O(1)$$

- This is quite fast...
- But the number of overall messages sent is

$$O(N \log N)$$

Pull and push-pull SI

- With pull, we have

$$E(p_{i+1}) = p_i^2$$

- This is *very* fast when p_i is small (end phase)...
- Karp *et al* (2000) show that the number of overall messages sent with push-pull is

$$O(N \log \log N)$$

- But termination is trickier when no updates are available (for anti-entropy does not matter)

SIR for spreading single updates

- For anti-entropy, use a pull or push-pull SI modell
- For the spreading of updates, the termination problem needs to be addressed: rumor mongering with SIR model
- Push approach
  - when a rumor (update) becomes “cold”, stop pushing
- Pull approach
  - same as push, only stop offering update when pulled when it becomes cold

SIR gossip

```
1:  loop
2:      wait(Δ)
3:      p ← random peer
4:      if push and in state I then
5:          send update to p
6:      end if
7:      if pull then
8:          send update-request to p
9:      end if
10: end loop
11:
12: procedure ONFEEDBACK(m)
13:     switch to state R with prob. 1/k
14: end procedure
```

```
15: procedure ONUPDATE(m)
16:     if in state I or R then
17:         send feedback to m.sender
18:     else
19:         store m.update                ▷ now in state I
20:     end if
21: end procedure
22:
23: procedure ONUPDATEREQUEST(m)
24:     if in state I then
25:         send update to m.sender
26:     end if
27: end procedure
```

Rumor mongering with push

$$\frac{ds}{dt} = -si$$

$$\frac{di}{dt} = si - \frac{1}{k}(1-s)i$$

$$\rightarrow s = e^{-(k+1)(1-s)}$$

- Stop spreading info with probability 1/k if unsuccessful infection attempt (become removed)
- s: susceptible, i: infective, r: removed
- Eg if k=1, 20% miss the gossip, if k=2, 6% miss it
  - In general, with push, the prob to miss the update is approx e^{-m} (m is the overall messages)

Some other rumor mongering algorithms

- Removal algorithms
  - Counter: removed after exactly k unsuccessful attempts
  - Random: removed with pr. 1/k after each unsuccessful attempt
- Blind: removal algorithm is run in each cycle irrespective of contacted node
- Feedback: removal algorithm runs if contacted node was not susceptible

Random networks

- Note that gossiping nodes pick another node in each cycle: they do not need to know all the nodes
- The actual communication pattern defines a random graph
  - by looking at these graphs, we can understand the properties of the communication better
  - we can design better gossip protocols if we understand the implications of our design decisions

Comments on theoretical models

- The ER model is often used to reason about gossip protocol design. This is problematic for a number of reasons
  - In the ER model nodes can get stuck without neighbors. This is the main reason for disconnectivity. In push or push-pull this is impossible
  - If we guarantee that all nodes communicate to at least 4 other nodes after receiving the update, we have a radically different model
- Message and node failure pushes the underlying network toward the ER model, but not completely

Aggregation

- Calculate a global function over distributed data
  - eg average, but more complex examples include variance, network size, model fitting, etc
- usual structured/unstructured approaches exist
  - structured: create an overlay (eg a tree) and use that to calculate the function hierarchically
  - unstructured: design a stochastic iteration algorithm that converges to what you want (gossip)

push-pull averaging

```
1: loop
2:     wait(Δ)
3:     p ← random peer
4:     send push(x) to p
5: end loop
```

```
6: procedure ONPUSHUPDATE(m)
7:     send pull(x) to m.sender
8:     x ← (m.x + x)/2
9: end procedure
10:
11: procedure ONPULLUPDATE(m)
12:     x ← (m.x + x)/2
13: end procedure
```

Illustration of averaging

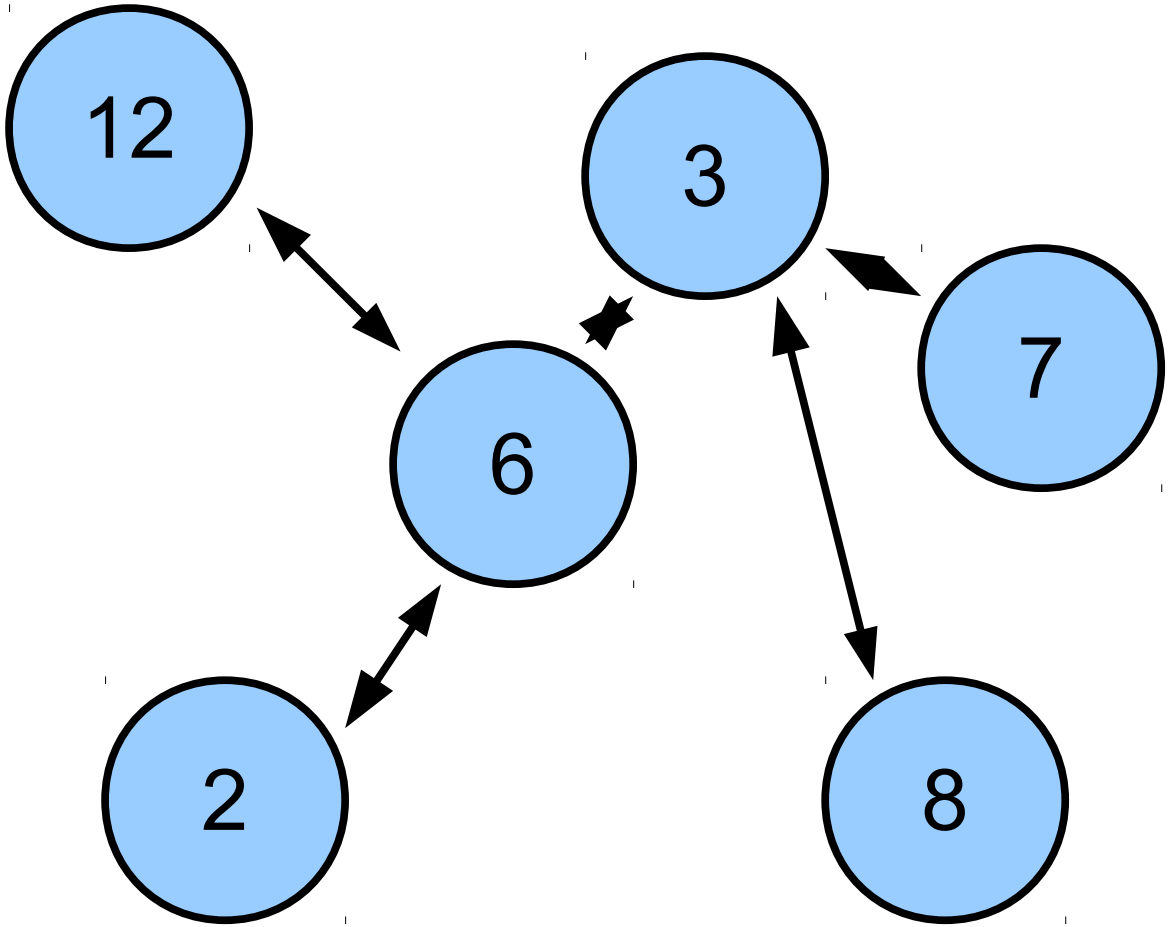
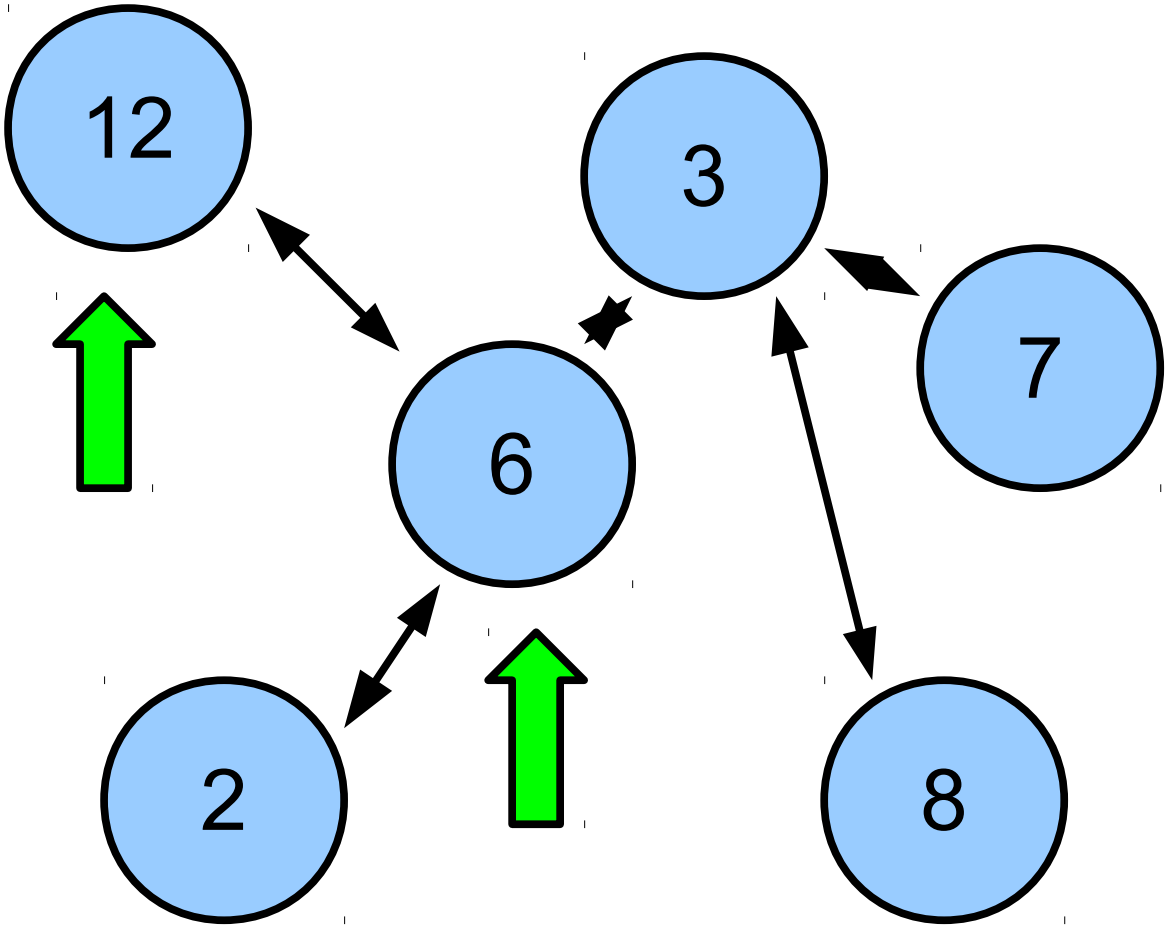


Illustration of averaging



(12+6)/2=9

Illustration of averaging

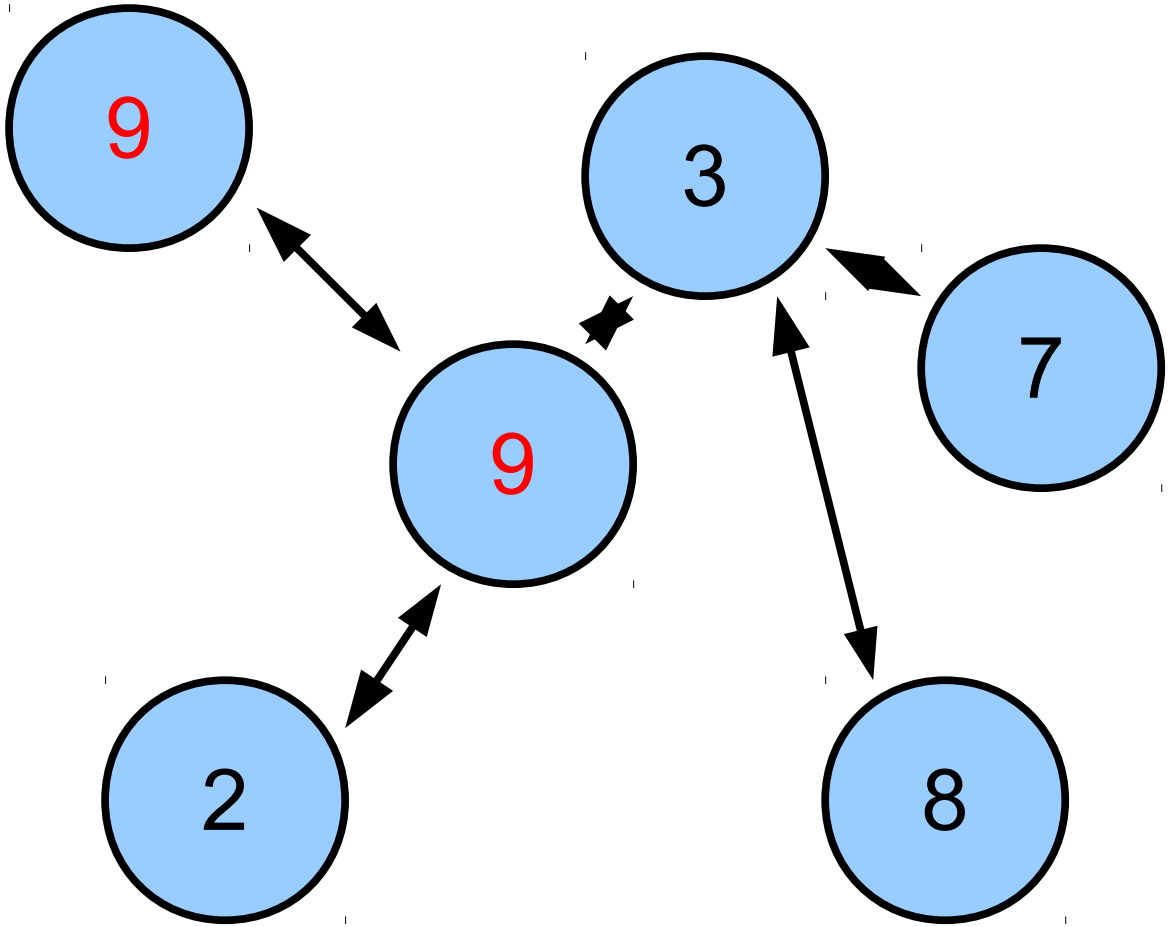
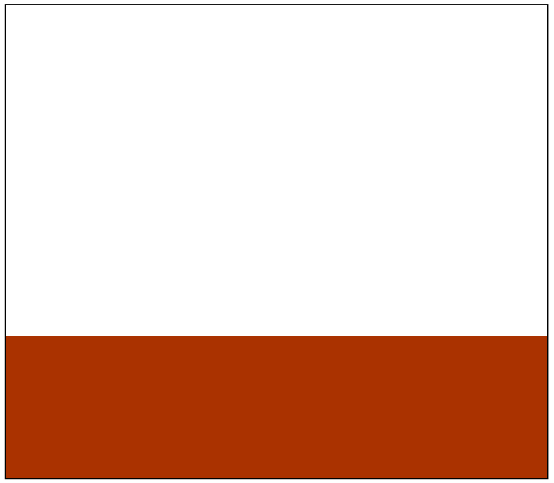
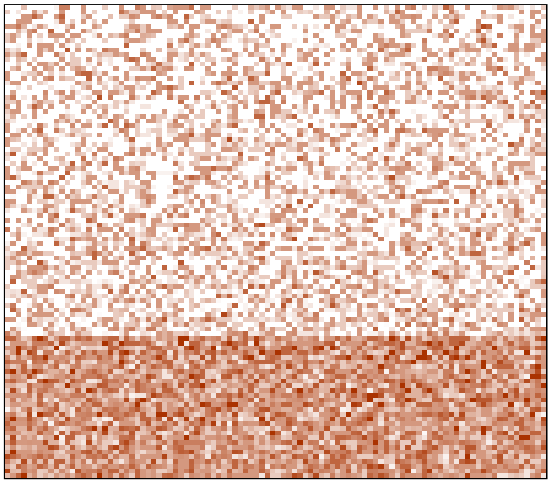


Illustration of averaging

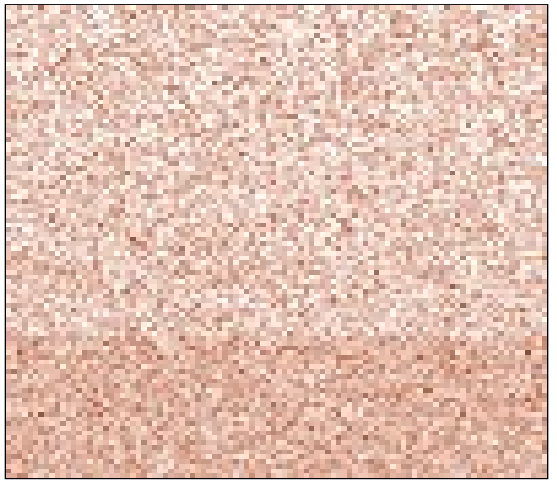
Initial state



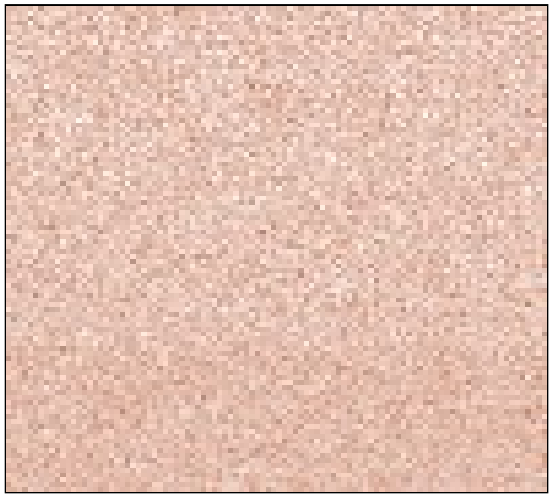
Cycle 1



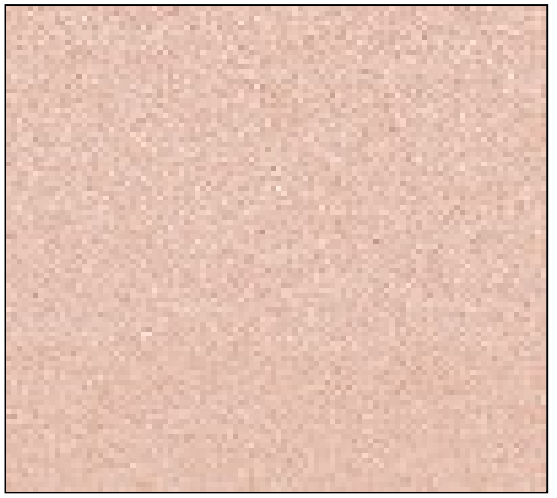
Cycle 2



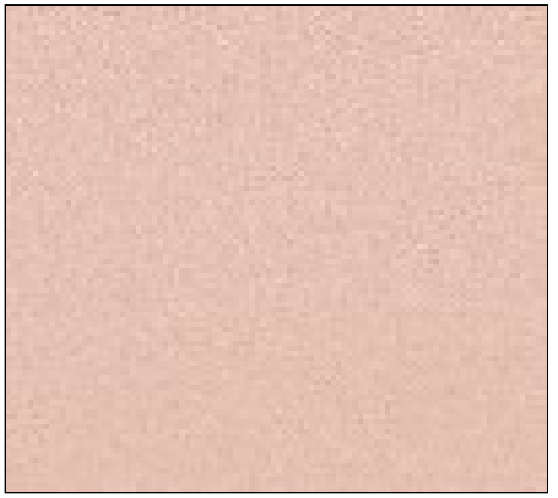
Cycle 3



Cycle 4



Cycle 5



Main intuition

- Mass conservation: the sum of approximations in the network is constant
- Variance reduction: in all steps, the variance of the approximations decreases (if the participating two values are different)

Some theoretical properties

- If y_i is the original value, and x_i is the current estimate at node i, and $\bar{y}$ is the correct average then let us define

$$\sigma_N^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{y})^2$$

- Then it can be shown that

$$E(\sigma_N^2(t+1)) \leq \frac{1}{2}\sigma_N^2(t)$$

Some applications

- We can calculate max (and min) if we replace (m.x+x)/2 with max(m.x, x) (or min(m.x, x))
- Apart from the average, in general we can calculate the f-mean

$$g(z_1, \ldots, z_N) = f^{-1}\left(\frac{\sum_{i=1}^{N} f(z_i)}{N}\right)$$

- f(x)=ln x gives geometric mean, f(x)=1/x gives harmonic mean, $f(x)=x^m$ gives the power mean

Some applications

- Using the averages of powers we can approximate the variance and higher moments, eg $\sigma_N^2 = \bar{y^2} - \bar{y}^2$
- Network size can be approximated if one node sets its value to 1 and the other nodes set it to 0: then the average is 1/N
- Sum can be calculated, since it is the average multiplied by the network size

Improvements

- Tolerates asymmetric message loss (only push or pull) badly
- Tolerates overlaps in pairwise exchanges badly
- [Kempe et al 2003] propose a slightly different version: push averaging
  - all nodes maintain s (sum estimate) and w (weight)
  - estimate is s/w
  - only push: send (s/2,w/2), and keep s=s/2, w=w/2
- several other variations exist

Push averaging

```
1: loop
2:     wait(Δ)
3:     p ← random peer
4:     send (x/2, w/2) to p
5:     x ← x/2
6:     w ← w/2
7: end loop
```

```
8: procedure ONUPDATE(m)
9:     x ← m.x + x
10:    w ← m.w + w
11: end procedure
```

Main intuition

- Mass conservation: the sum of x values in the network is constant, as well as the sum of w values (which is N).
- Variance reduction: the is no similarly simple clear intuition; however, a variance-like potential function Φ_t can be shown to decrease in expectation

$$\Phi_t = \sum_{i,j} \left(v_{t,i,j} - \frac{w_{t,i}}{n} \right)^2$$

Some applications

- Min and max can be calculated without w
- The f-mean and moments can be calculated
- Sum can be calculated by setting w=1 at exactly one node, and w=0 at all the other nodes
- Network size can also be calculated (set x=1 at all nodes and calculate sum)

EM Gaussian mixture

- k-component Gaussian mixture

$$p(x) = \sum_{s=1}^{k} \pi_s p(x|s)$$

$$p(x|s) = (2\pi)^{-d/2}|C_s|^{-1/2}\exp[-(x-m_s)^\top C_s^{-1}(x-m_s)/2]$$

- We want to estimate the parameters θ to maximize log likelihood

$$\theta = \{\pi_s, m_s, C_s\}_{s=1}^{k}$$

$$\mathcal{L} = \sum_{i=1}^{n} \log p(x_i;\theta)$$

EM Gaussian mixture

- Each data point x_i has a distribution $q_i(s)$ that describes the "level of responsibility" of each Gaussian for generating x_i
- The EM algorithm iterates between estimating qi (E-step) and θ (M-step)
- Similar to k-means clustering

EM Gaussian mixture

- E-step: $q_i(s)=p(s|x_i,\theta)$
- M-step

$$\pi_s = \frac{\sum_{i=1}^{n} q_i(s)}{n}$$

$$m_s = \frac{\sum_{i=1}^{n} q_i(s) x_i}{n\pi_s}$$

$$C_s = \frac{\sum_{i=1}^{n} q_i(s) x_i x_i^\top}{n\pi_s} - m_s m_s^\top$$

Gossip-based implementation

- Assume that each node i stores a vector x_i, and a global k is known. Let's run a distributed EM!
  - Initialize the qi values arbitrarily
  - Perform M-step by calculating the three averages

$$\pi_s = \frac{\sum_{i=1}^{n} q_i(s)}{n} \qquad \frac{\sum_{i=1}^{n} q_i(s) x_i}{n} \qquad \frac{\sum_{i=1}^{n} q_i(s) x_i x_i^\top}{n}$$

    and the use these values to calculate θ
  - Perform E-step (completely local operation)

A Gossip Skeleton

- Originally for information dissemination in a very simple but efficient and reliable way
- Later the gossip approach has been generalized resulting in many local probabilistic and periodic protocols
- we will introduce a simple common skeleton and look at
  - information dissemination
  - topology construction
  - aggregation

A Push-Pull Gossip Skeleton

```
1: loop
2:     wait(Δ)
3:     p ← selectPeer()
4:     send push(state) to p
5: end loop
```

```
6: procedure ONPUSHUPDATE(m)
7:     send pull(state) to m.sender
8:     state ← update(state,m.state)
9: end procedure
10:
11: procedure ONPULLUPDATE(m)
12:     state ← update(state,m.state)
13: end procedure
```

Rumor mongering as an instance

- state: set of active updates
- selectPeer: a random peer from the network
  - very important component, we get back to this soon
- update: add the received updates to the local set of updates
- propagation of one given update can be limited (max k times or with some probability, as we have seen, etc)

Peer Sampling

- A key method is selectPeer in all gossip protocols (influences performance and reliability)
- In earliest works all nodes had a global view to select a random peer from
  - scalability and dynamism problems
- Scalable solutions are available to deal with this
  - random walks on fixed overlay networks
  - dynamic random networks

Random walks on networks

- if we are given any fixed network, we can sample the nodes with any arbitrary distribution with the Metropolis algorithm:

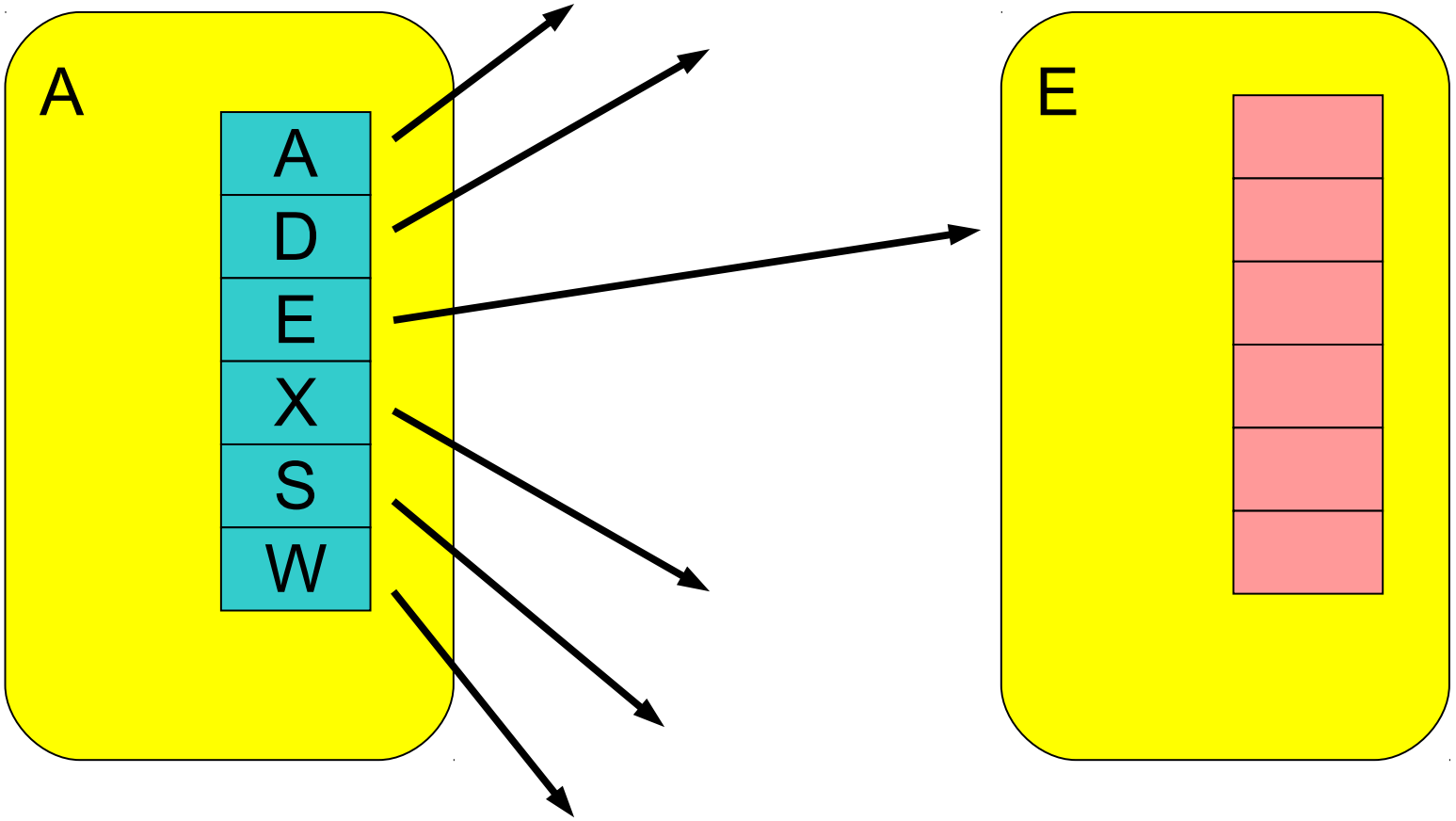
$$P_{i,j} = \begin{cases} \frac{1}{2} \cdot \frac{1}{d_i} & \text{if } \frac{\pi(i)}{d_i} \leq \frac{\pi(j)}{d_j}; \\ \frac{1}{2} \cdot \frac{1}{d_j} \cdot \frac{\pi(j)}{\pi(i)} & \text{if } \frac{\pi(i)}{d_i} > \frac{\pi(j)}{d_j}. \end{cases}$$

- This Markov chain has stationary distribution π where d_i is the degree of node i (undirected graph)

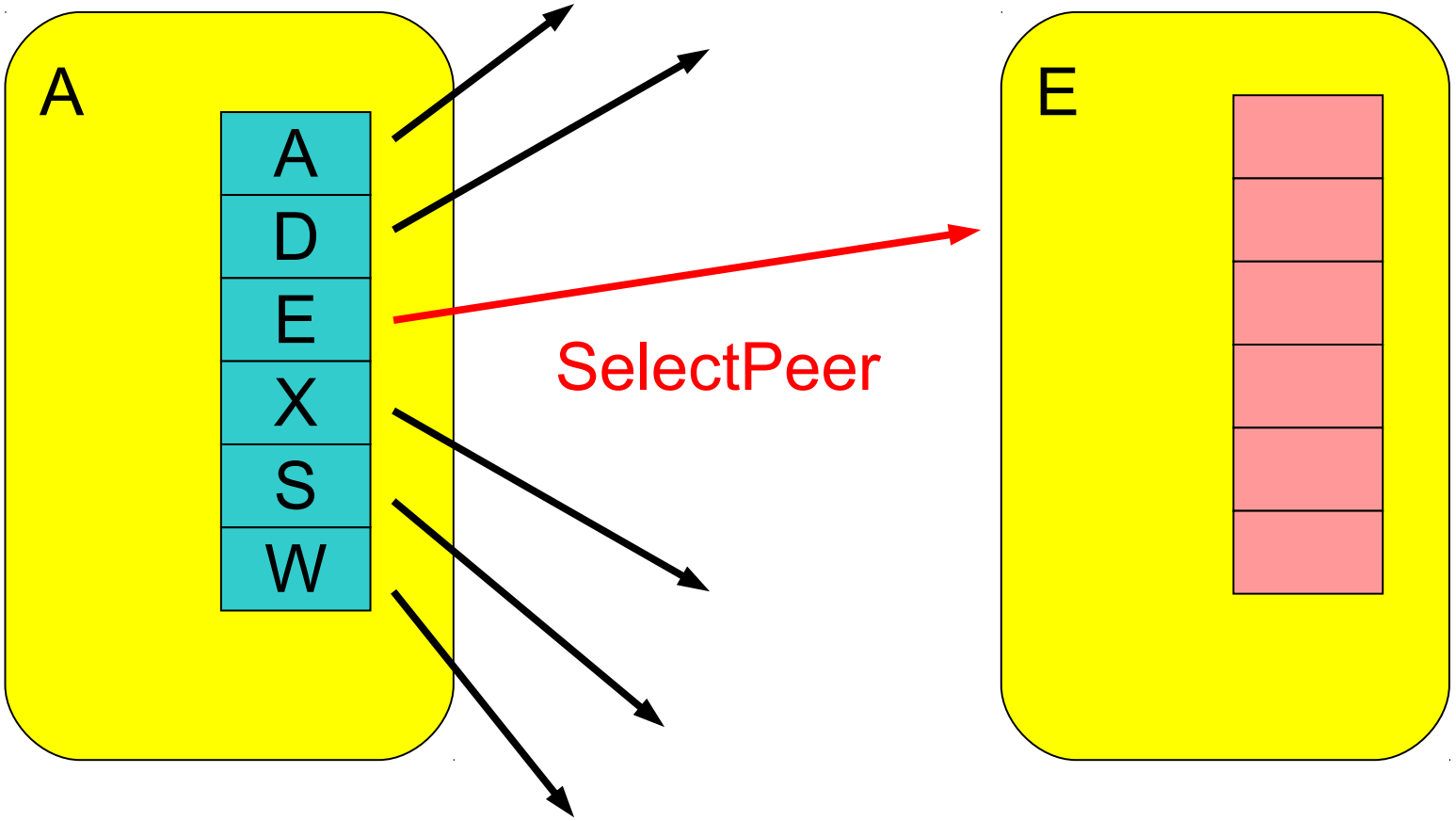
Gossip based peer sampling

- basic idea: random peer samples are provided by a gossip algorithm: the peer sampling service
- The peer sampling service uses itself as peer sampling service (bootstrapping)
  - no need for fixed (external) network
- state: a set of random overlay links to peers
- selectPeer: select a peer from the known set of random peers
- update: for example, keep a random subset of the union of the received and the old link set

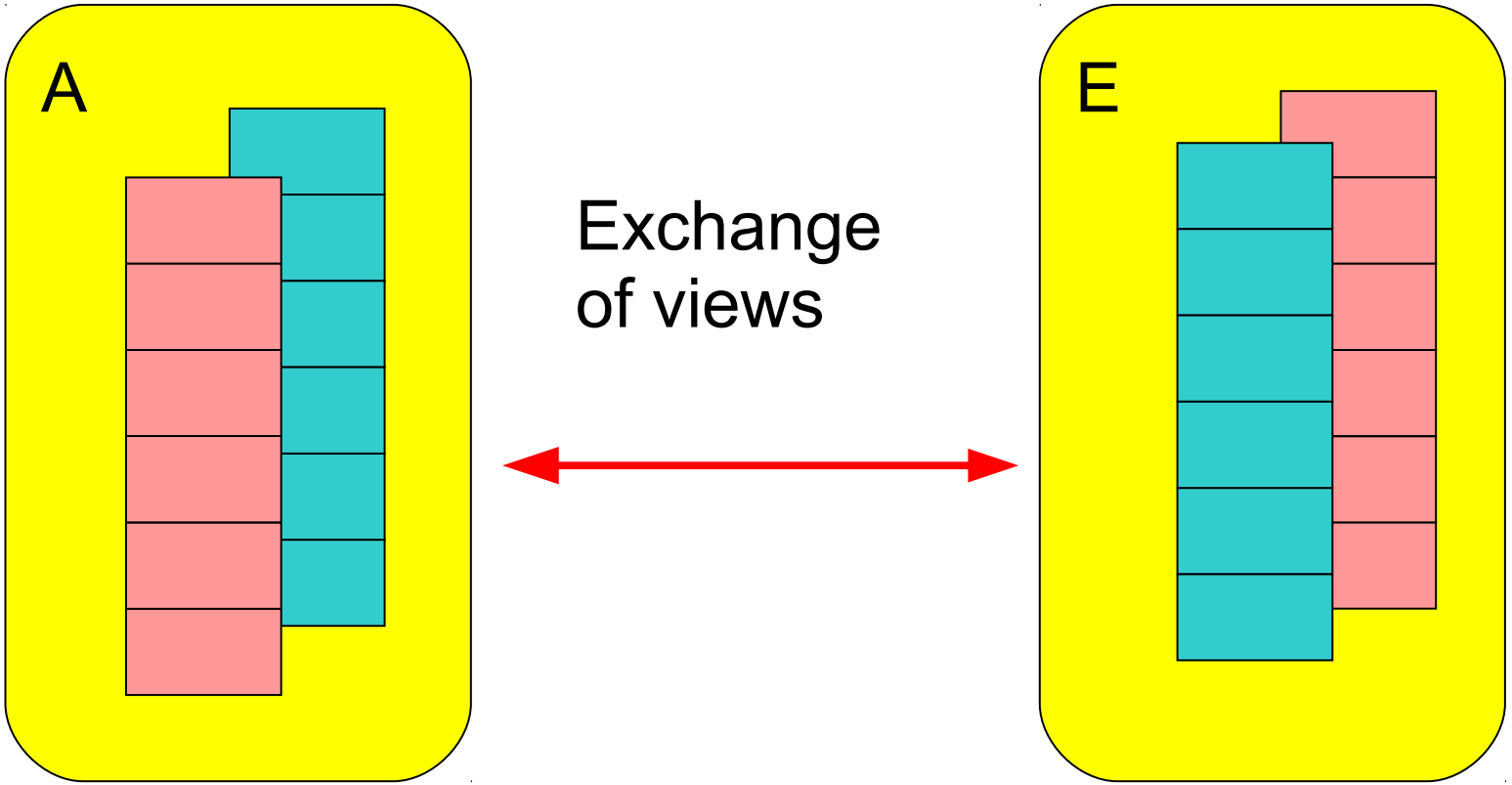
Gossip protocols for topology management



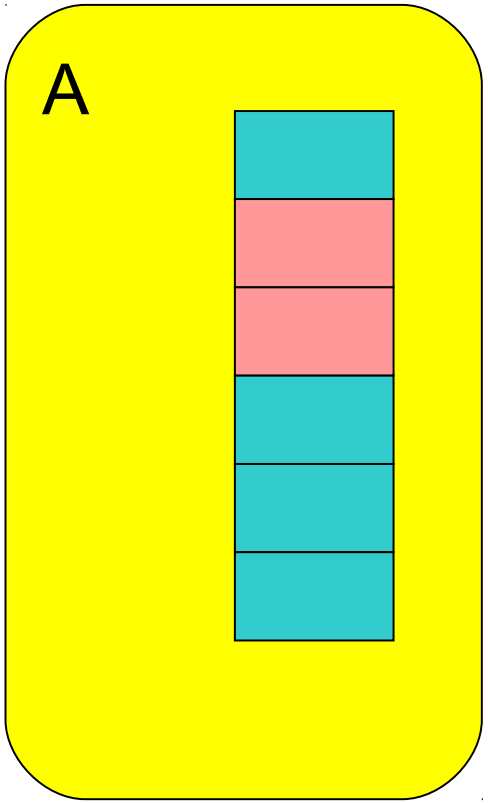
Gossip protocols for topology management



Gossip protocols for topology management

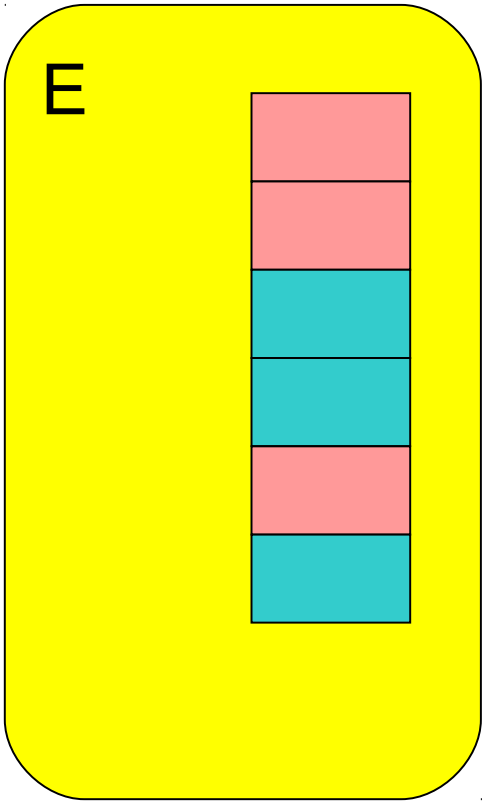


Gossip protocols for topology management



Both sides apply update

thereby redefining topology



Gossip based peer sampling

- in reality a huge number of variations exist
  - timestamps on the overlay links can be taken into account: we can select peers with newer links, or in update we can prefer links that are newer
- these variations represent important differences w.r.t. fault tolerance and the quality of samples
  - the links at all nodes define a random-like overlay that can have different properties (degree distribution, clustering, diameter, etc)
  - turns out actually not really random, but still good for gossip

Gossip based topology management

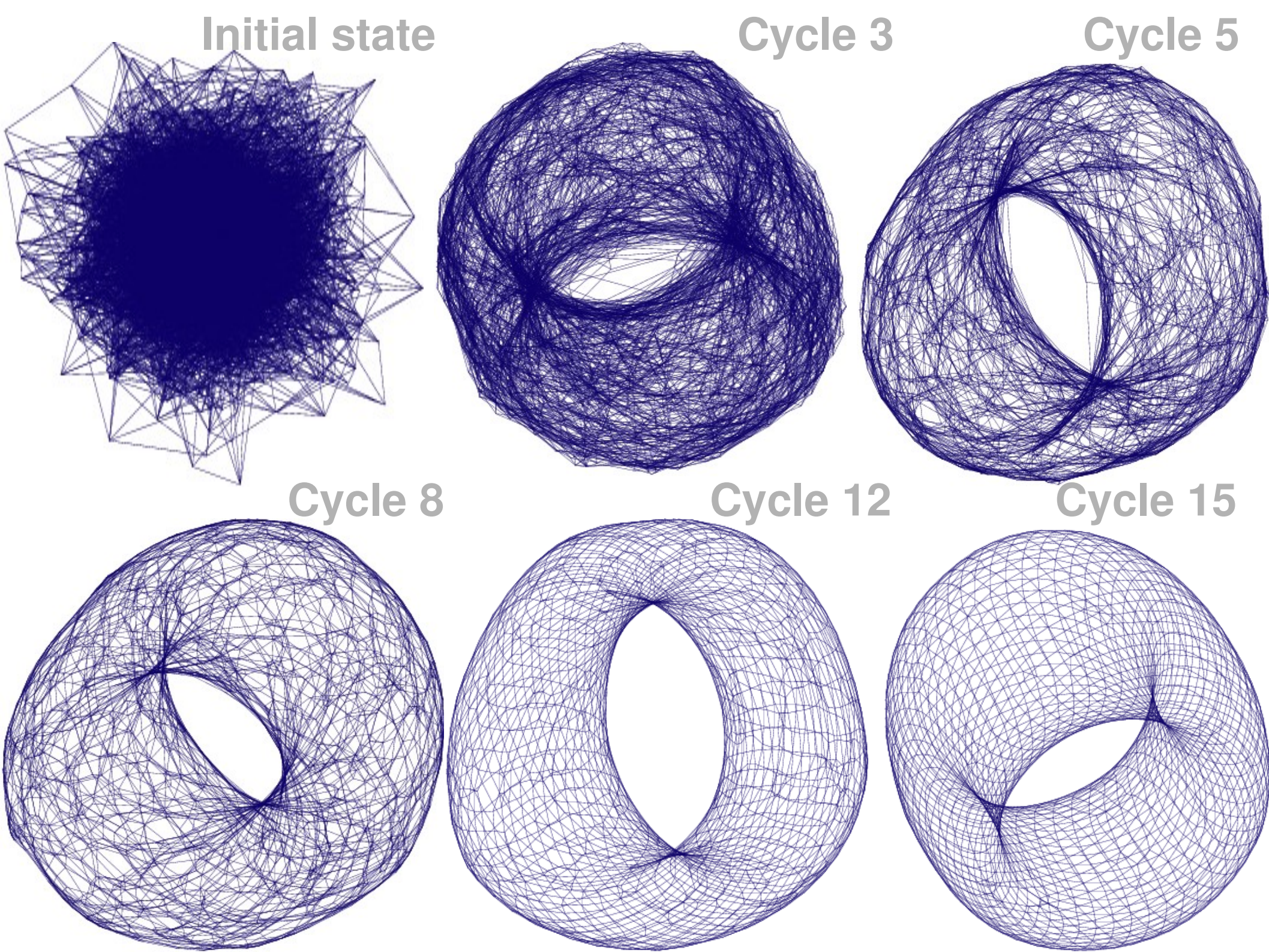
- We saw we can build random networks. Can we build any network with gossip?
- Yes, many examples
  - proximity networks
  - DHT-s (Bamboo DHT: maintains Pastry structure with gossip inspired protocols)
  - semantic proximity networks
  - etc

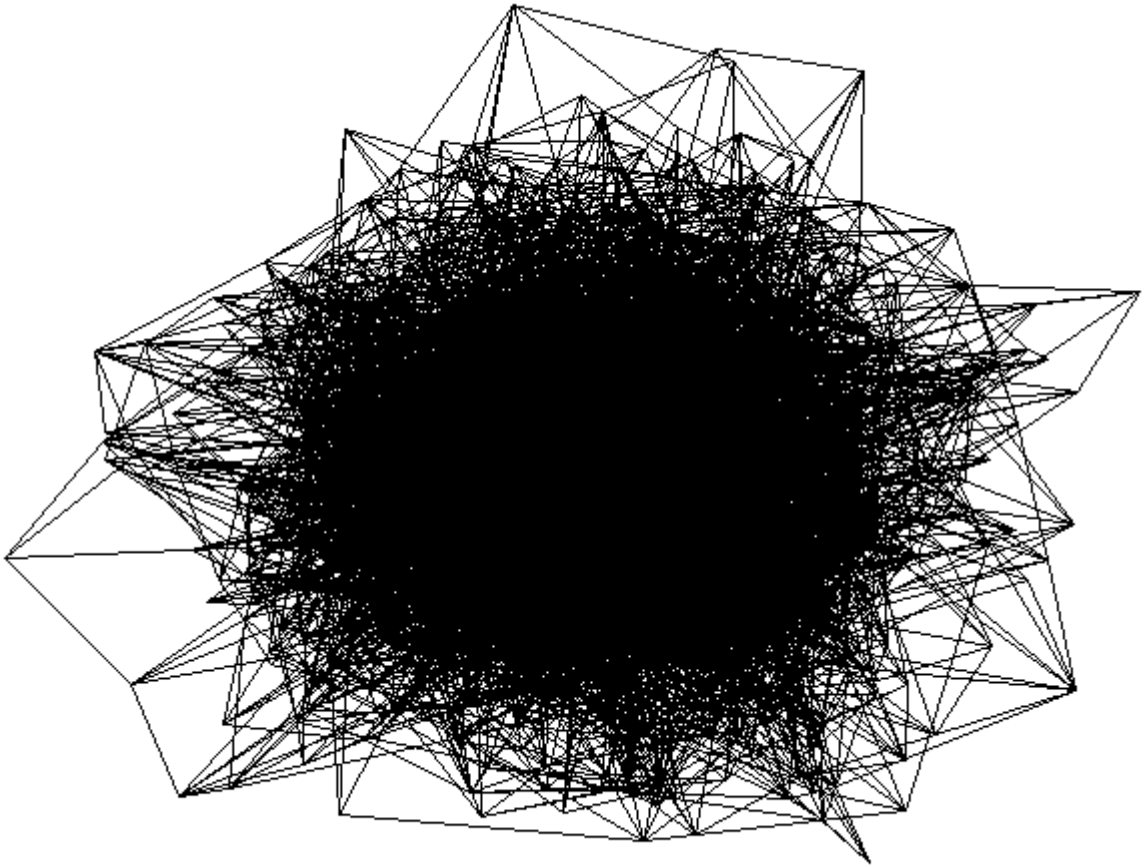
T-Man

- T-MAN is a protocol that captures many of these in a common framework, with the help of the ranking method:
  - ranking is able to order any set of nodes according to their desirability to be a neighbor of some given node
  - for example, based on hop count in a target structure (ring, tree, etc)
  - or based on more complicated criteria not expressible by any distance measure

Gossip based topology management

- basic idea: random peer samples are provided by a gossip algorithm: the peer sampling service
- state: a set of overlay links to peers
- selectPeer: select the peer from the known set of peers that ranks highest according to the ranking method
- update: keep those links that point to nodes that rank highest


Initial state
Cycle 3
Cycle 5
Cycle 8
Cycle 12
Cycle 15



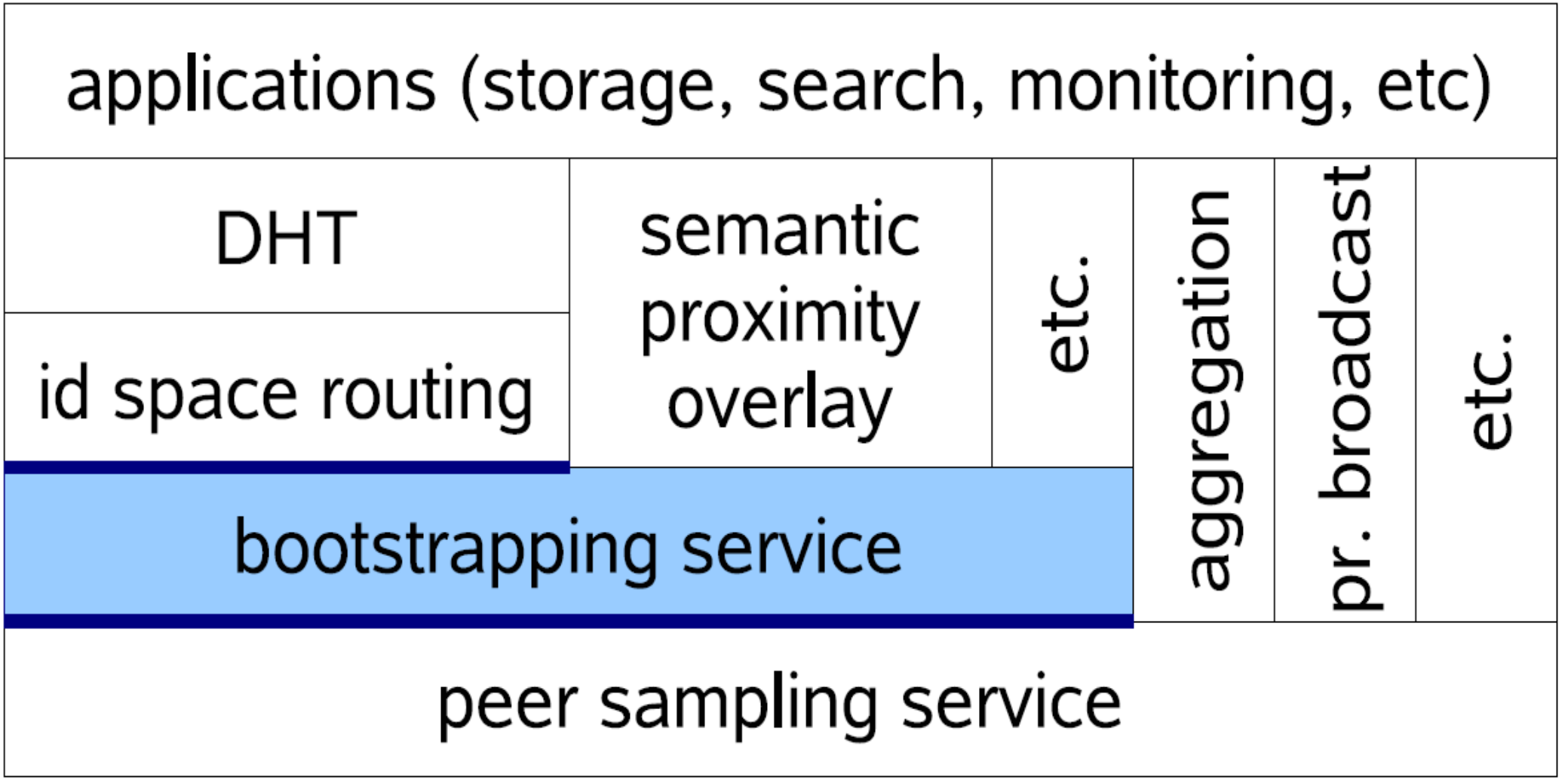
Some other examples

- firefly-inspired synchronization
- partitioning (slicing) and sorting in P2P networks
- asynchronous implementation of matrix iterations
  - ranking (PageRank)
  - reputation systems
- emergent cooperation

Modular design

- We have seen that all gossip protocols use the peer sampling service that is itself a gossip protocol
- Can be generalized: gossip protocols can be stacked or arbitrarily combined
  - actual local communication is the same (all protocols can often piggyback the same message)
  - conceptual structure is modular

Example modular architecture



Outlook

- Gossip is similar to many other fields of research that also have some of the following features:
  - periodic, local, probabilistic, symmetric
- examples include
  - swarm systems, cellular automata, parallel asynchronous numeric iterations, self-stabilizing protocols, etc

A slide on viruses and worms

- We focused on “good” epidemics but malicious applications are known
  - viruses and worms replicate themselves via similar algorithms using some underlying network such as email contacts or the Internet itself
- The dynamics is described by SIS model
- Underlying networks are typically scale free (power law degree distribution)
  - can be proven: no threshold: it is nearly impossible to completely eliminate a “disease”

Some open problems

- gossip in mobile contact networks and its potential applications (also malware...)
- security
  - gossip is robust to benign failure but very sensitive to malicious attacks
  - current “secure” gossip protocols sacrifice simplicity and light-weight
- interdisciplinary connections: toward a deeper understanding of self-organization and gossip protocols as a special case