BSI
Unstructured Overlays: Gossip and Epidemics

Davide Frey
ASAP Team, INRIA Rennes

Gossip (Wikipedia)

Gossip consists of casual or idle talk of any sort, sometimes (but not always) slanderous and/or devoted to discussing others.

While gossip forms one of the oldest and (still) the most common means of spreading an

reputation for the intro … o the

information thus transm

Reliable way of spreading information

Epidemic (Wikipedia)

In epidemiology, an **epidemic** is a disease that appears as new cases in a given human population, during a given period, at a rate that substantially exceeds what is “expected”.

Non-biological usage:

The term is often use

widespread and growi

Efficient way of spreading something

Gossip/epidemics in distributed computing

Replace

- people by computers (nodes or peers),
- words by data

We retain

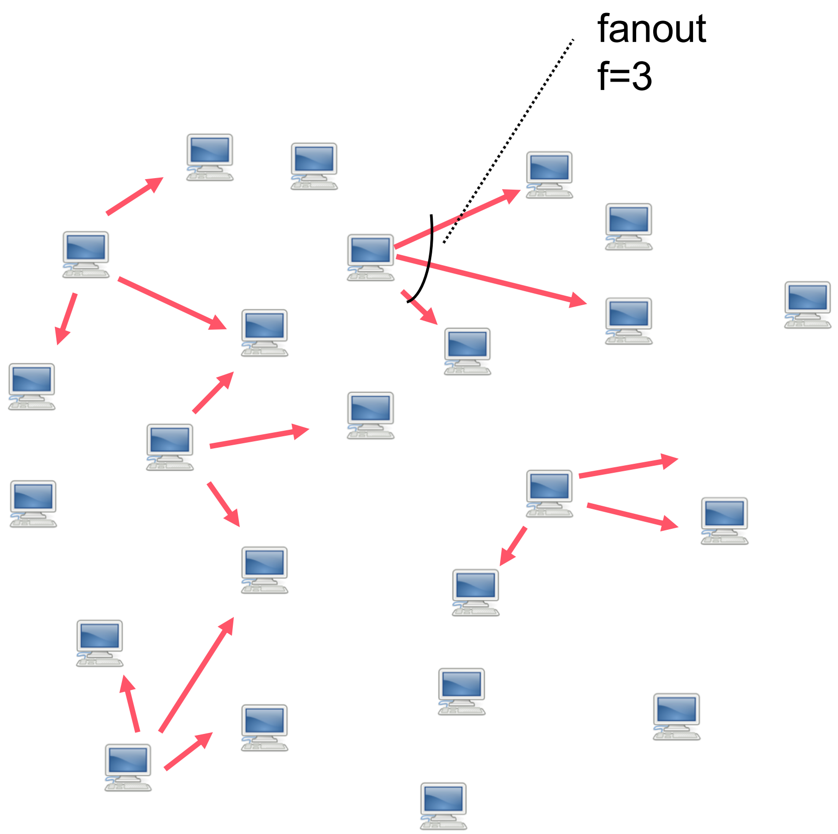
- Gossip: peerwise exchange of information
- Epidemic: wide and exponential spread

Refer to gossip in the following

Gossip / Epidemic Protocols

Fundamental tool for decentralized applications

- Completely decentralized
- Periodic pairwise exchanges
- Some form of randomness



Why Gossip

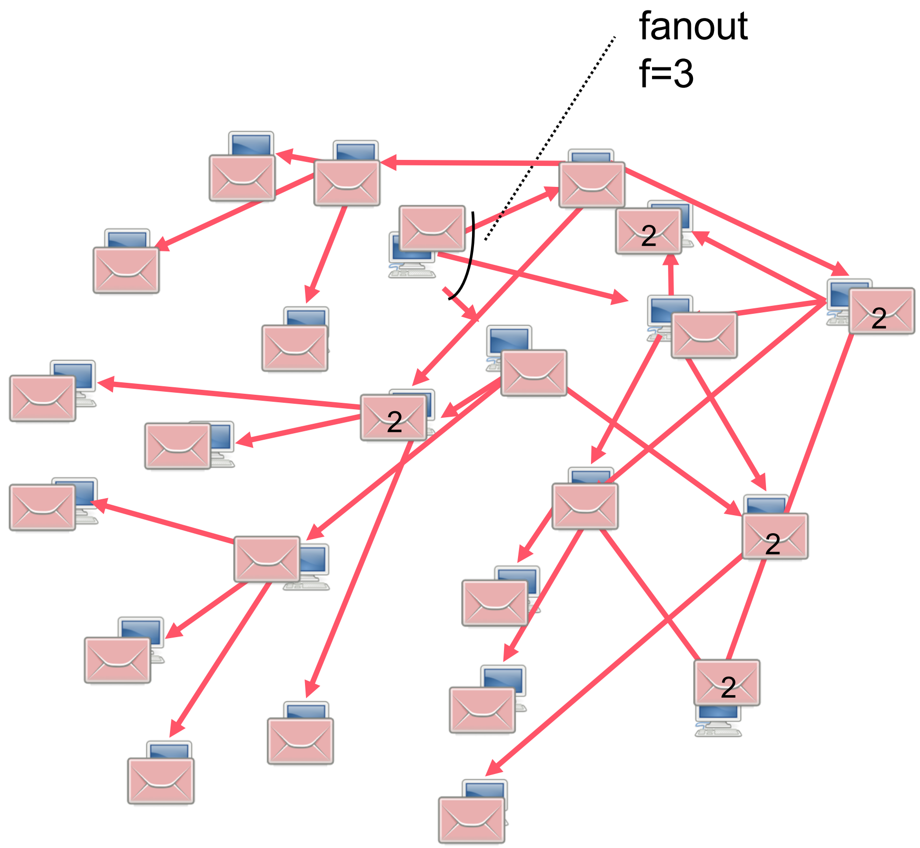
Scenario:

- Very Large scale Systems
- Lots of data
- Continuous Changes

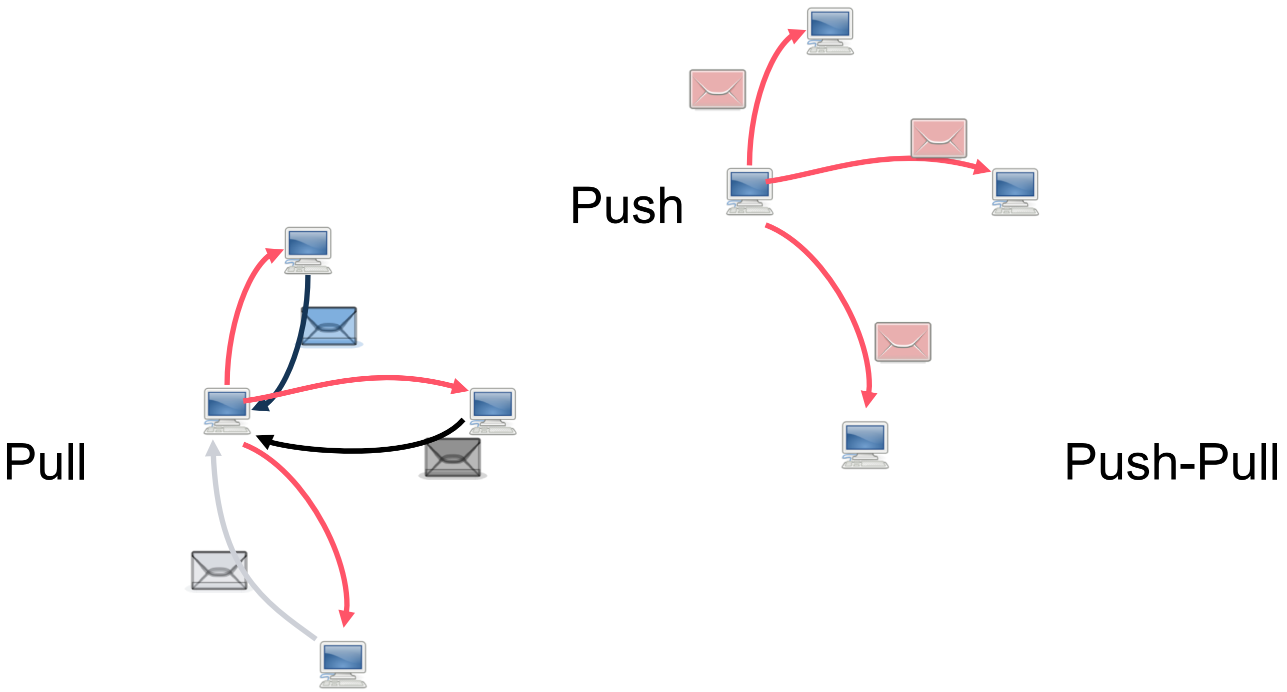
Gossip:

- Peer to peer communication: no unique point of failure
- Eventual convergence
- Probabilistic nature

Gossip for Data Dissemination

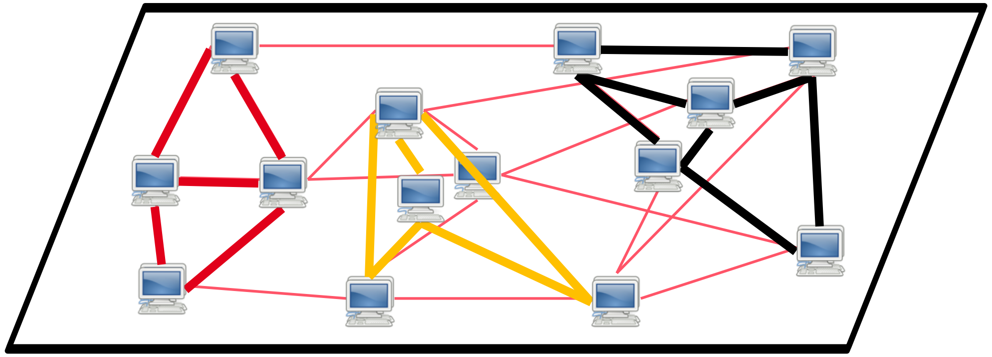


Gossip for Data Dissemination



Gossip for Overlay Maintenance

- Overlay Maintenance
  - Random Peer Sampling
  - Clustered Topologies: KNN
    - Similarity metric



So What Makes a Gossip Protocol

- Some form of randomization
- Some periodic behavior
- Exchange of messages of bounded size

Strengths:

- Simplicity
- Emergent structure
- Convergence
- Robustness

Weaknesses:

- Overhead
- Vulnerability to malicious behavior

Applications of Gossip

Consistency Management
[Demers &al, PODC

Epidemic disseminatio
Bimodal Multicast [Birman&al, ACM
[Kermarrec&al, IEEETPDS
Lpbcast [Eugster&al DSN01, ACM
Stream[Patel & al, NCA 20

Content-based search
Vicinity[Voulgaris & Steen,Euro-Par 05]
VoroNet [Beaumont & al, IPDPS 07]
RayNet[Beaumont & al, OPODIS 07]

Aggregation
[Jelasity&al, ACM TOCS 05]
Astolabe [Birman & al, 2003]

Slicing
[Jelasity, Kermarrec, P2P06]
[Fernandez & al, ICDCS07]

Overlay maintenance
Lpbcast [Eugster & al,ACM TOCS 03]
Cyclon[Voulgaris& al, 2005]
Newscats[Jelasity & al, 2003]

Publish-subscribe
Sub-2-Sub [Voulageris & al. IPTPS06]
Tera[Bald

Clustering
Vicinity, Jstream, Tman, Gossple

Streaming
BAR Gossip [Li & al, OSDI06]
Middleware 2009]

Secure Sampling
Brahms [Bortnikov & al, 08]

Recommendation
Gossple[Bertier & al, Middleware 2010]
WhatsUp[Boutet & al, IPDPS 2013]

Plan for the Following

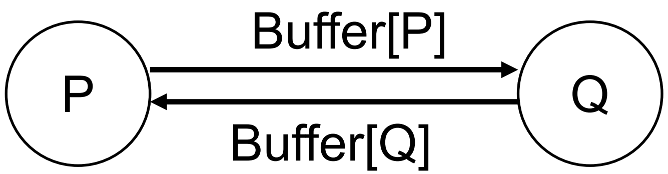
- Gossip Basics
- Overlay Maintenance
  - Random peer sampling
  - Clustering

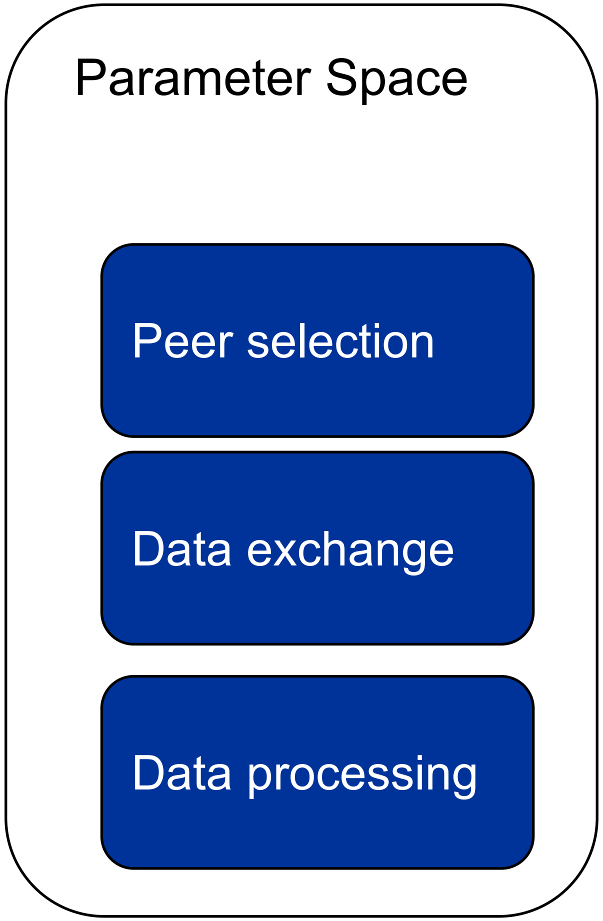
Generic Gossip Protocol

Each node maintains a set of neighbors (c entries)

Periodic peer-wise exchange of information

Each process runs an active and passive threads





Generic Gossip Protocol

Active Cycle

Periodically

Peer selection

- Select a/some peer(s) p
- Select some data D
- Send D to p

Data exchange

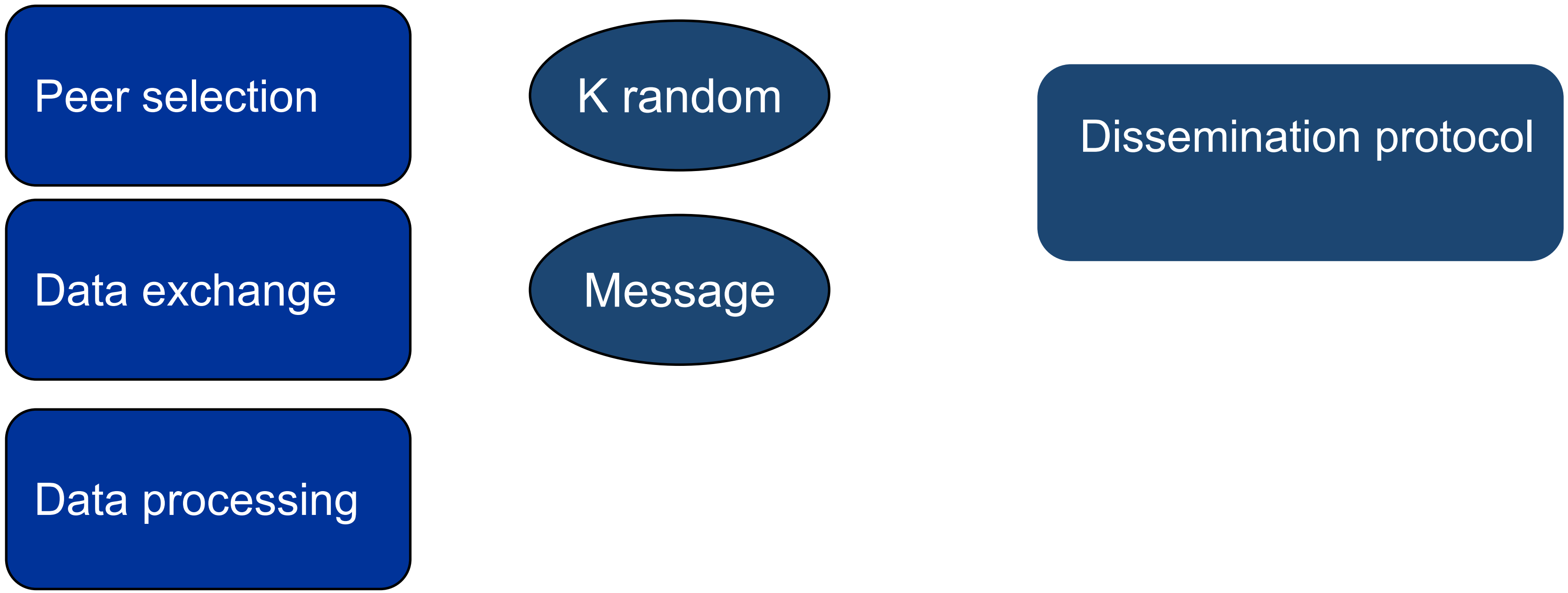
Passive Cycle

Upon message M from p

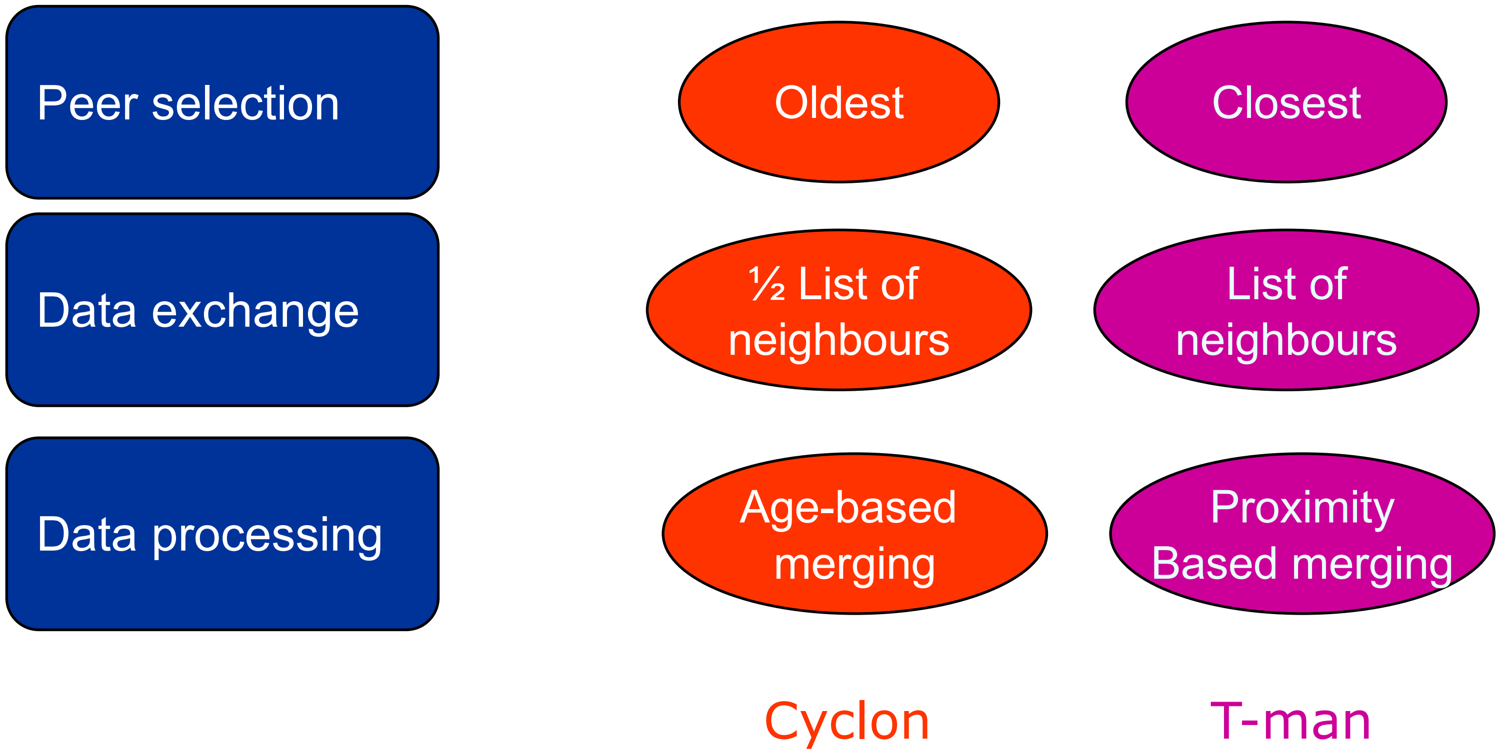
- Incorporate M into own state
- If (M not a response)
  - Select some data D
  - Send D to p

Data processing

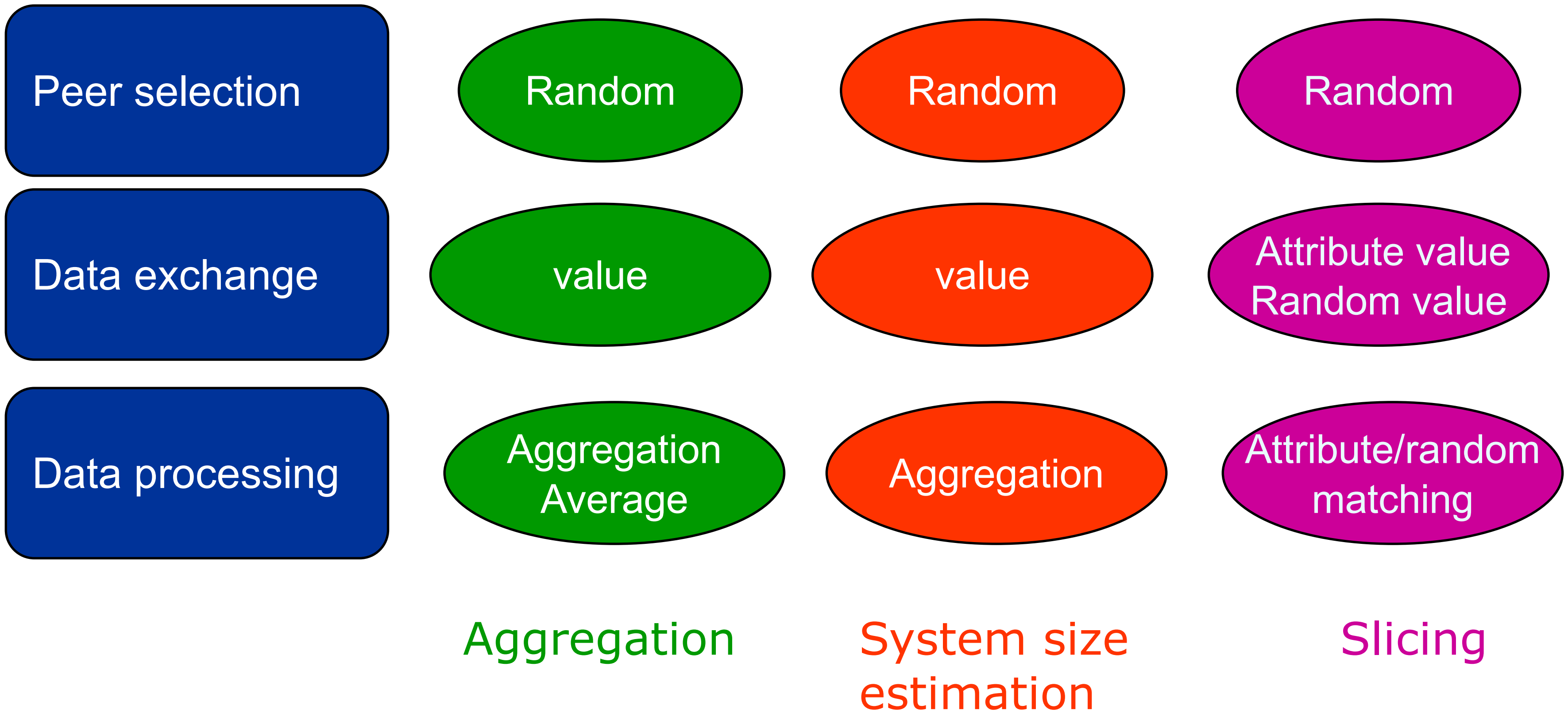
Dissemination



Overlay maintenance



Decentralized computations



Epidemic-based dissemination

Goal:
Broadcast reliably to a large number of peers

System model:

- *n* processes
- Each process forwards the message once to *f* (fanout) neighbors, picked up uniformly at random.
- Alternatively f times to 1 neighbor.

Success metrics:

- Proportion of infected processes

  $Y_r = Z_r / n$

  Z_r is the number of infected processes prior to round r
- Probability of atomic "infection"

  $P(Z_r = n)$

Proportion of infected processes

Large system of size n

Probabibility that the epidemic catches (1-p_{ext})

Proportion of processes eventually contaminated

$\pi = 1 - e^{-\pi f}$ where f is the fanout

Independent of n, a fixed average of descendants will lead to the same proportion of infected processes

Probability of atomic infection

Erdos/Renyi examine final system state, the system is represented as a graph where each node is a process, there is an edge from n_1 to n_2 if n_1 is infected and chooses n_2.

An epidemic starting at n_0 is successful if there is a path from n_0 to all members. If the fanout is $\log(n) + c$, the probability that a random graph is connected is

$$p(connect) = e^{-e^{-c}}$$

Other measures

Latency of infection

[Bollobas, Random Graphs, Cambridge University Press, 2001]

Logarithmic number of rounds

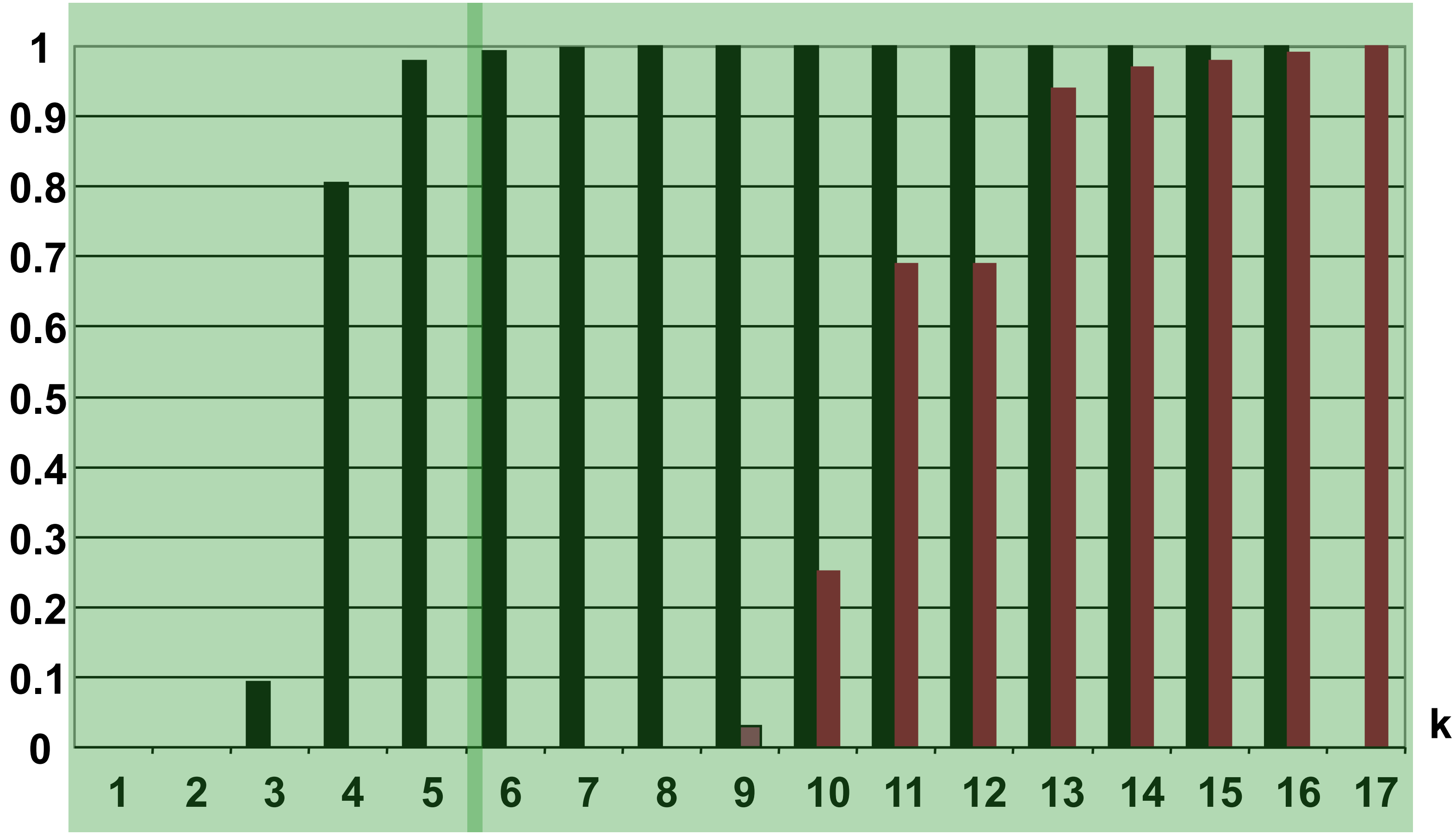
$$R = \frac{\log(n)}{\log(\log(n))} + O(1)$$

Resilience to failure

[KMG, IEEE Tpds 14(3), Probabilistic reliable dissemination in Large-scale systems, 2003]

$$k = (n / n')[\log(n') + c + O(1)]$$

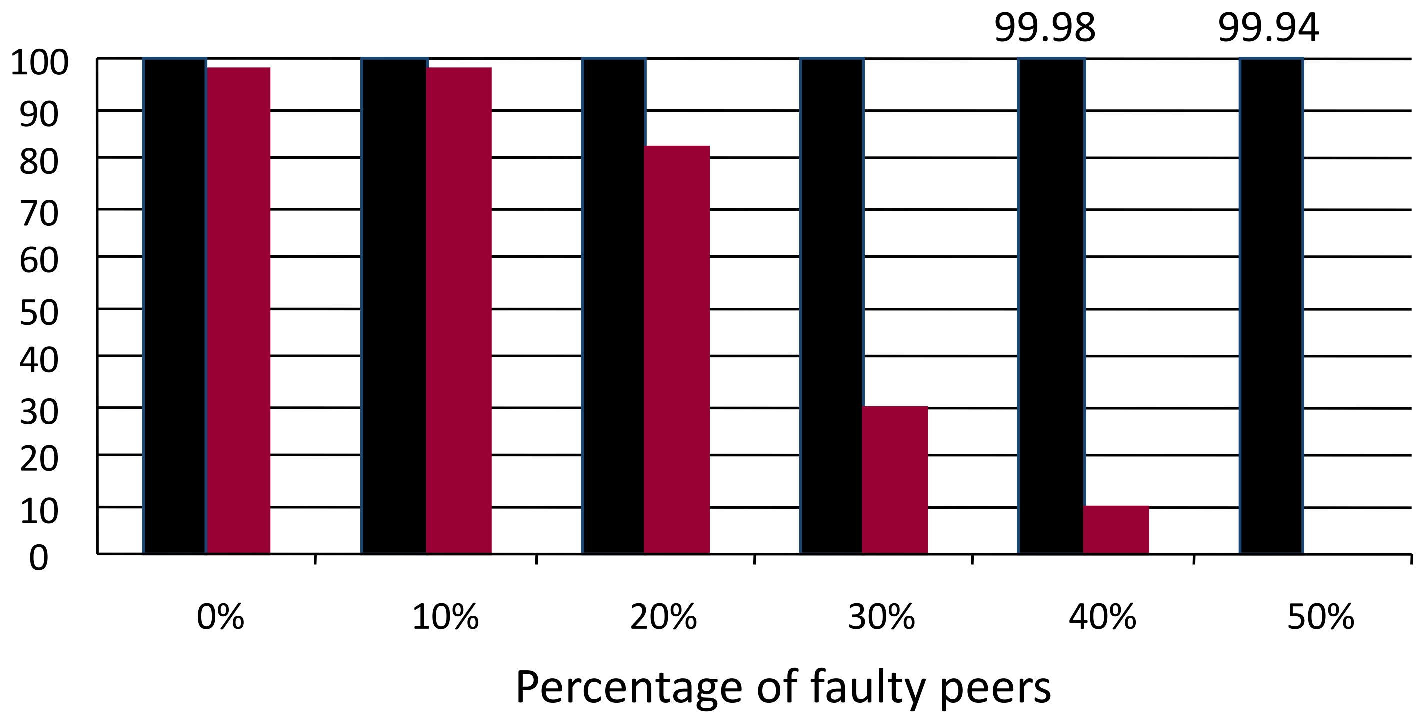
Performance (100,000 peers)



Proportion of “atomic”infection / reliable broadcast

Proportion of connected peers in non reliable broadcast /non-atomic infection

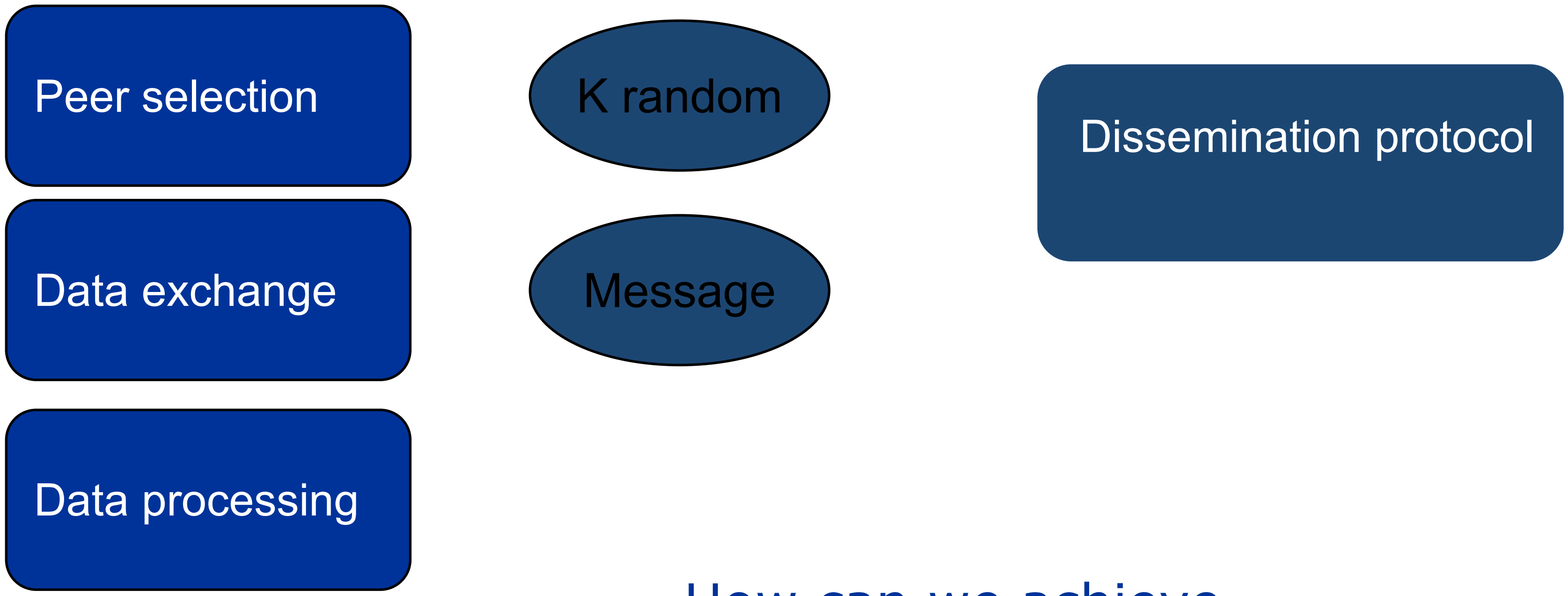
Failure resilience (100,000 peers)



Proportion of "atomic" broadcast

Proportion of connected peers in non "atomic" broadcast

Dissemination relies on Random Sampling



How can we achieve
Random sampling?

Today

- Gossip Basics
- Overlay Maintenance
  - Random peer sampling
  - Clustering

Gossip Overlays: Random Peer Sampling

Goal:

- Provide each peer with a continuously changing random sample of the network.

Effect:

- Overlay consists of a continuously changing random-like graph

The Peer Sampling Service

Creates unstructured overlay network topologies

Interface

- *Init()*: service initialization
- *GetPeer():* returns a peer address, ideally drawn uniformly at random

The Peer Sampling service

System Model

- System of *n* peers
- Peers join and leave (and fail) the system dynamically and are identified uniquely (IP @)
- Epidemic interaction model:
  Peers exchange some membership information periodically to update their own

Data Structures

- Each peer maintains a view (membership table) of *c* entries
  - Network @ (IP@)
  - Timestamp (freshness of the descriptor)

Protocol

Active Cycle
Periodically

Peer selection

P <- selectPeer()

myDescriptor <- (my@, now)
buffer <- merge (view,
    {myDescriptor})

send buffer to p

Data exchange
(View Propagation)

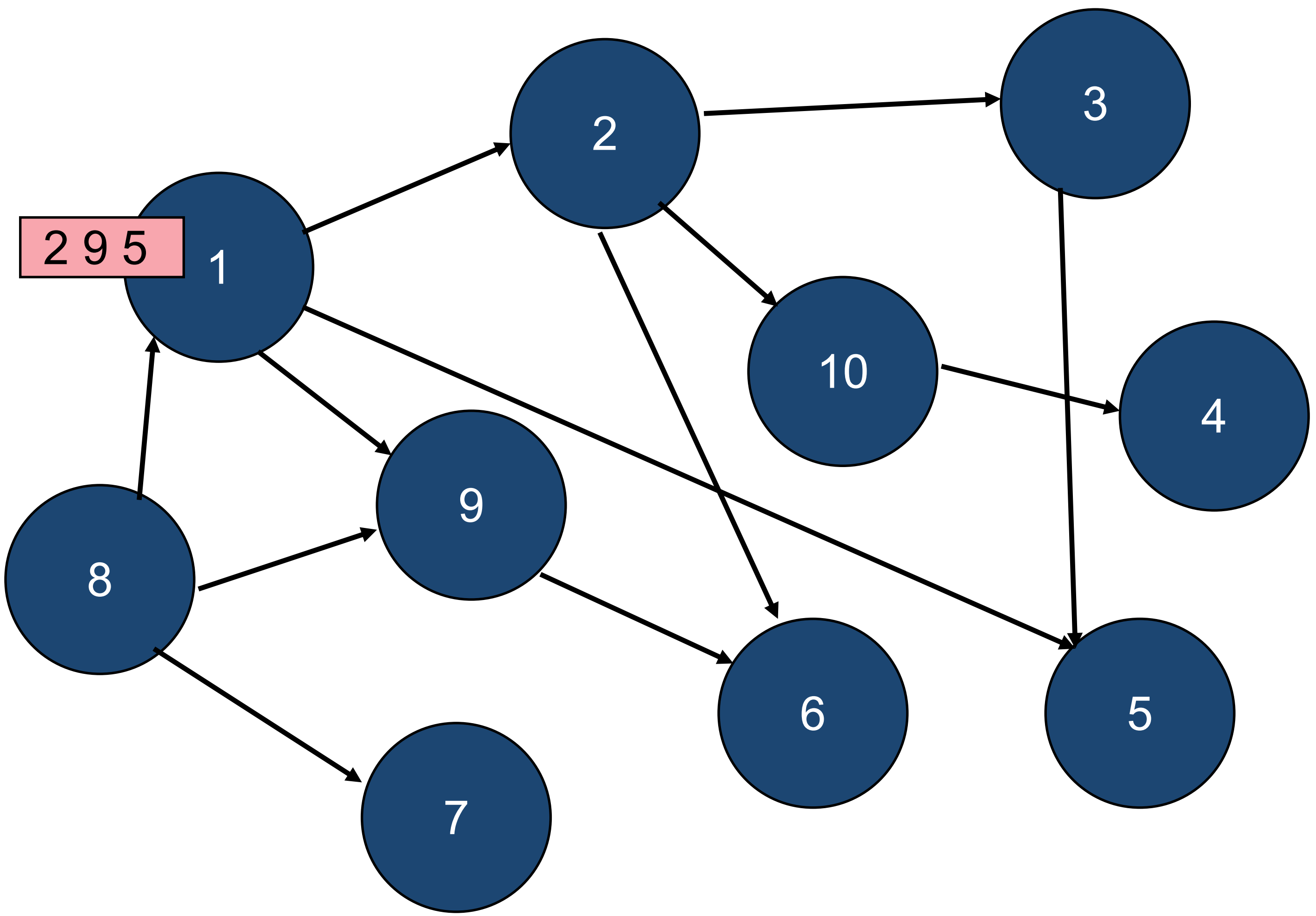
Passive Cycle
When message received from p

buffer <- merge(view_p, view)
View <-selectView(buffer)

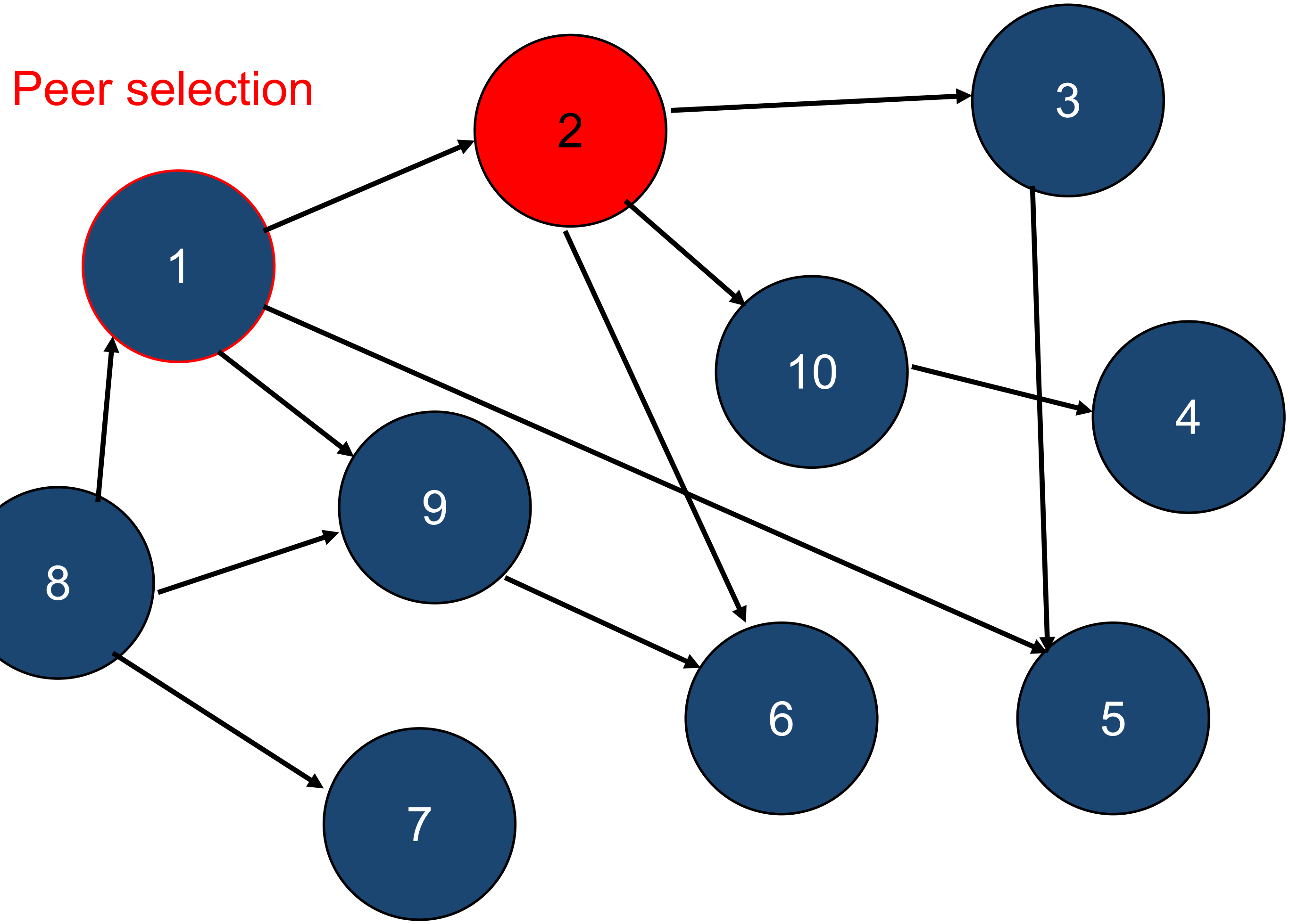
if pull **and** not receiving response **then**
  myDescriptor <-(my@, now)
  buffer <-merge(view,{myDescriptor})
  send buffer to p

Data processing
(View Selection)

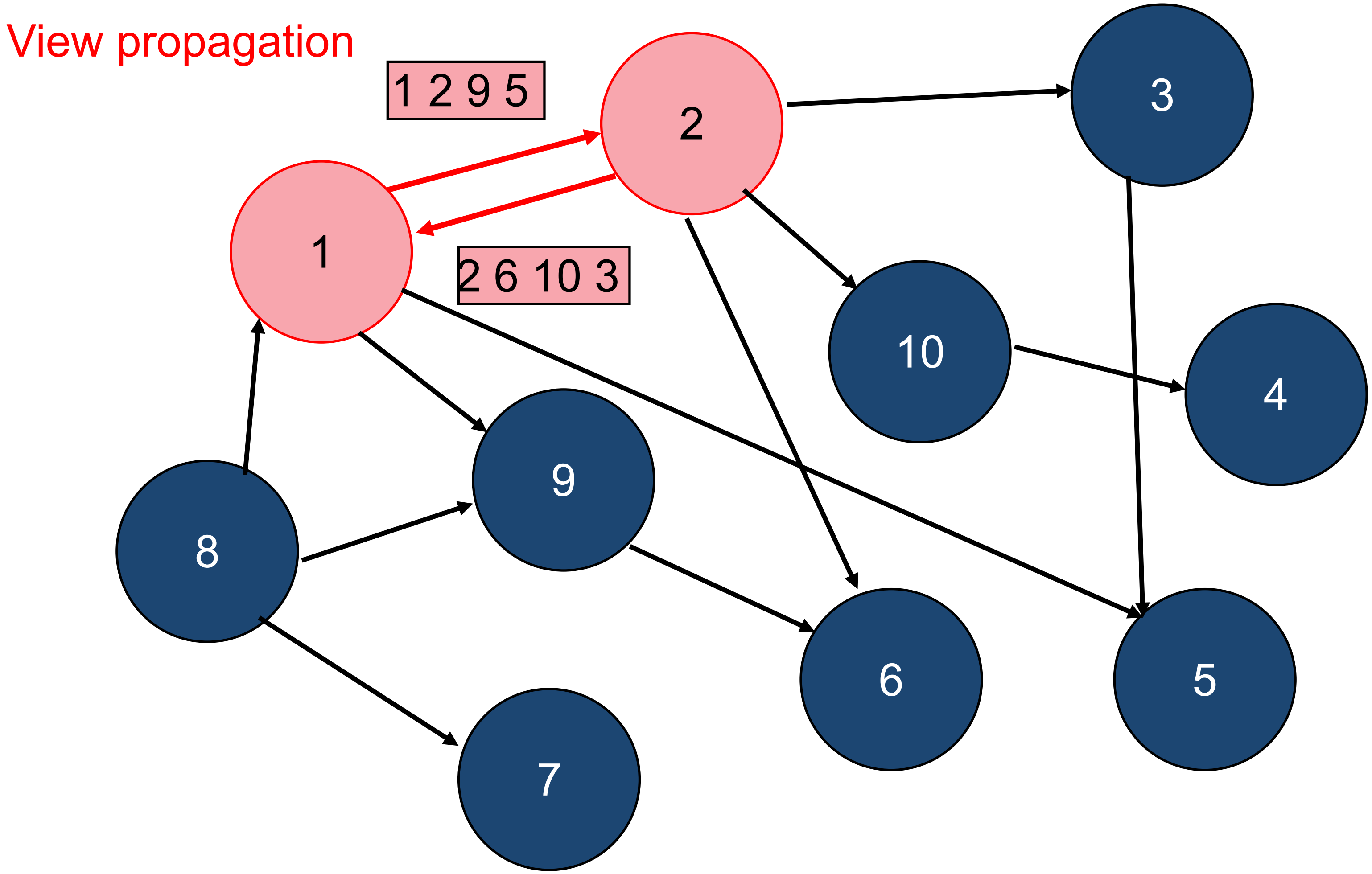
Generic protocol



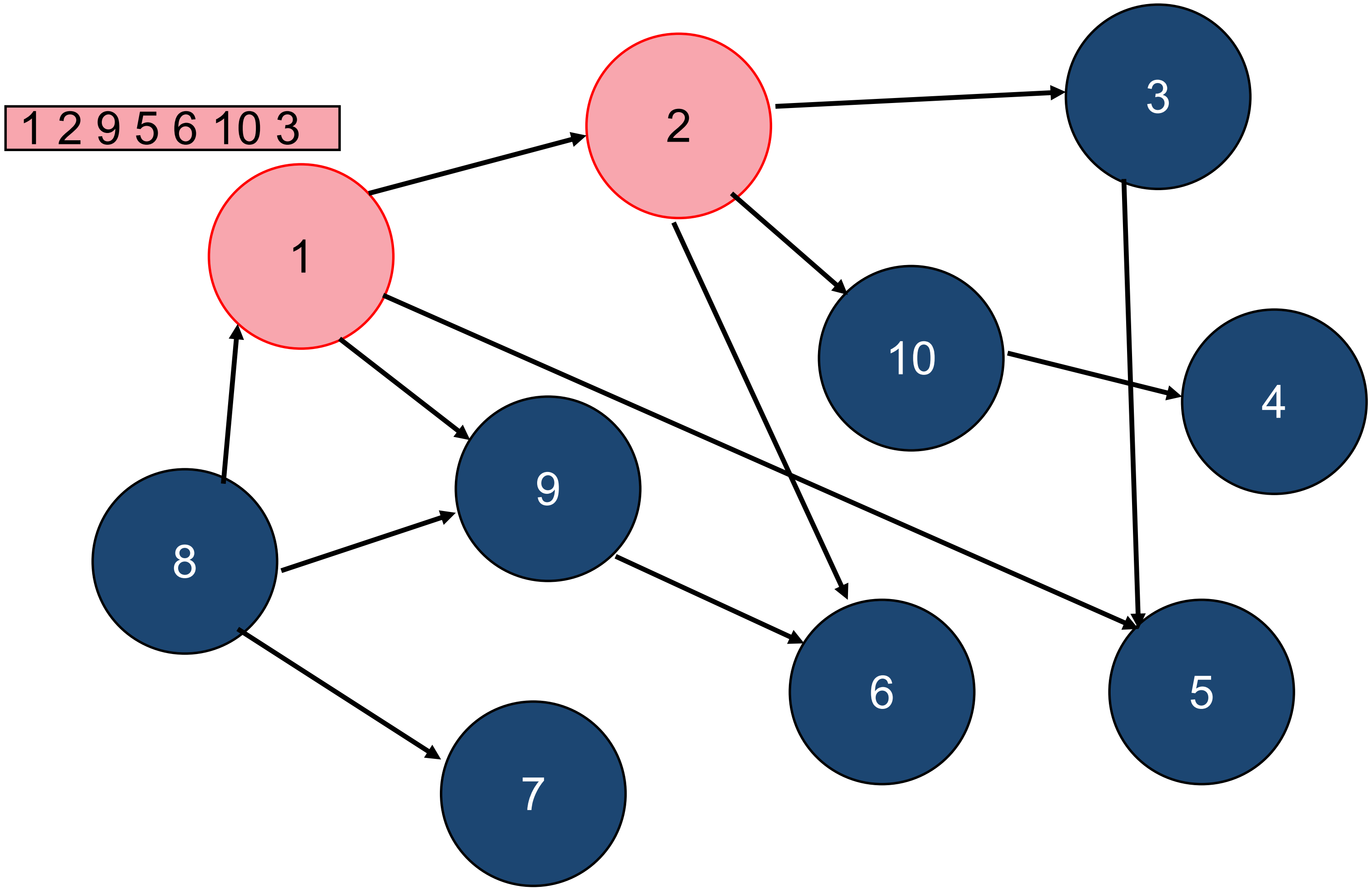
Generic protocol



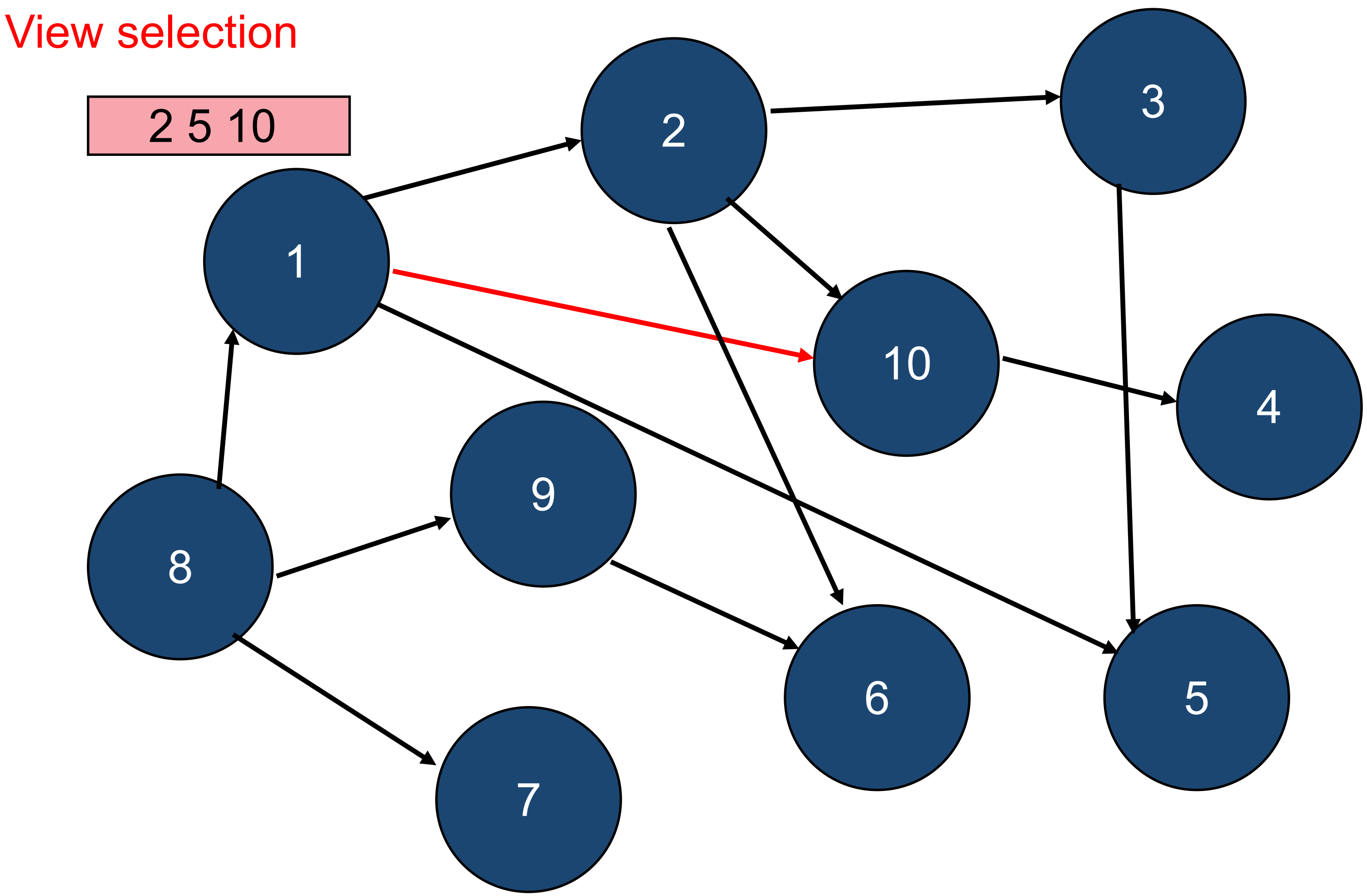
Generic protocol



Generic protocol



Generic protocol



Protocol

Active Cycle
Periodically

Peer selection

P <- selectPeer()

myDescriptor <- (my@, now)
buffer <- merge (view, {myDescriptor})

send buffer to p

Data exchange
(View Propagation)

Passive Cycle
When message received from p

buffer <- merge(view_p, view)
View <-selectView(buffer)

if pull **and** not receiving response **then**
myDescriptor <-(my@, now)
buffer <-merge(view,{myDescriptor})
send buffer to p

Data processing
(View Selection)

Design space

- **Peer selection**

Periodically each peer initiates communication with another peer

- **Data exchange (View propagation)**

How peers exchange their membership information?
What do they exchange?

- **Data processing (View selection)**: Select (c, buffer)

c: size of the resulting view
Buffer: information exchanged

Design space: peer selection

Three Strategies

Rand: pick a peer uniformly at random

Head: pick the "youngest" peer

Tail: pick the "oldest" peer

Note that *head* leads to correlated views.

Design space: data exchange

Buffer (h)

initialized with the descriptor of the gossiper

contains *c/2* elements

ignore *h* "oldest"

Two Strategies

Push: buffer sent

Push/Pull: buffers sent both ways

(Pull: left out, the gossiper cannot inject information about itself, harms connectivity)

Design space: Data processing

Select(*c,h,s,buffer*)
1. Buffer appended to view
2. Keep the freshest entry for each node
3. *h* oldest items removed
4. *s* first items removed (the one sent over)
5. Random nodes removed

c: size of the resulting view
H: self-healing parameter
S: shuffle
Buffer: information exchanged

Merge strategies

Blind (*h=0,s=0)*: select a random subset
Healer (*h=c/*2): select the "freshest" entries
Shuffler (*h=0, s=c/*2): minimize loss

Design space summary

Peer selection

rand	Select a peer at random from the view
tail	Select the node with the highest hop count

Head leads to correlated views

View propagation

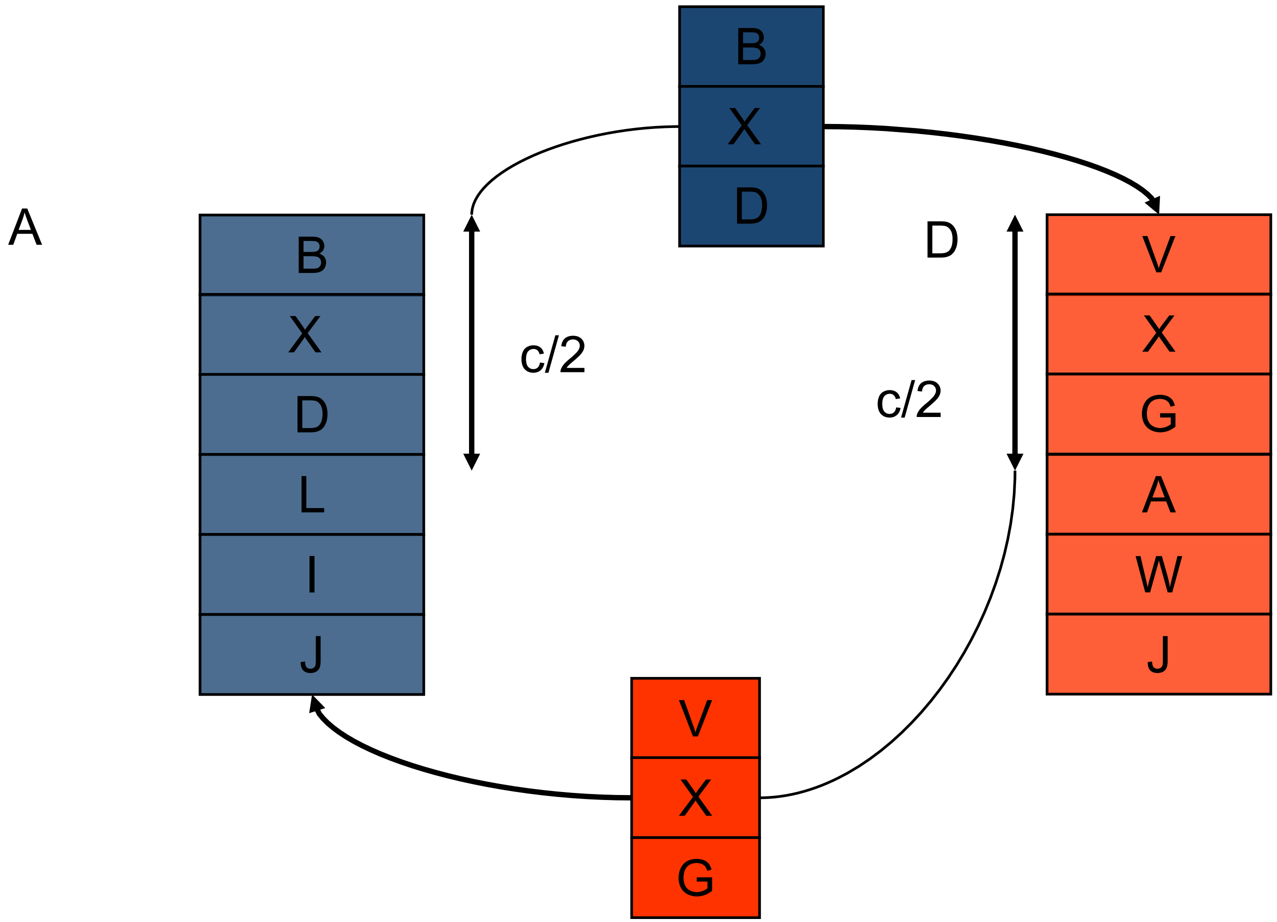
push	The node sends its buffer to the selected peer
pushpull	The node and the selected peer exchange information

Pull: risk of partition (a node has no possibility to inject information about itself)

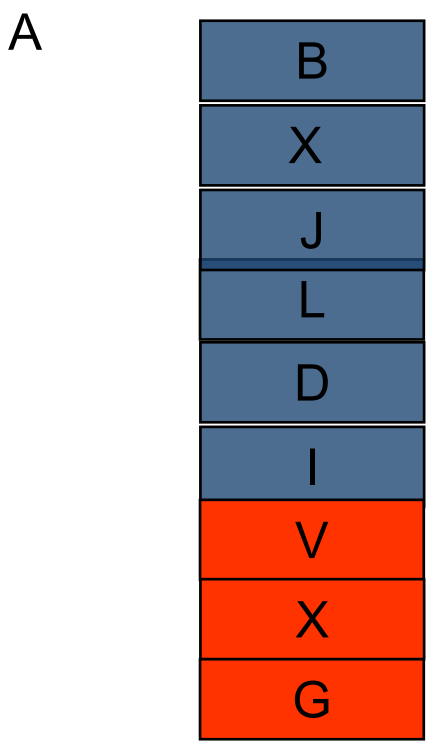
View selection

blind	H = 0, S = 0	Blind selection of a random subset
healer	H = c/2	Select the freshest entries
shuffler	H = 0, S = c/2	Minimize loss of information

Example



Example



1. Buffer appended to view
2. Keep the freshest entry for each node
3. h (=1) oldest items removed
4. s (=1) first items removed (the one sent over)
5. Random nodes removed

Some systems

Lpbcast [Eugster & al, DSN 2001,ACM TOCS 2003]
Peer selection: random
View propagation: push
View selection: random

Newscast [Jelasity & van Steen, 2002]
Peer selection: head
View propagation: pushpull
View selection: head

Cyclon [Voulgaris & al JNSM 2005]
Peer selection: random
View propagation: pushpull
View selection: Shuffle

Experimental Study

- Relationship « who knows who »
  - Highly dynamic
  - Capture quickly changes in the overlay networks
- Protocol Variants
  - Healer (h=c/2, s=0)
  - Shuffler (h=0, s=c/2)
- Scenarios
  - lattice
  - random
  - growing networks
- Metrics
  - Degree distribution
  - Average path length
  - Clustering coefficient

Degree distribution

Out degree = c (30) in 10.000 node system

Distribution of in-degree

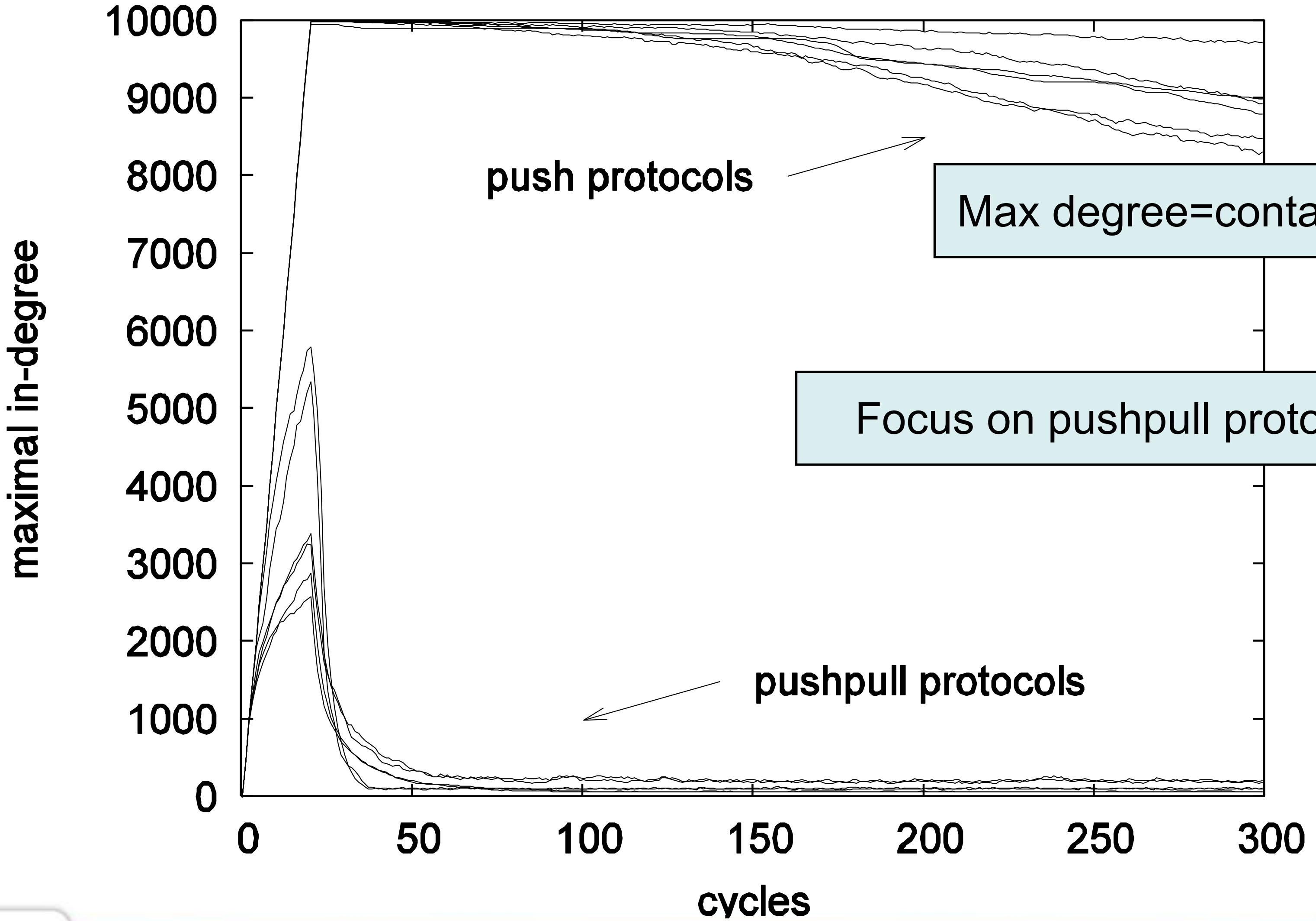
Detect hotspot and bottleneck

Load balancing properties

Convergence

Self-organization ability irrespective of the initial topology

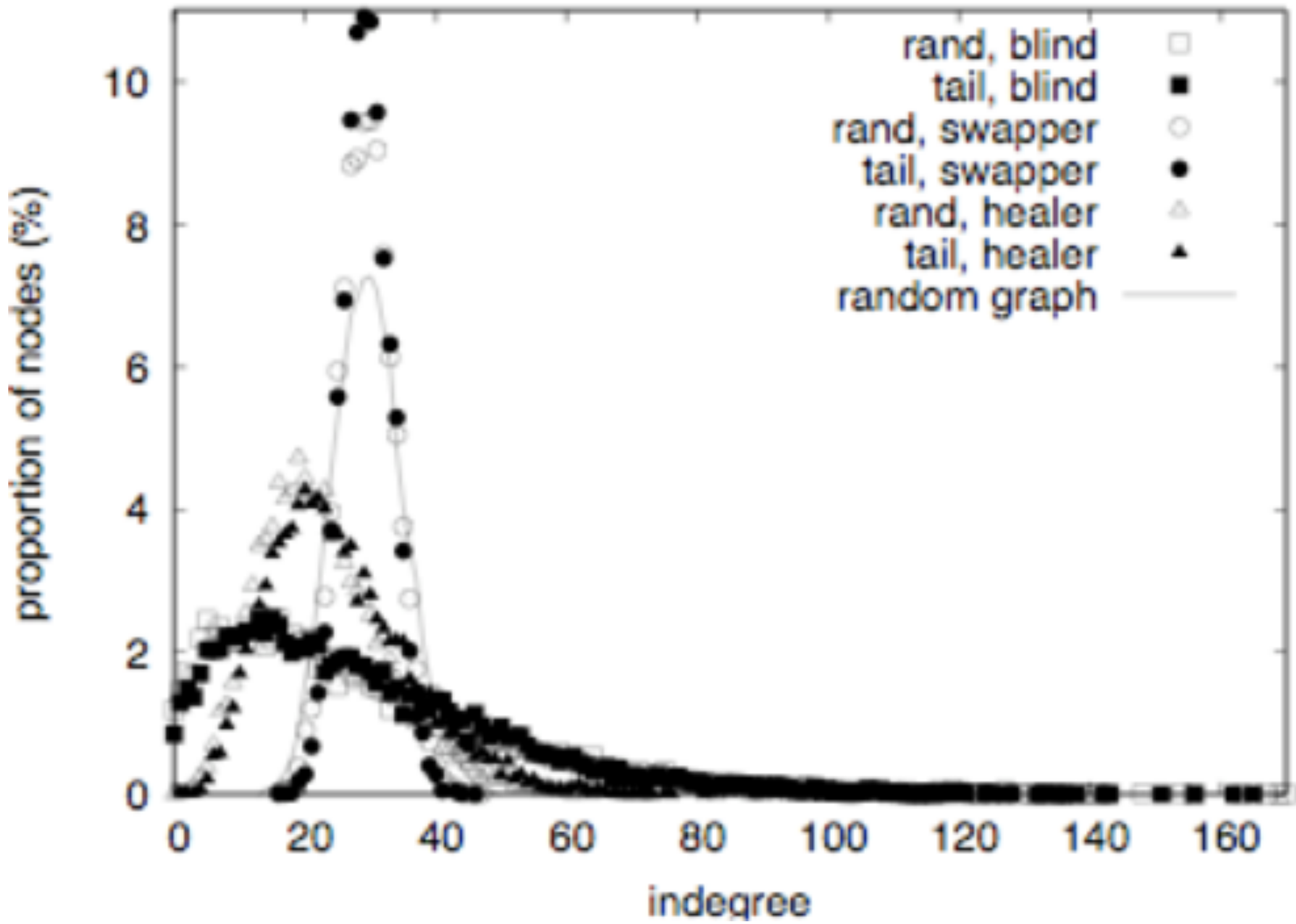
Degree distribution growing scenario



Max degree=contact node

Focus on pushpull protocols

Degree distribution



Degree distribution

Convergence

- Even in growing scenario
- Shuffler and healer result in lower standard deviation for opposite reasons

Shuffler

- Controlled degree distribution
- New links to a node are created only when the node itself injects its own fresh node descriptor during communication.

Healer

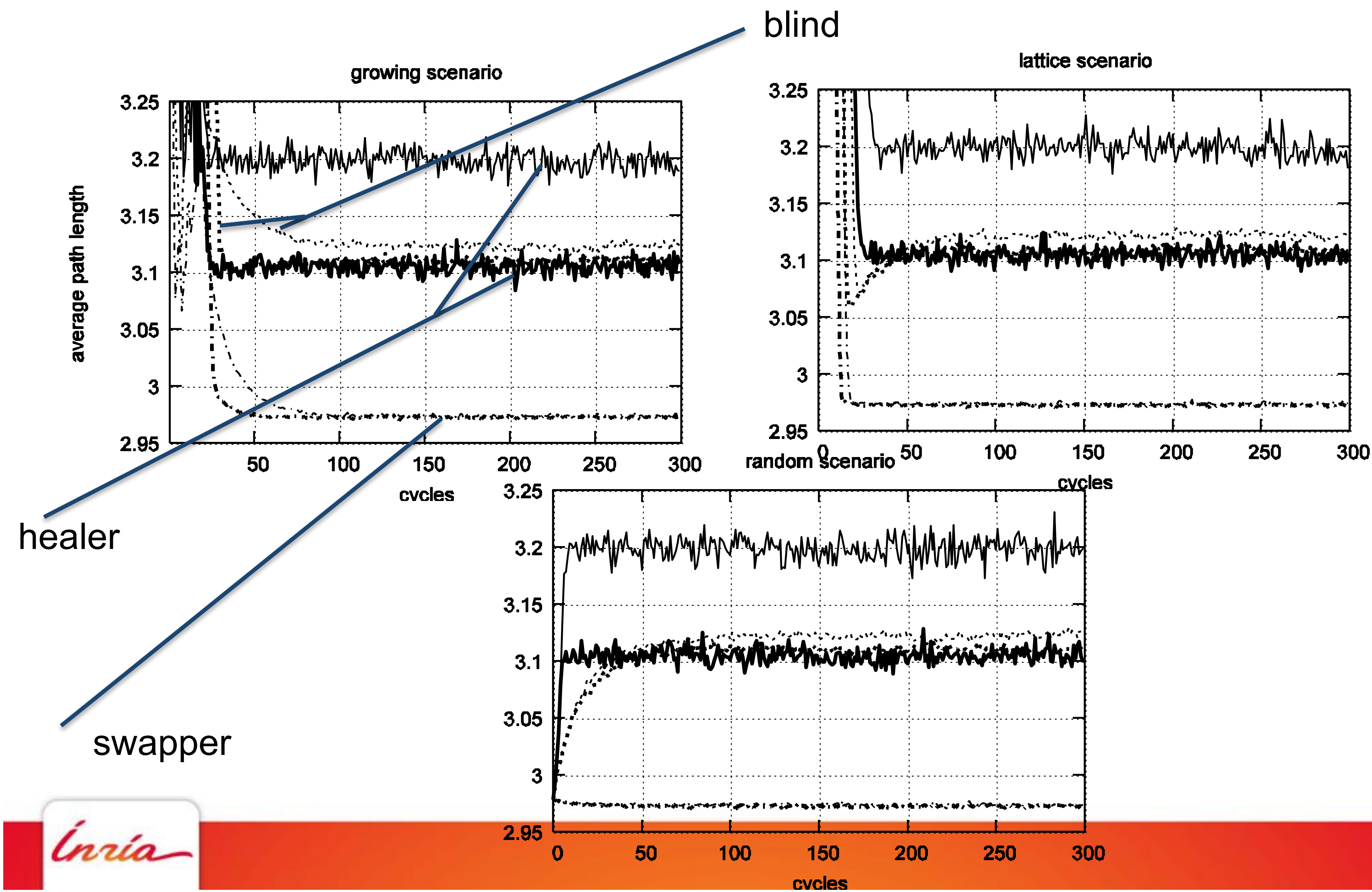
- Short life time of links
- When a node injects a new descriptor about itself, this descriptor is copied to other nodes for a few cycles.
- Later all copies are removed because they are pushed out by new links injected in the meantime

Average path length

Shortest path length between a and b

- minimal number of edges required to traverse in the graph to reach b from a
- Defines a lower bound on the time and costs of reaching a peer.
- Short average path length essential for scalability

Average path length



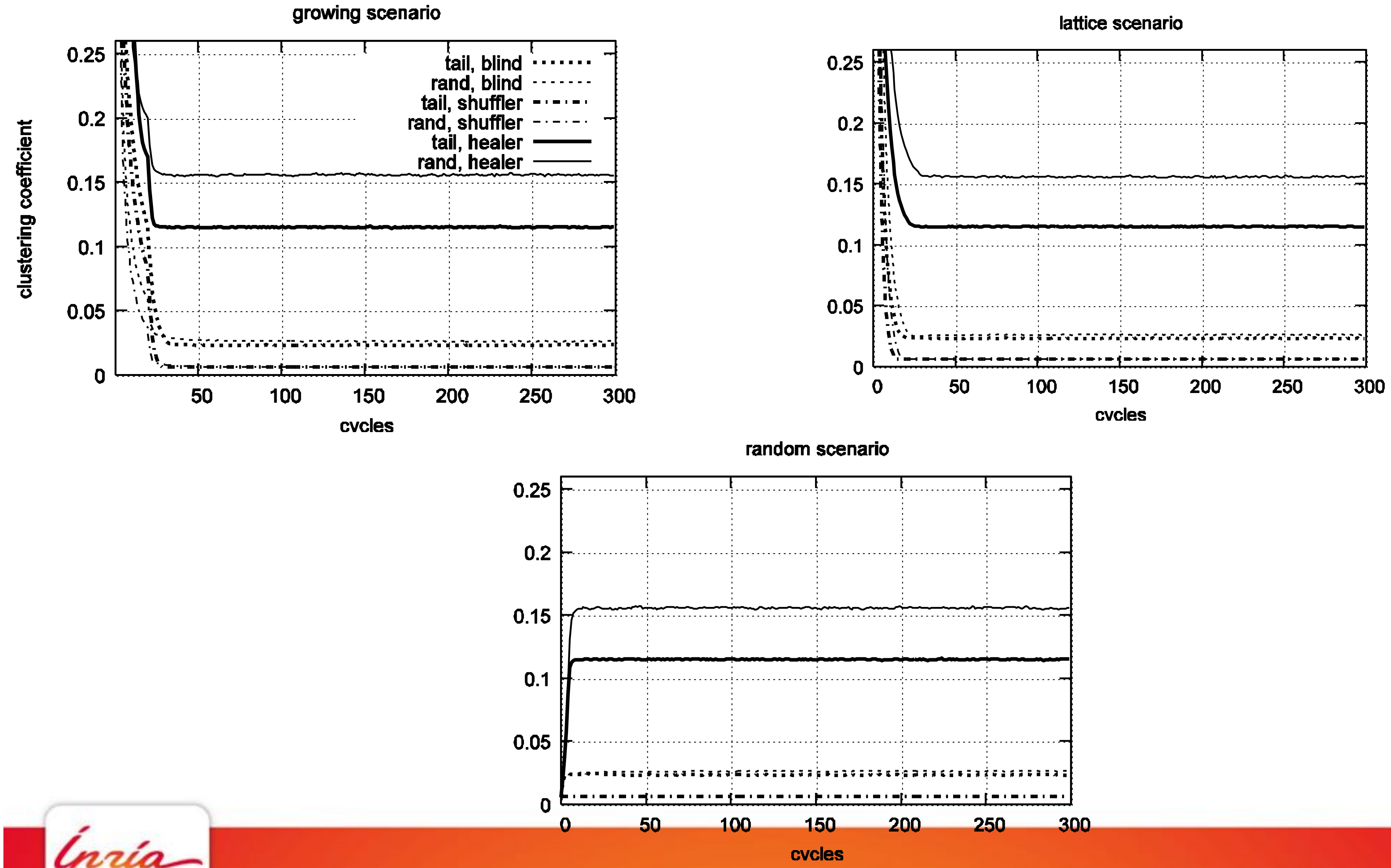
Clustering coefficent

Indicates to what extent neighbours of neighbours are neighbours

(1 for complete graph)

Important factor for information dissemination and partitioning risks

Clustering coefficient

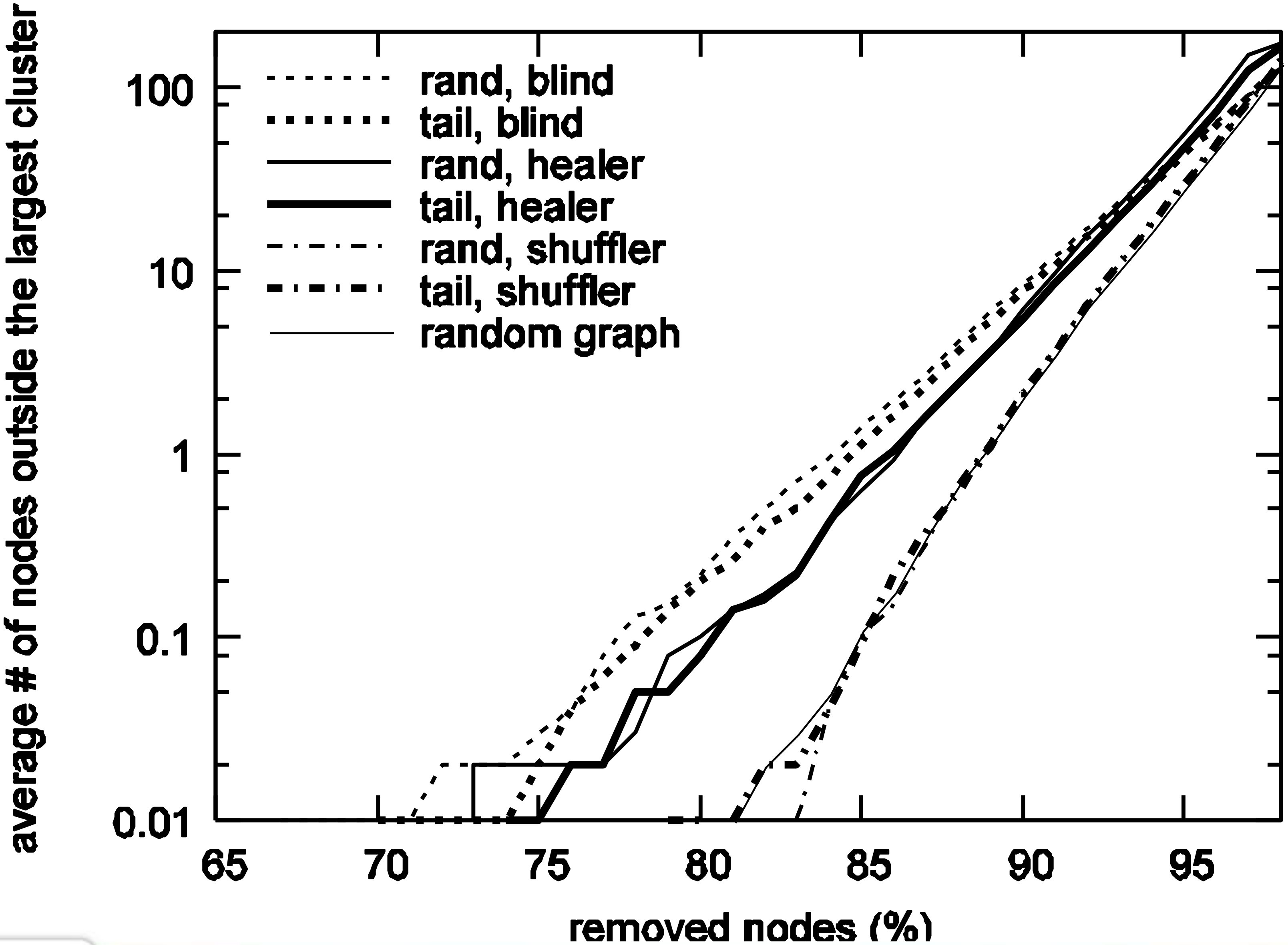


Clustering coefficient

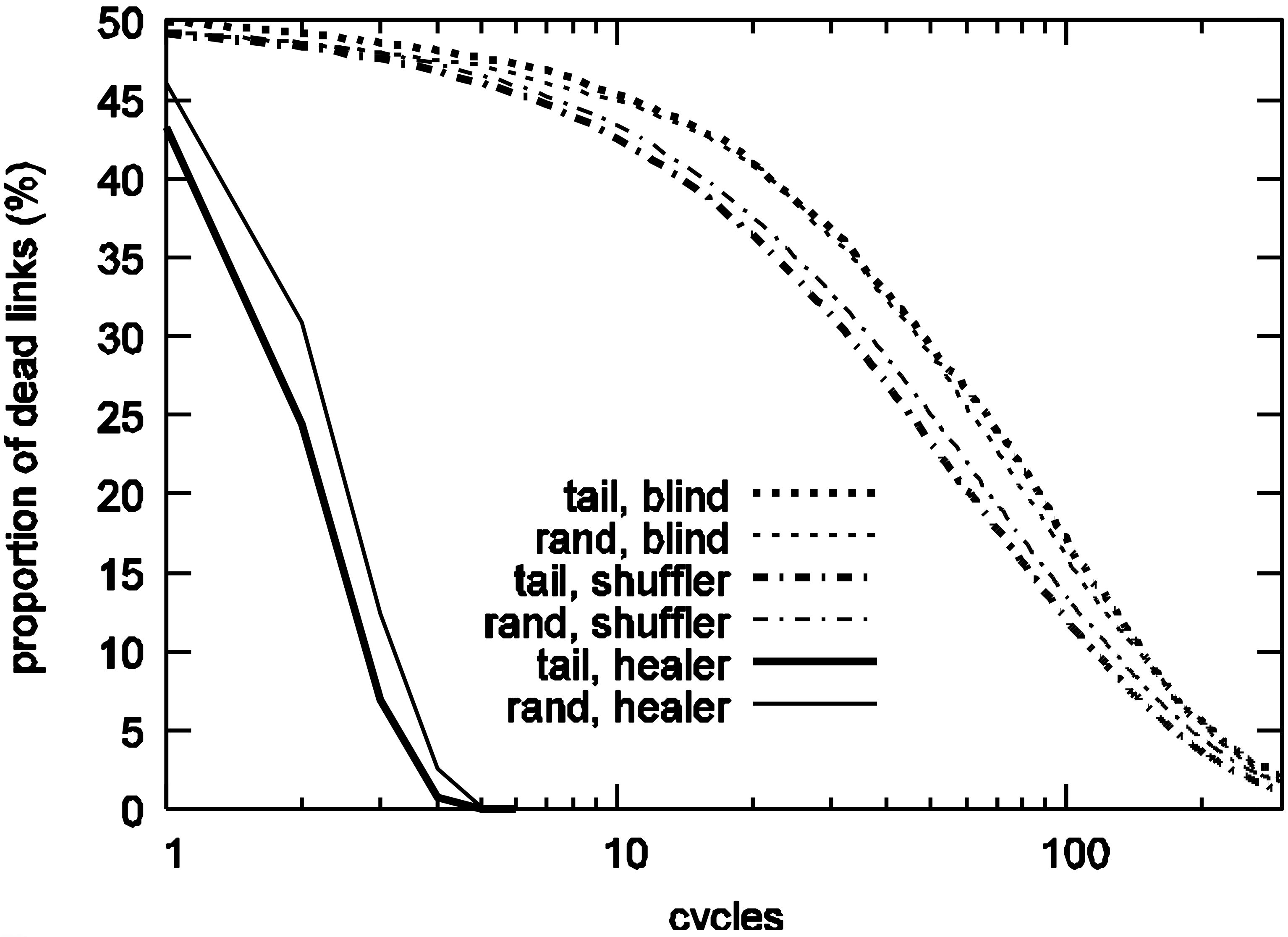
Results

- clustering coefficient also converges
- controlled mainly by H.
  - Large value of H result in significant clustering, where the deviation from the random graph is large.
    - large part of the views of any two communicating nodes overlap right after communication (freshest entries).
  - Large values of S, clustering is close to random

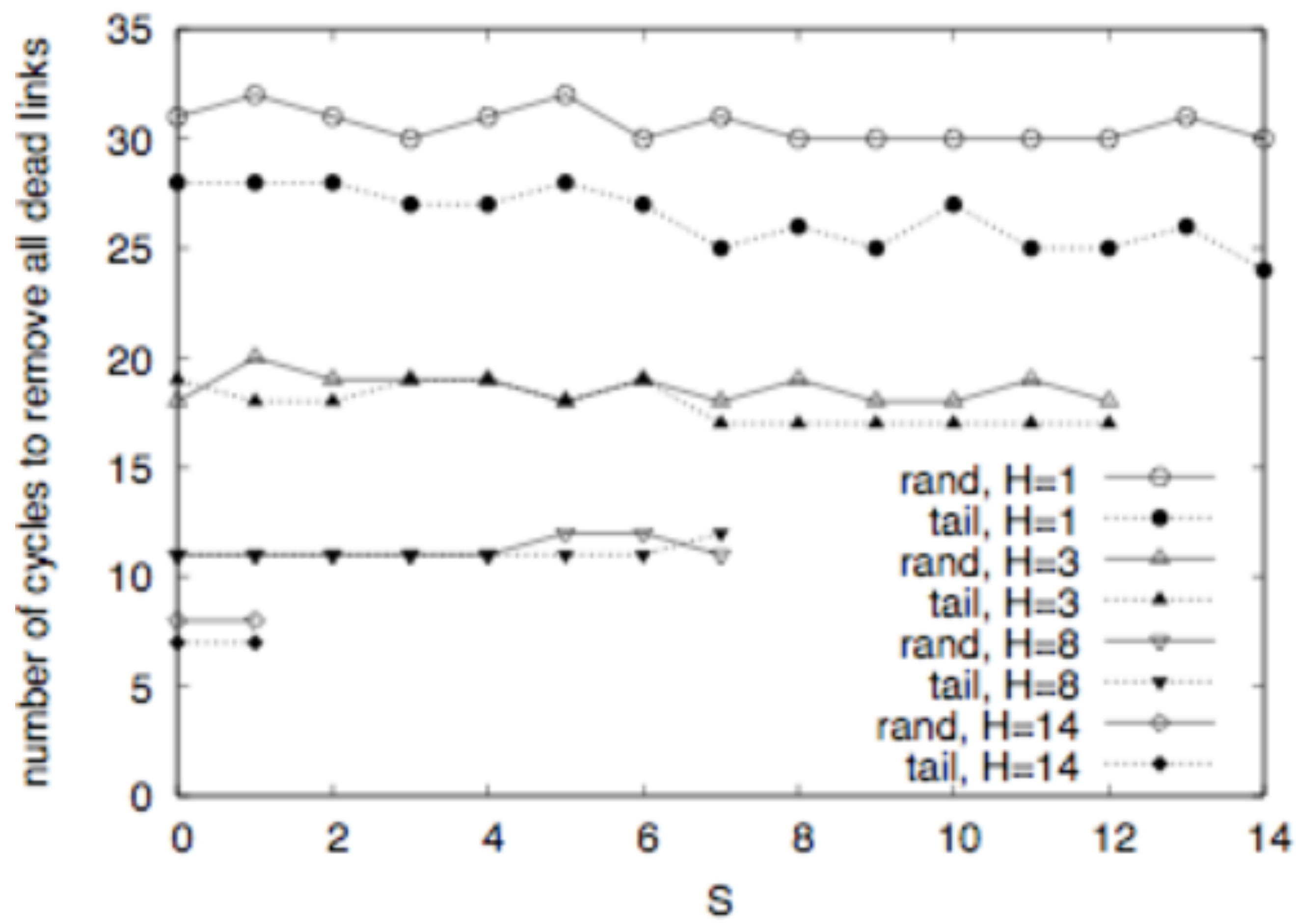
Catastrophic failures



Self-healing with 50% failures



Self-healing with 50% failures



Peer sampling service: Summary

- Experimental study
  - How random are the resulting graphs?
  - What properties may affect the applications
- Global randomness
  - Best configuration is the shuffler irrespective of the peer selection
- Load balancing
  - Blind performs poorly
  - Best configuration is shuffler while healer performs well
- Fault-tolerance
  - Most important parameter is H: the higher the better
  - Shuffler is slow in removing dead links

Today

- Gossip Basics
- Overlay Maintenance
  - Random peer sampling
  - Clustering

Structuring the network

- T-Man[Jelasity&Babaoglu, 2004]
- Peers optimize their view using the view of their close neighbours
- Ranking function

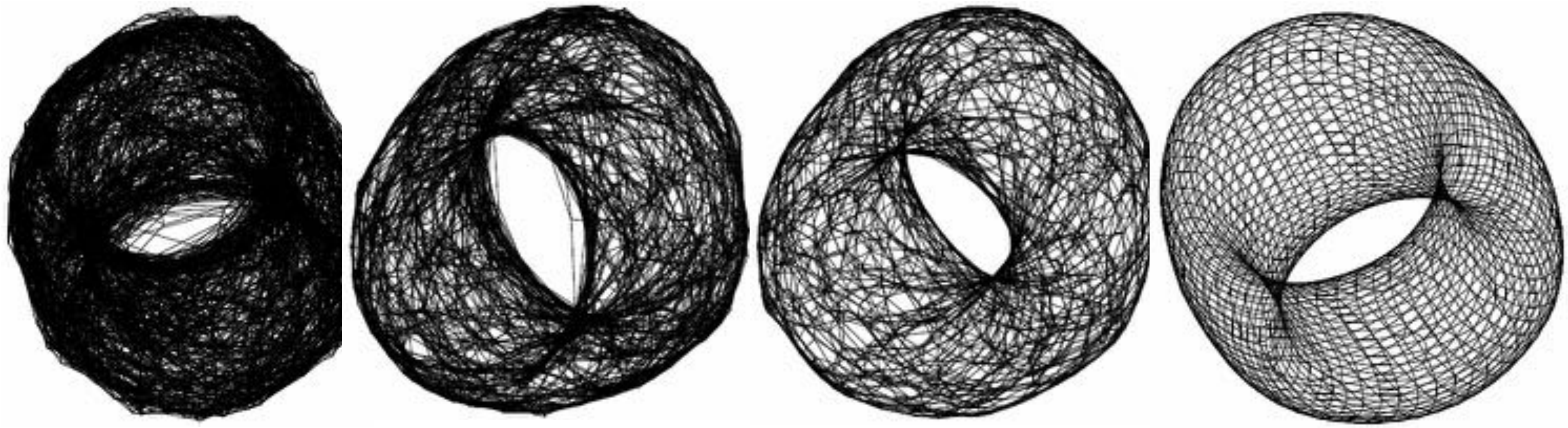
$$R(x, \{y_1, \ldots, y_m\}) \text{ ranks } y_j \text{ strictly lower than } y_i \text{ if } y_i \text{ precedes strictly } y_j \text{ in all possible rankings}$$

- Peer selection
  - Rank nodes in the view according to *R*
  - Returns a random sample from the first half
- Data exchange
  - Rank the elements in the (view+buffer) according to *R*
  - Returns the first *c* elements
- Data processing
  - Keep the c closest

Gossip-based topology management

- Line: d(a,b) =|a-b|
- Ring: interval[0,N], d(a,b)=min(N-|a-b|,|a-b|)
- Mesh and torus: d=Manhattan distance
- Sorting problems: any other application dependent metric

T-man: torus



Cycle 3 Cycle 5 Cycle 8 Cycle 15

T-man wrap up

- Generate a large number of structured topologies
- Exponential convergence (logarithmic in the number of nodes)
- Irrespective of the initial topology
- Exact structure

Clustering similar peers

- Vicinity: Introducing application-dependent proximity metric [VvS, EuroPar 2005]
- Two-layered approach
  - Biased gossip reflecting some application semantic
  - Unbiased peer sampling service

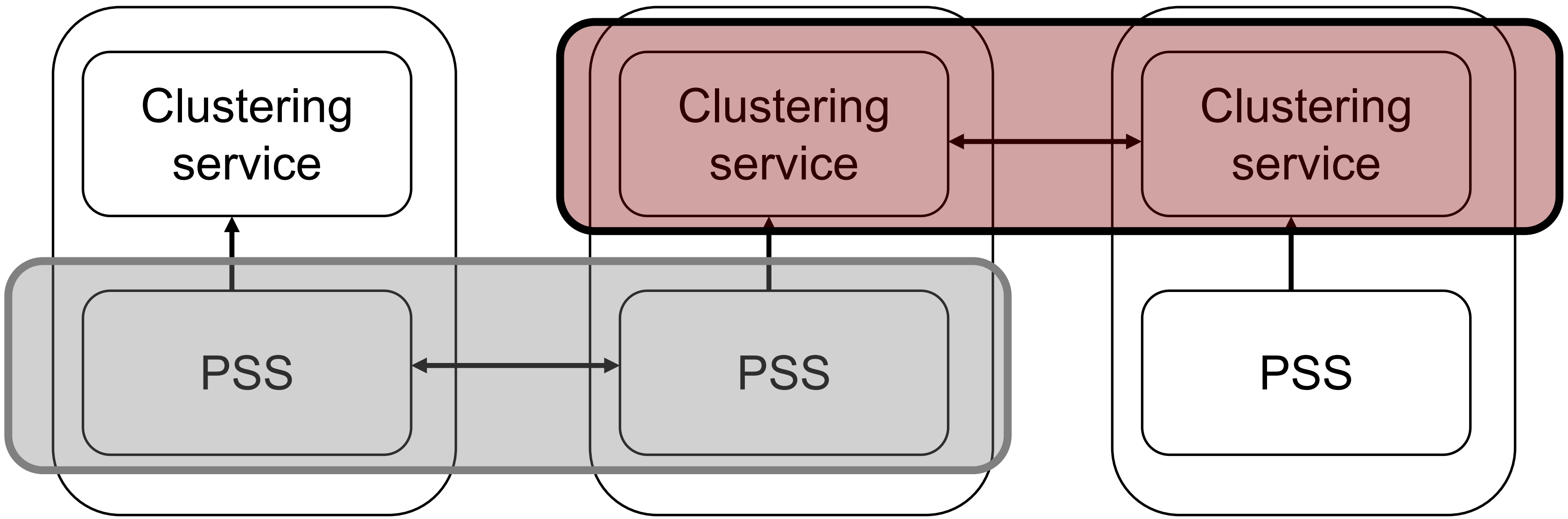
System model

- Semantic view of *l* semantic neighbours
- Semantic proximity function *S(P,Q).*
  - The higher the value of *S(P,Q),* the "closer" the nodes.
  - The objective is to fill P's semantic view to optimize

$$\sum_{i=1}^{l} S(P, Q_i)$$

Gossiping framework

- Target selection
  - Close peers
  - All nodes are examined: create a “small-world” like structure so that new nodes are discovered.



Gossip parameter setting

- Clustering protocol
  - Peer selection
    tail "oldest timestamp"
  - Data exchange
    aggressively biased,
    select the *g* items the closest from semantic and random views
  - Data processing
    select the *l* closest peers (buffer, semantic and random views)
- Peer sampling service
  - ….