Distributed Systems

Topology bootstrap

Alberto Montresor
Università di Trento

(joint work with Mark Jelasity and Ozalp Babaoglu)

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Introduction

- **Current P2P applications:**
  - One protocol, one overlay network, one application
  - Massive dynamism is under-emphasized
- **Novel application scenarios**
  - *On demand services*
    - A structured overlay could be built on resources dynamically assigned on demand, maybe temporarily, maybe for payment
  - Flexible resource management
    - We may want to be able to change resource assignments based on QoS requirements

© Alberto Montresor

Overview

- **What we need to realize those scenarios?**
  - Dynamically allocate nodes to multiple applications (*slicing*)
  - Start overlays from scratch (*bootstrap*)
  - Manage “federations” of overlays (*merging / splitting*)
  - Deal with massive dynamism
    - Catastrophic failures
    - Variations in QoS requirements (flash crowds)
  - Monitor applications (*aggregation*)
  - Stop applications when they are done

© Alberto Montresor

Motivation

- **System model**
  - An huge collection of networked nodes (resource pool)
  - Potentially managed by a single organization that deploys massive services and application on such networks
- **Examples**
  - ISPs that place smart phones / modems / set-top boxes at their customers' homes
  - BT Telecom, France Telecom

© Alberto Montresor

Architecture

Applications

Other middleware services (DHTs, indexing, publish-subscribe, etc.)

Slicing Service	Topology Bootstrap	Monitoring Service	Broadcast

Peer sampling service

© Alberto Montresor

Peer sampling

- **Provides random samples from the set of participating nodes**
- **Based on an unstructured (random) overlay**
  - Maintained via a gossip-based protocol that continuously exchanges random overlay links among the nodes
  - Cheap, simple and minimal functionality
- **Newscast**
  - Random exchange, keep fresh links
  - Auto-cleaning: removes old information (crashed nodes) automatically

© Alberto Montresor

The problem

- **Topology bootstrap**
  - Informal definition: *building a topology from the ground up as quickly and efficiently as possible*
- **Do not confuse with *node bootstrap***
  - Placing a single node in the right place in the topology
- **Do not confuse with topology maintenance**
  - Stabilization (routing table cleanup)
  - Can be used later to optimize the topology

© Alberto Montresor

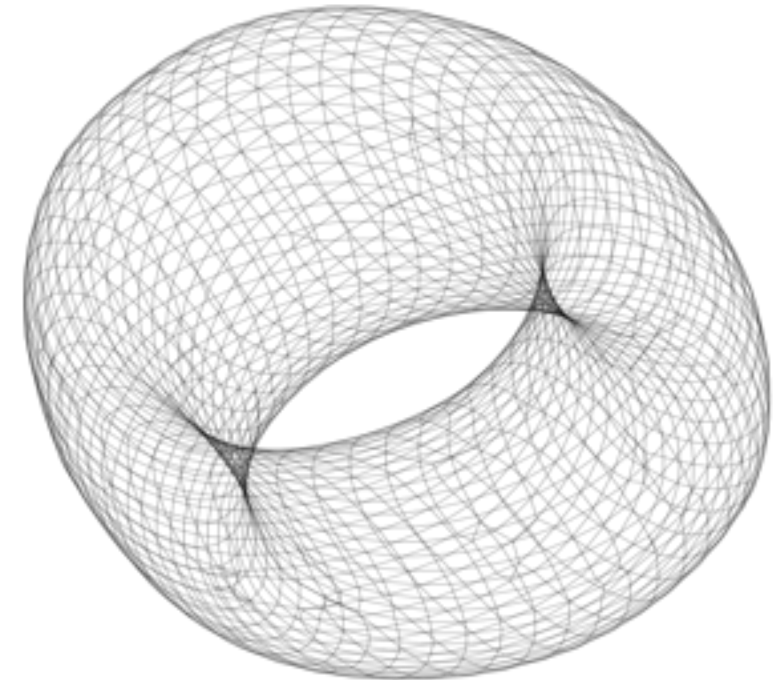
State-of-the-art

- **Current DHT protocols do not support bootstrapping:**
  - They assume an already formed network
  - Even supposing the network exists, they work "*unless a tremendous number of nodes joins the system*" [Chord]
  - We could envision a "join orchestration approach", but it would require a linear time

© Alberto Montresor

The T-Man algorithm

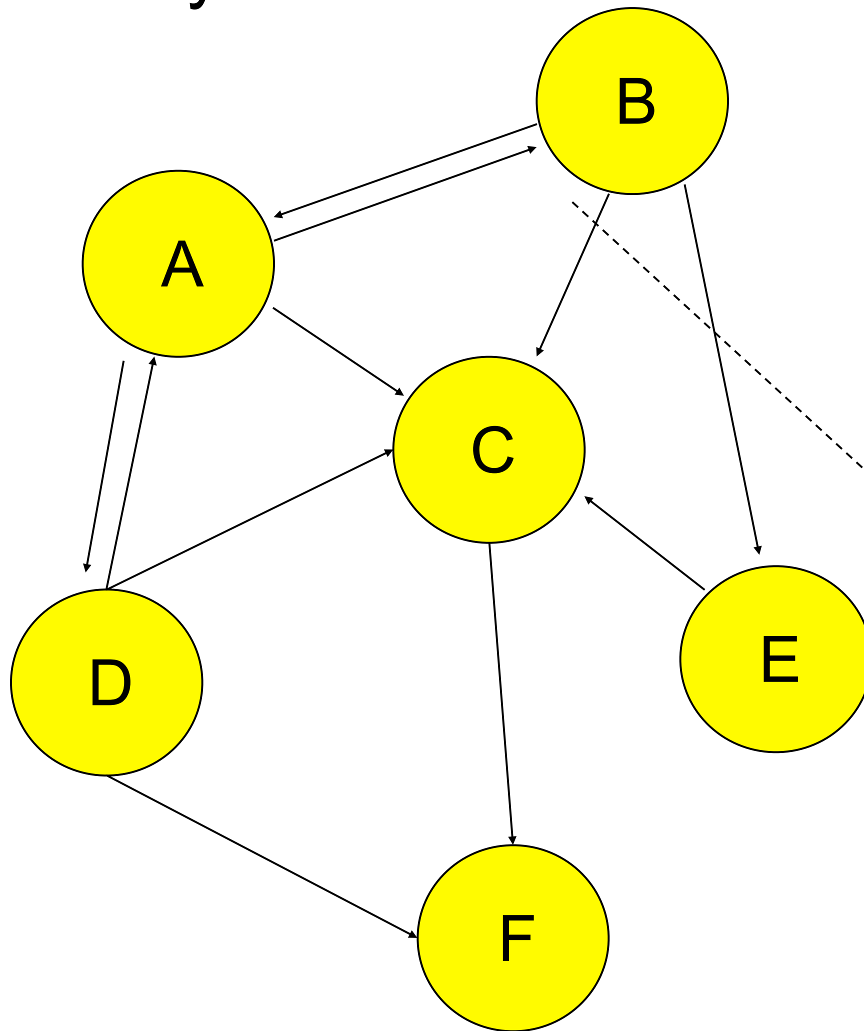
- **T-man is a generic protocol for topology formation**
  - Topologies are expressed through *ranking functions*: "which are my preferred neighbors?"
- **Examples**
  - Rings, tori, trees, DHTs, etc.
  - Distributed sorting
  - Semantic proximity for file-sharing
  - Latency for proximity selection
  - Etc.



© Alberto Montresor

System abstraction

Overlay network

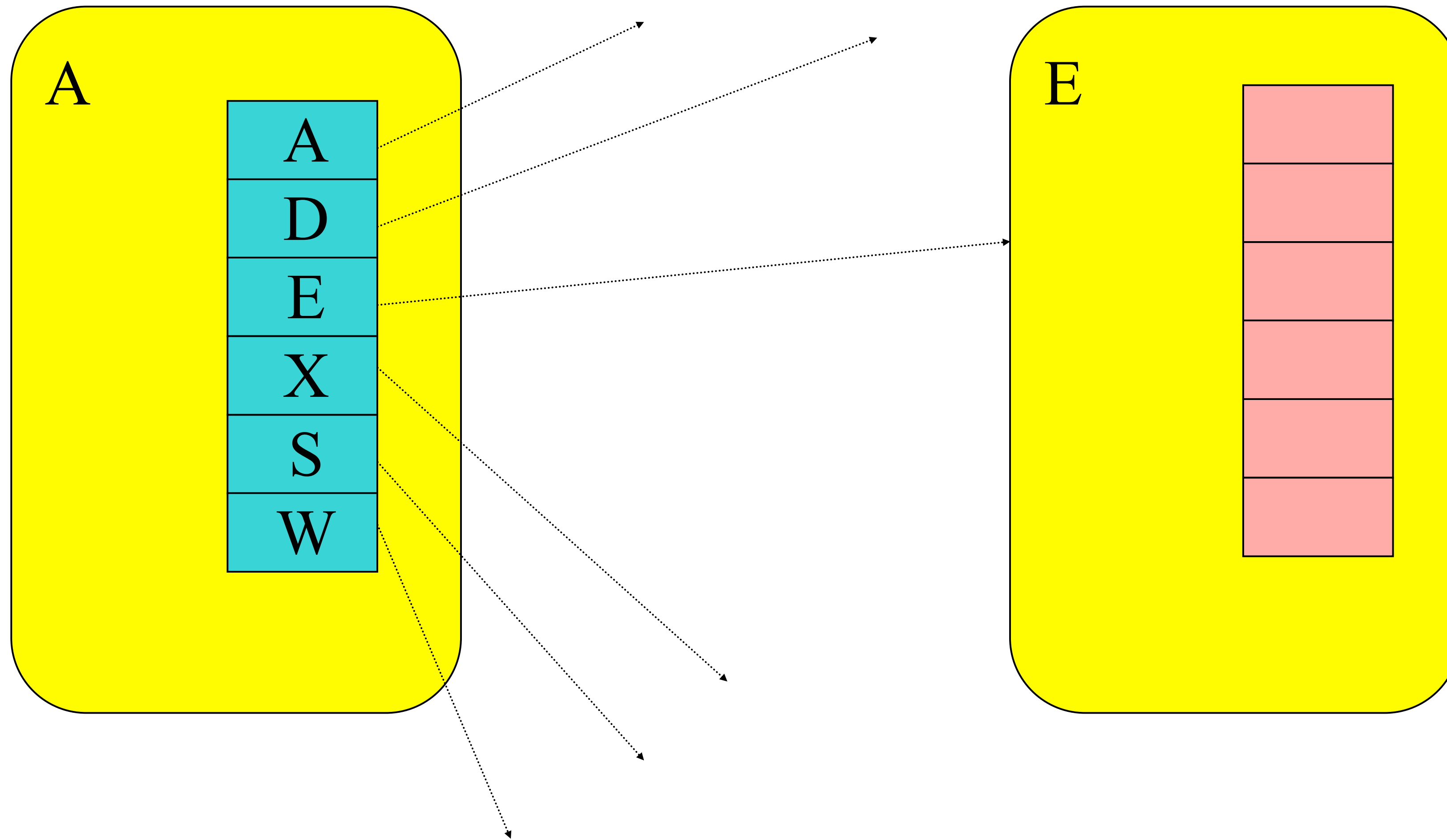


View of B:

Descriptor of A
Descriptor of C
Descriptor of E

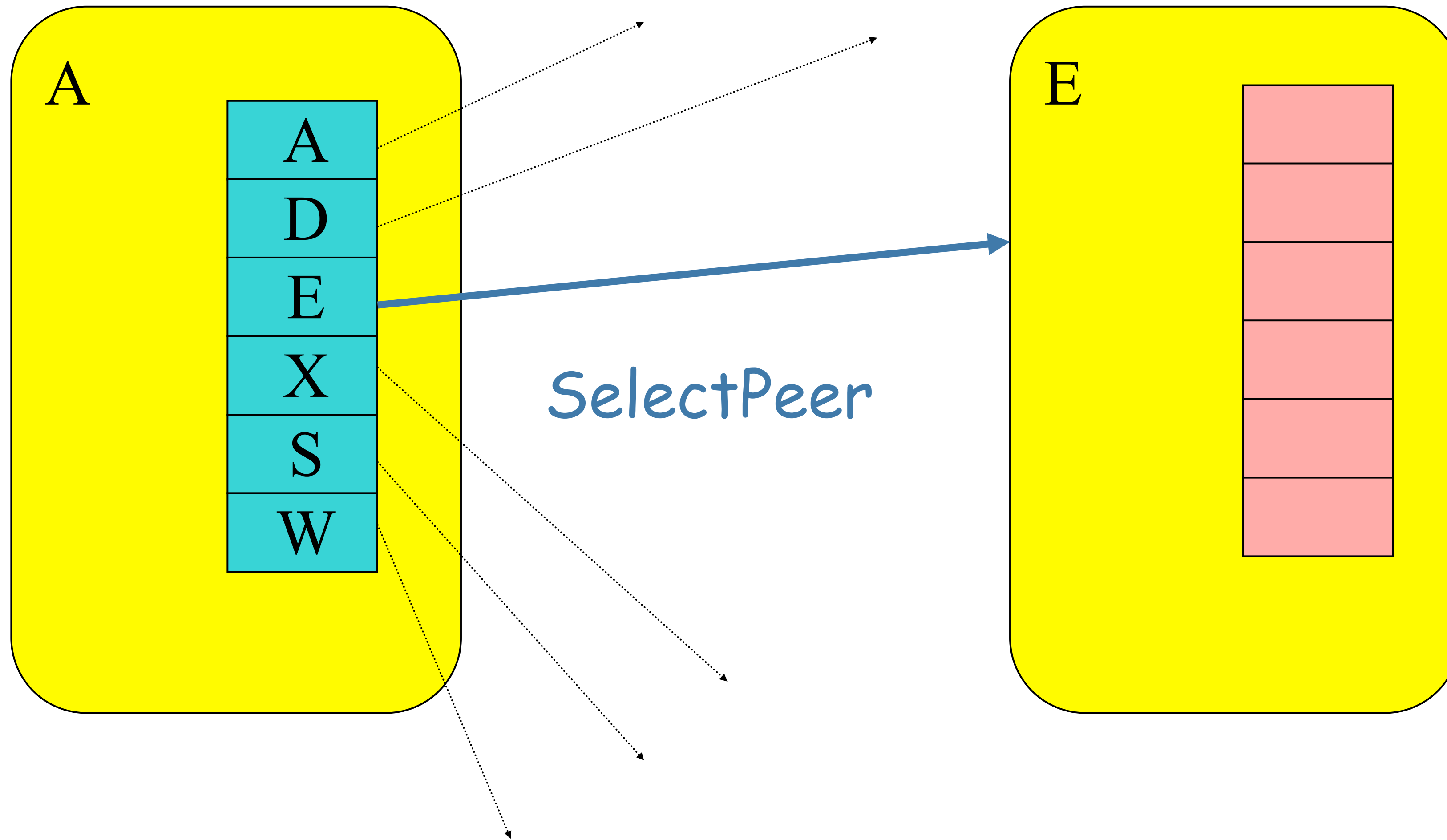
© Alberto Montresor

Gossip protocol for topology management



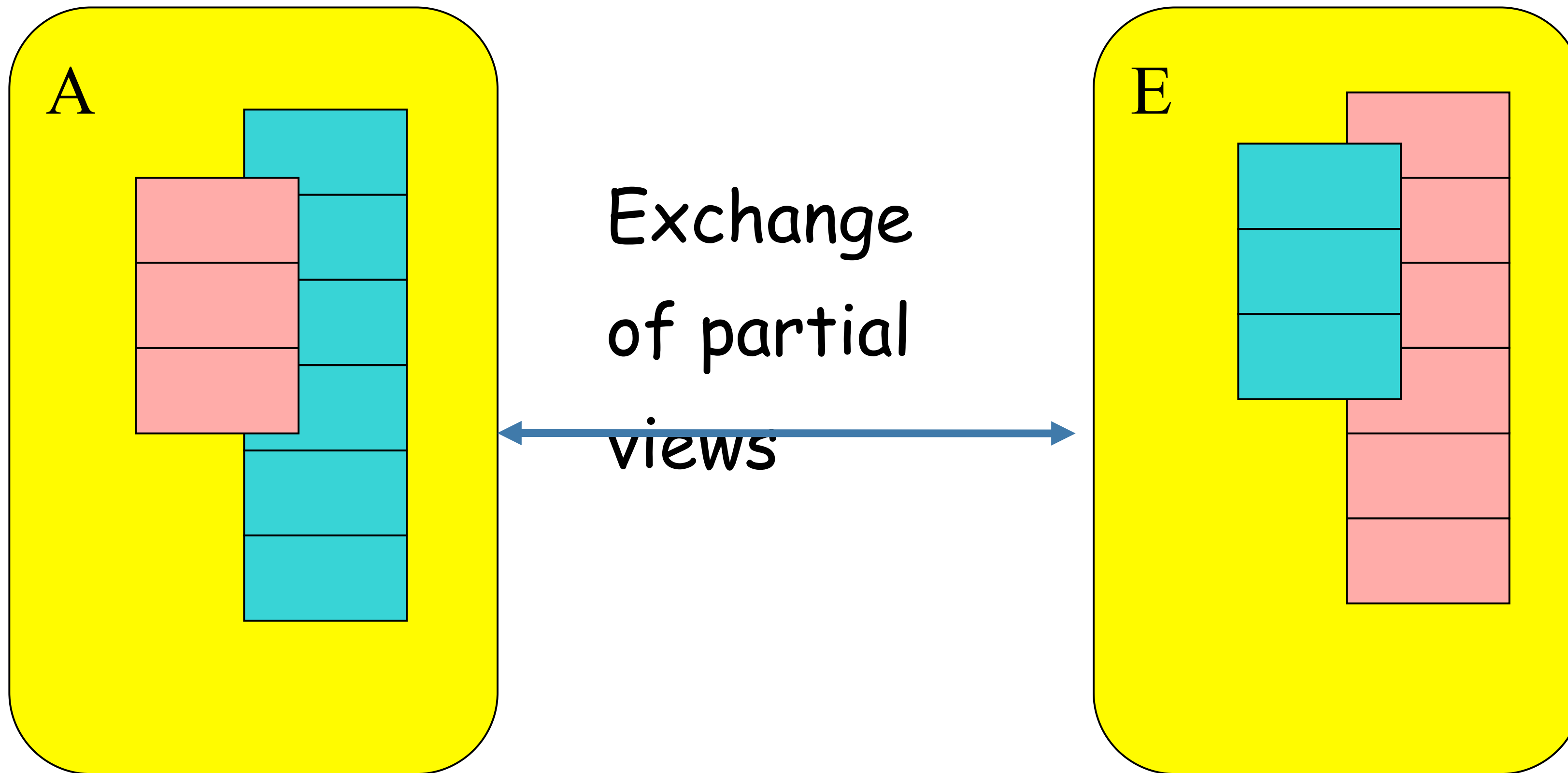
© Alberto Montresor

Gossip protocol for topology management



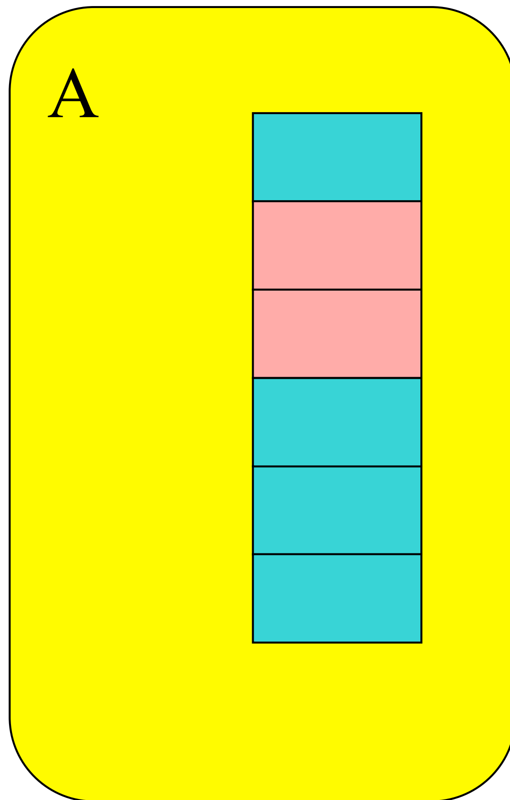
© Alberto Montresor

Gossip protocol for topology management



© Alberto Montresor

Gossip protocol for topology management



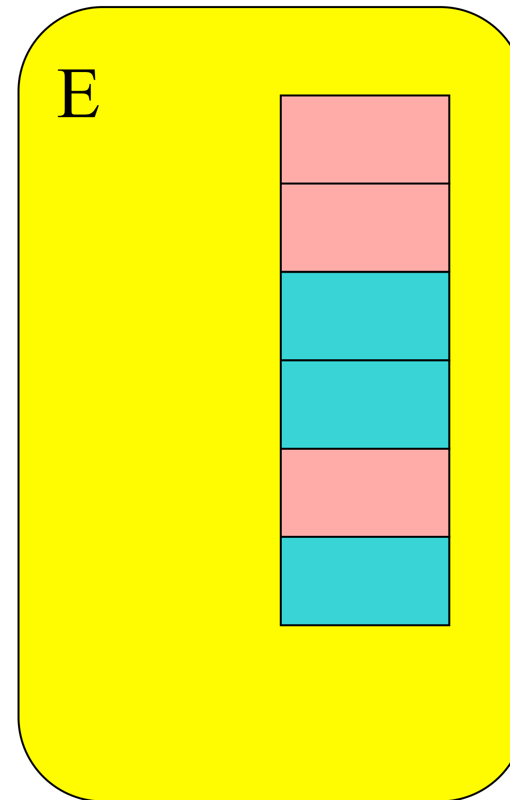
Both sides

apply update

thereby

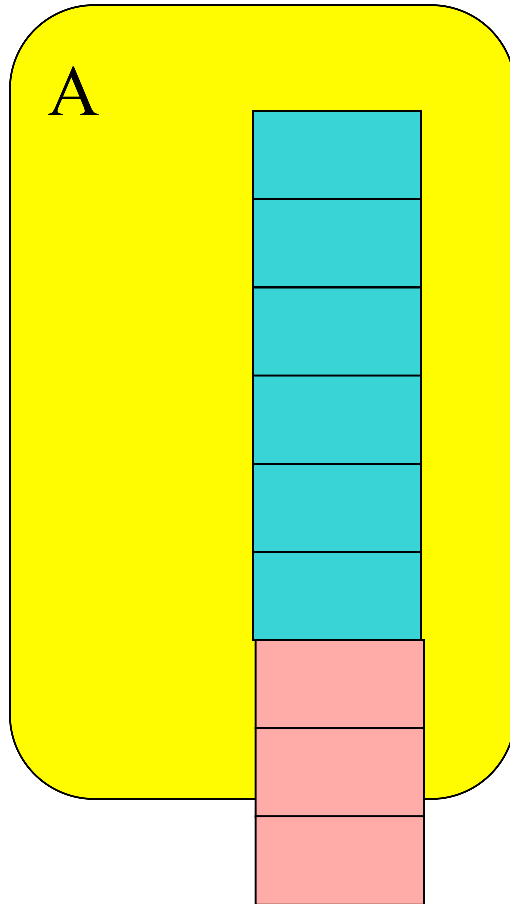
redefining

topology



© Alberto Montresor

Gossip protocol for topology management



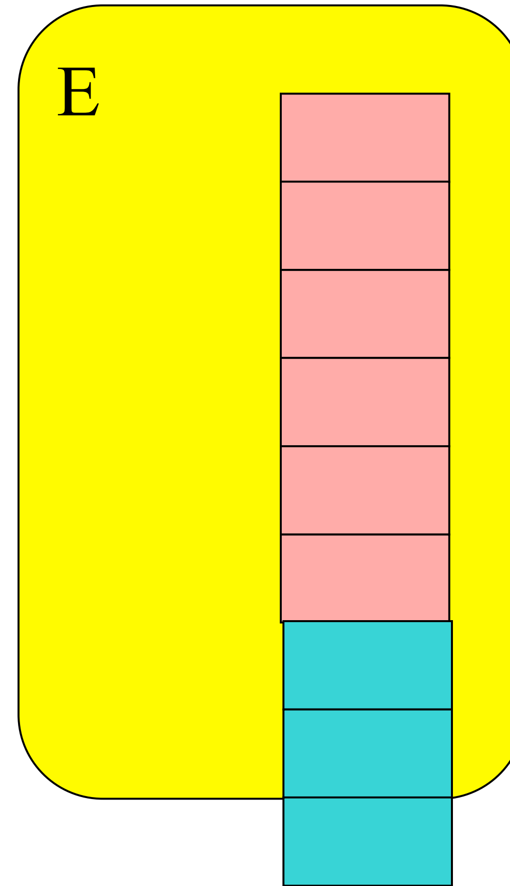
Both sides

apply update

thereby

redefining

topology



© Alberto Montresor

The T-Man algorithm

// *view* is a collection of neighbors

Init: *view* = *rnd.view* ∪ { (*myaddress*, *mydescriptor*) }

// active thread

// executed by p

do once every
δ time units

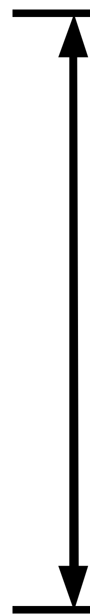
q = selectNeighbor(view)

msg_p = extract(*view*, q)

send msg_p **to** q

receive msg_q **from** q

view = merge(*view*, msg_q)



A "cycle" of length δ

// passive thread

// executed by p

do forever

receive msg_q from *

msg_p = extract(*view*, q)

send msg_p **to** q

view = merge(*view*, msg_q)

© Alberto Montresor

Ranking function

- **Node descriptors contain attributes of the nodes**
  - *A number in a sorting application*
  - The id of a node in a DHT
  - A semantic description of the node
- **Given a node x, we may define a *ranking function* over node descriptors to identify the preferred neighbors of x**
  - given a collection *U* of nodes, rank them in some order of preference
  - Alternatively: given a collection *U* of nodes, returns the *r* nodes preferred by *x*

© Alberto Montresor

Ranking Function

- **The ranking function may be based on a distance over a space**
  - Space: set of possible descriptor values
  - Distance: a metric *d(x,y)* over the space
  - The ranking function of node *x* is defined over the distance from node *x*
- **Given a collection U of nodes**
  - rank them in *increasing distance order*
  - alternatively: returns the *r* *closest* node based on the notion of distance

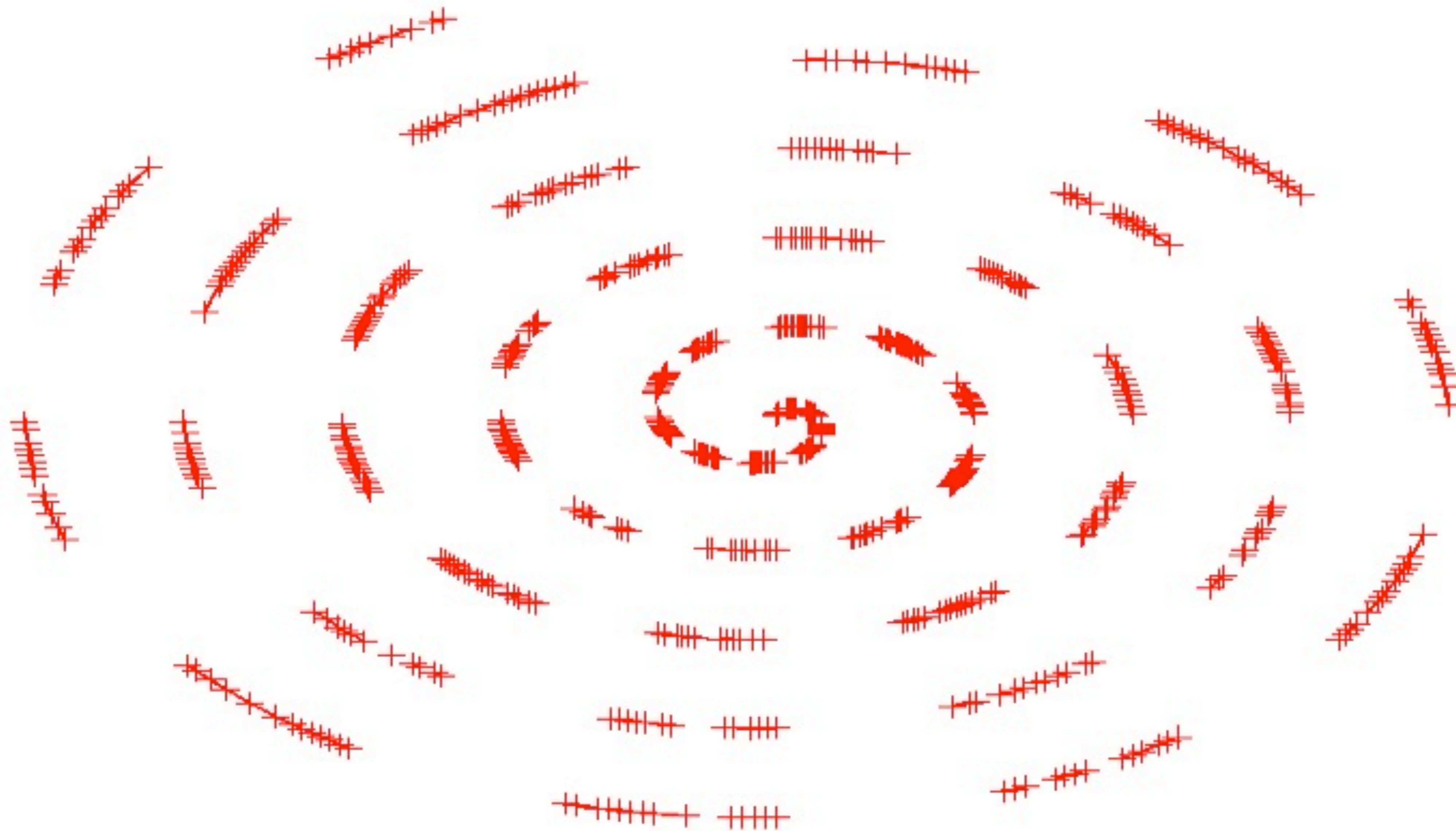
© Alberto Montresor

Distance functions

- **Example: Line or ring**
  - Space: [0, 1[
  - Distance over the line: $d(a,b) = | a-b |$
  - Distance over thr ring: $d(a,b) = min \{ | a-b | , 1-|a-b| \}$
- **Example: Grid or torus (Manhattan Distance)**
  - Space: [0, 1[· [0, 1[
  - Distance: $d(a,b) = | a_x - b_x | + | a_y - b_y |$

© Alberto Montresor

Example: Line



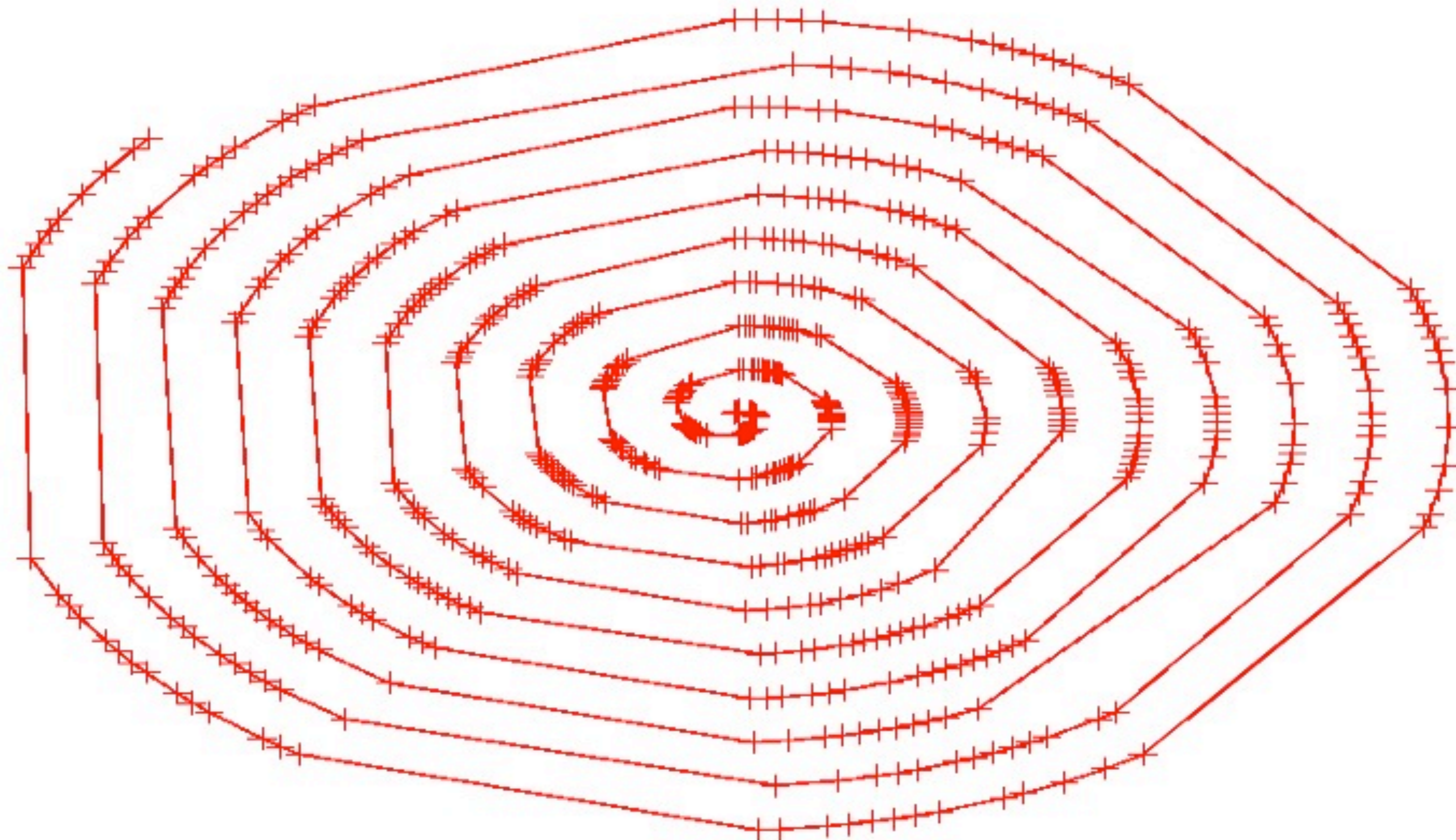
© Alberto Montresor

Sorted Line / Ring

- **Directional ranking function over the ring defined as follows:**
  - Distance function, line: $d(a,b)=|a-b|$
  - Distance function, ring: $d(a,b)=min(|a-b|, 1-|a-b|)$
- **Given a collection U of nodes and a node x, return**
  - the *r/2* nodes "smaller" than x that are closest to x
  - the *r/2* nodes "larger" than x that are closest to x

© Alberto Montresor

Sorted Line



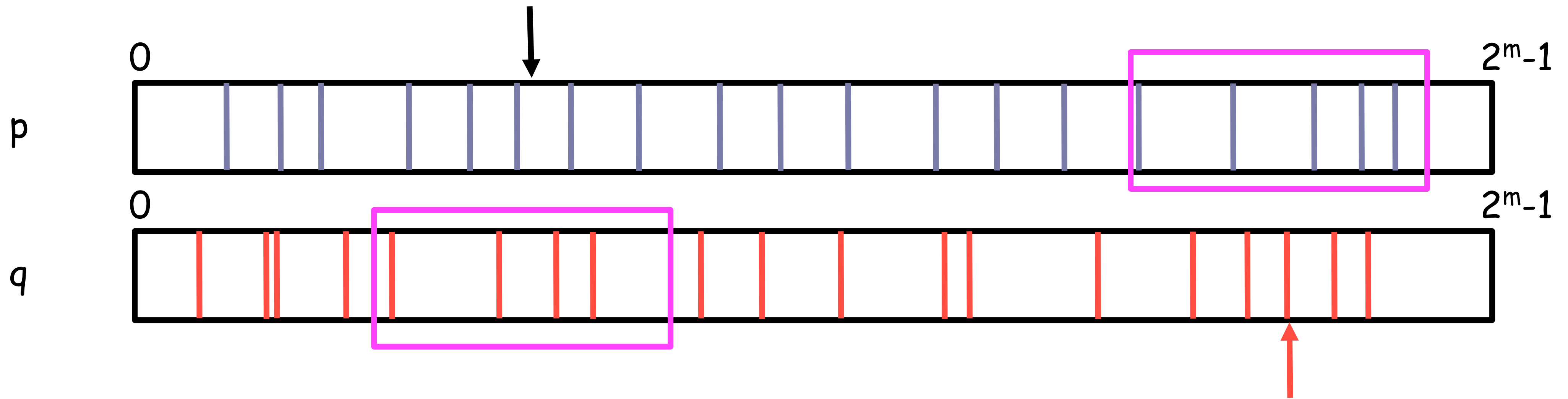
© Alberto Montresor

Sorted Ring

- **selectPeer():**
  - randomly select a peer ***q*** from the ***r*** nodes in my view that are *closest to **p** in terms of distance*
- **extract():**
  - send to ***q*** the ***r*** nodes in local view that are *closest to* **q**
  - ***q*** responds with the ***r*** nodes in its view that are *closest to* **p**
- **merge():**
  - both ***p*** and ***q*** *merge* the received nodes to their view

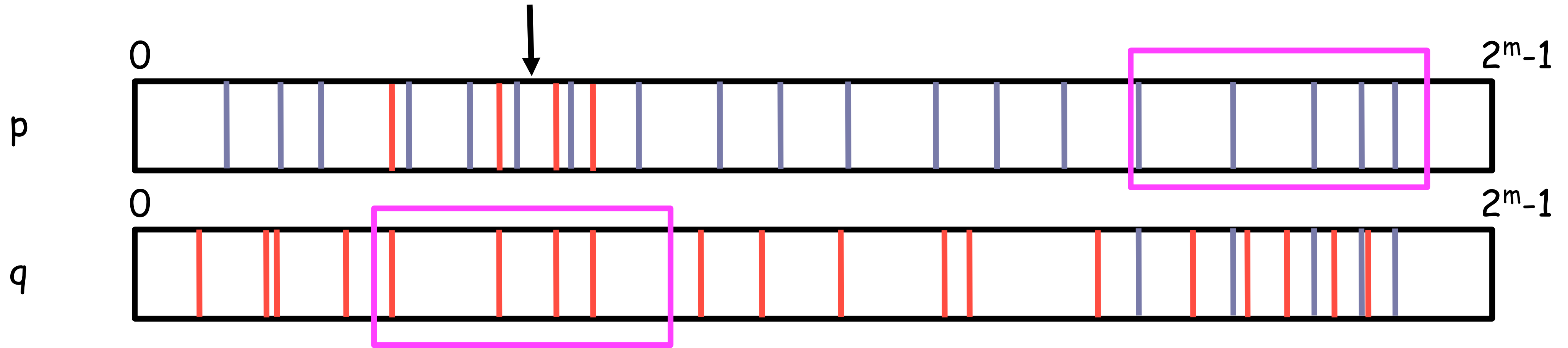
© Alberto Montresor

Sorted Ring



© Alberto Montresor

Sorted Ring

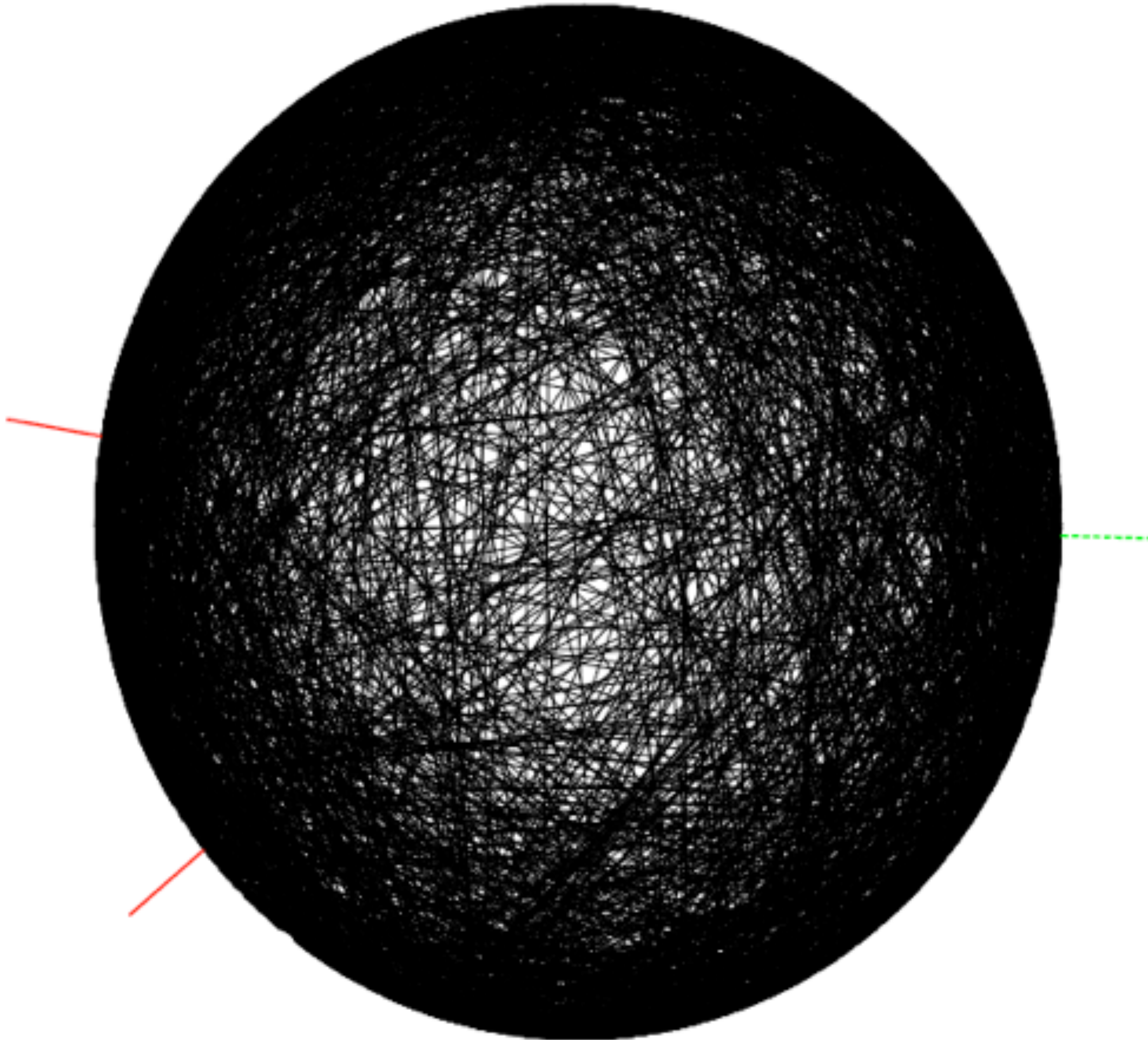


© Alberto Montresor

Sorted Ring

Nodes: 1000
Showing 1 successor,
1 predecessor

Cycles 00.000 Nodes executed in a cycle



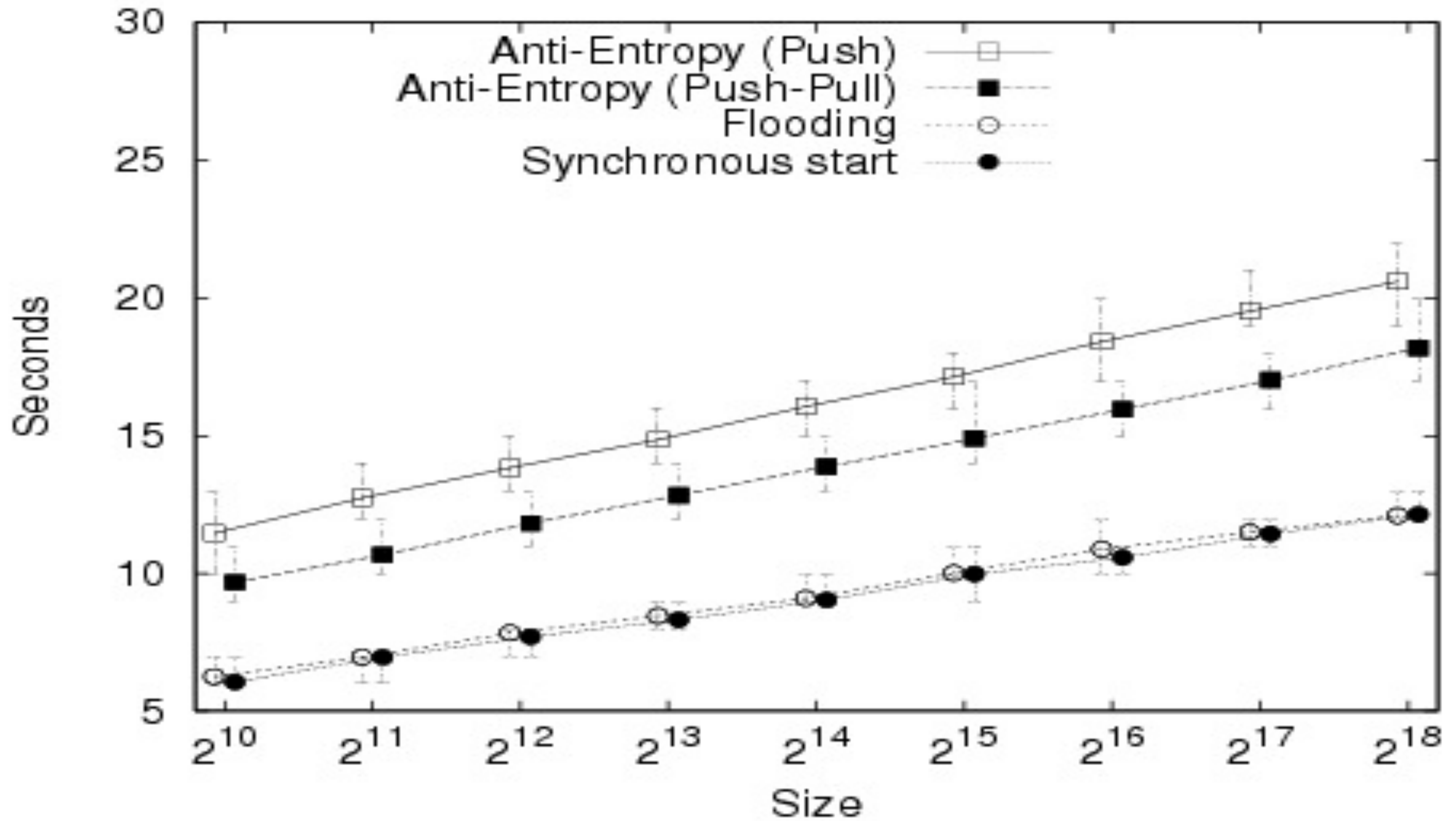
© Alberto Montresor

How to start and stop

- In the previous animation
  - Nodes starts simultaneously
  - Convergence is measured globally
- In reality
  - We must start the protocol at all nodes
    - Broadcasting, using the random topology obtained through the peer sampling services
  - We must stop the protocol
    - Local condition: when a node does not observe any view change for a predefined period, it stops
    - Idle: number of cycles without variations

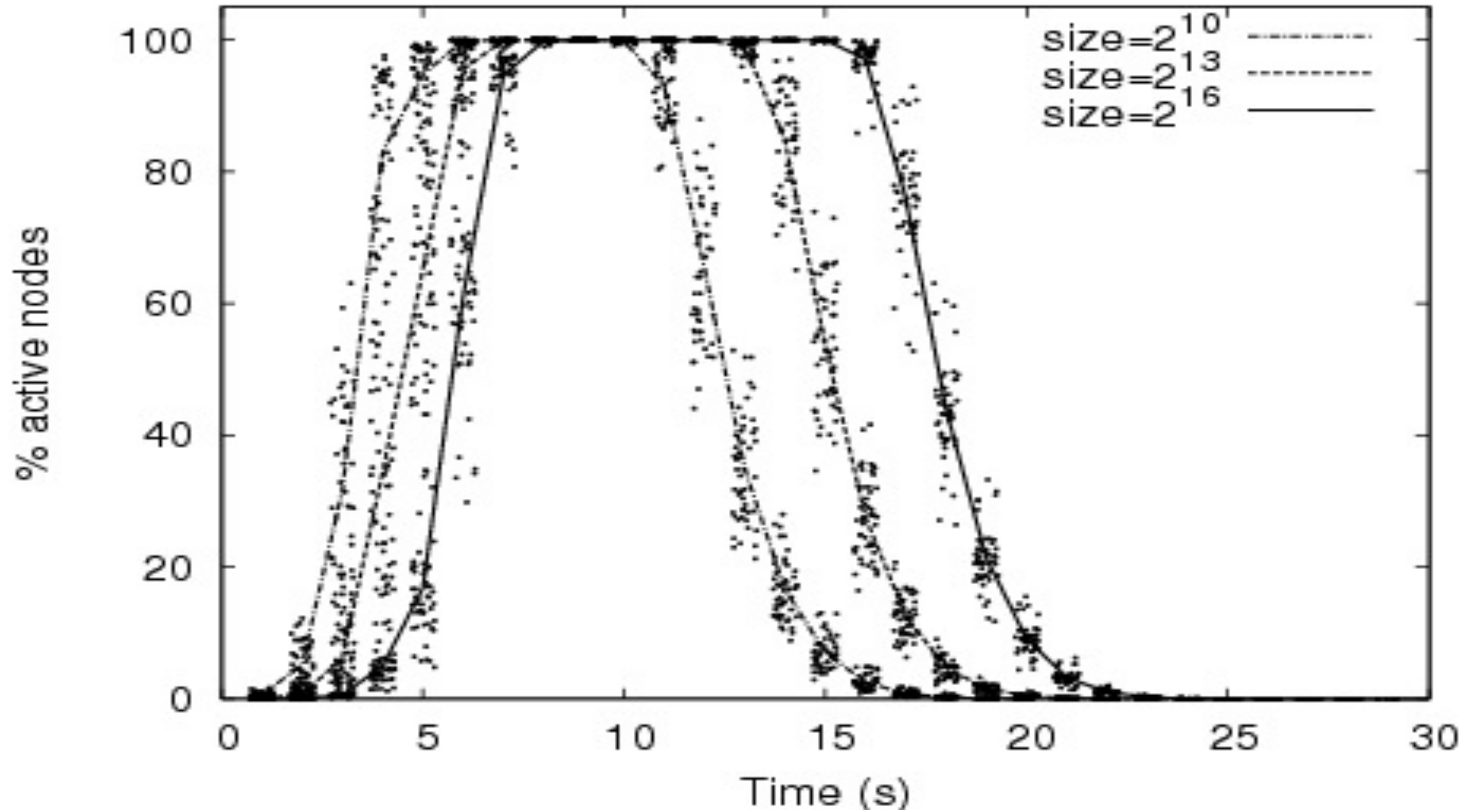
© Alberto Montresor

Scalability, with different starting protocols



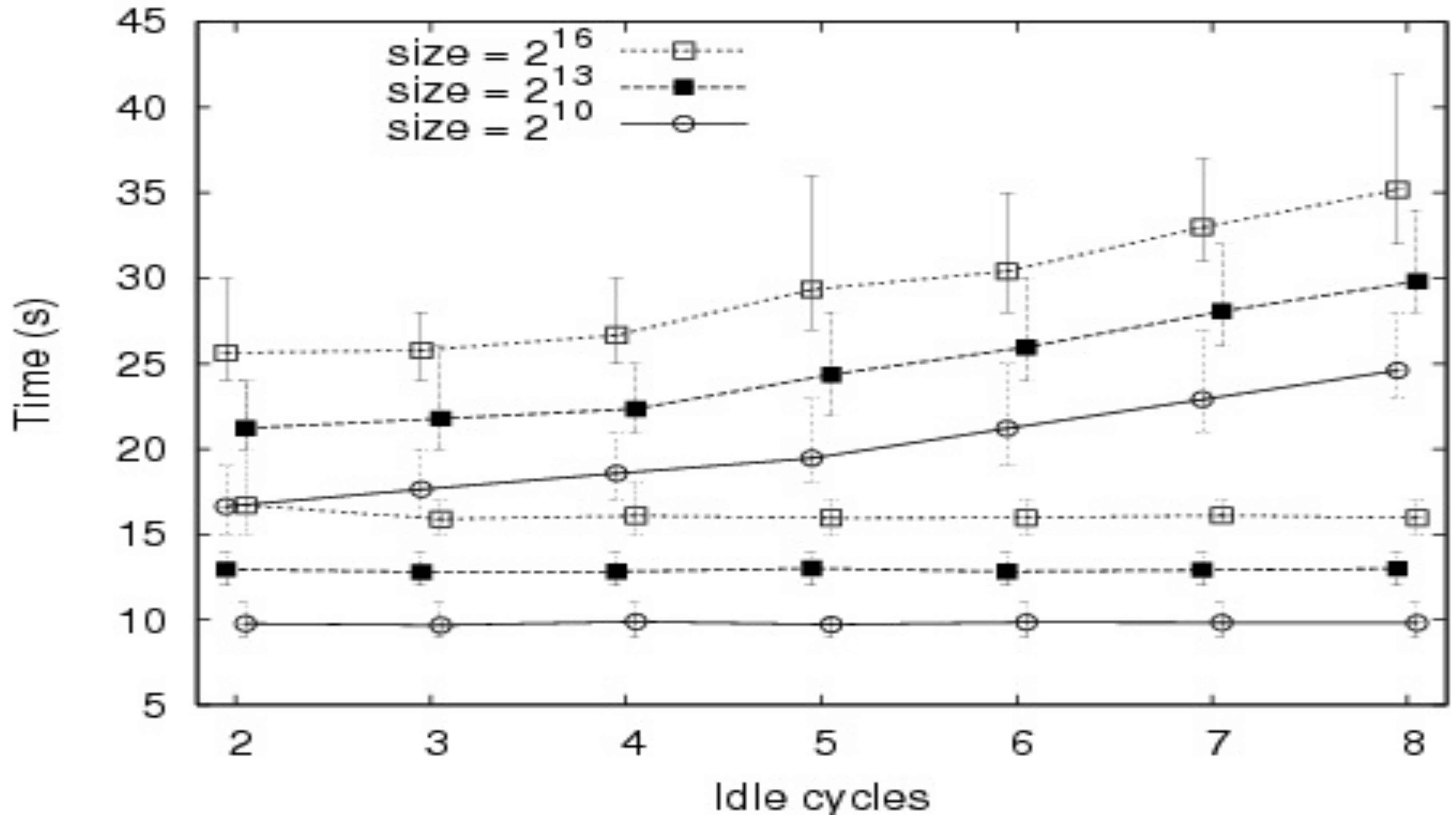
© Alberto Montresor

Bootstrap protocol: Starting and stopping



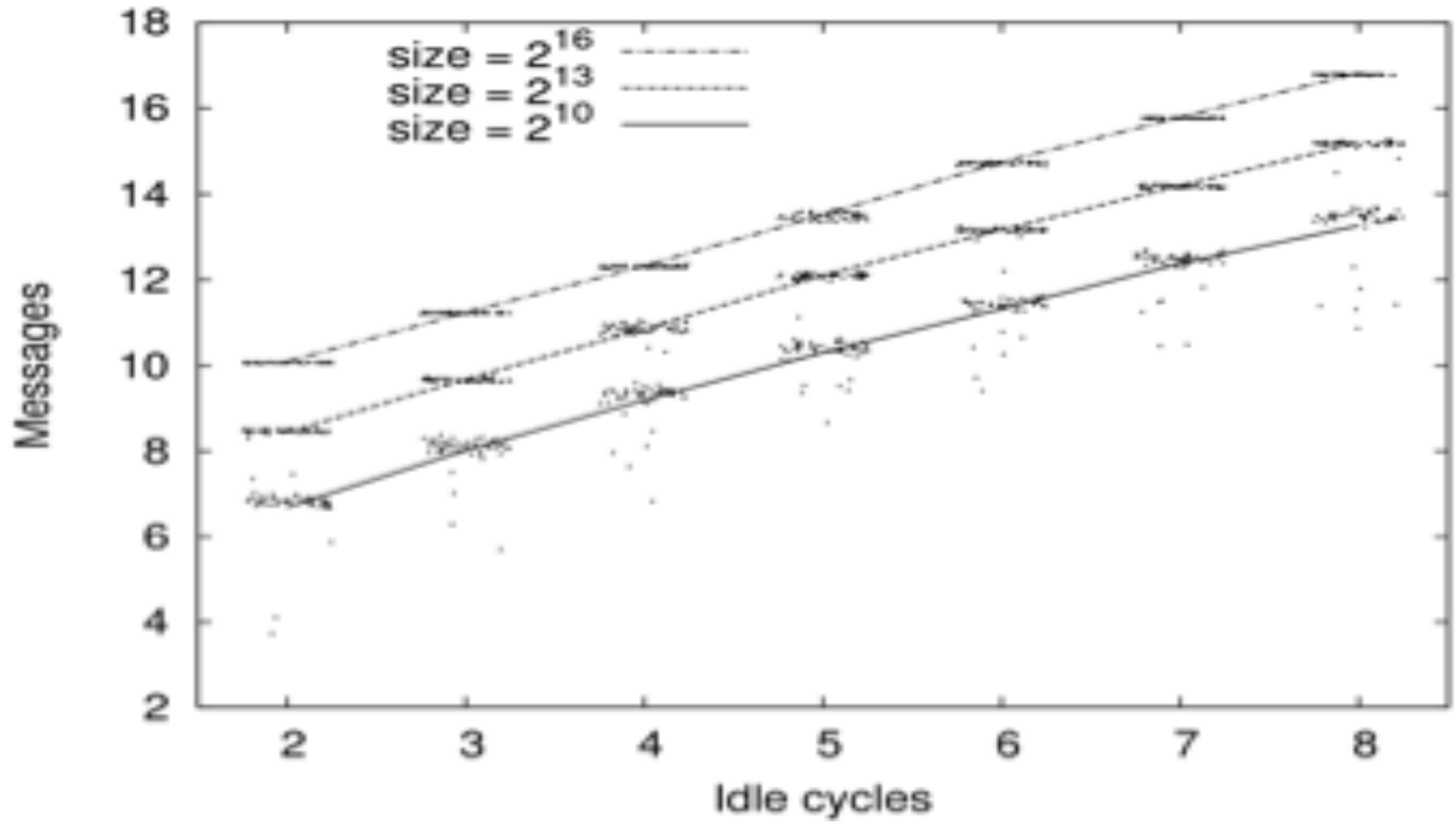
© Alberto Montresor

Stopping bootstrap: different values for idle



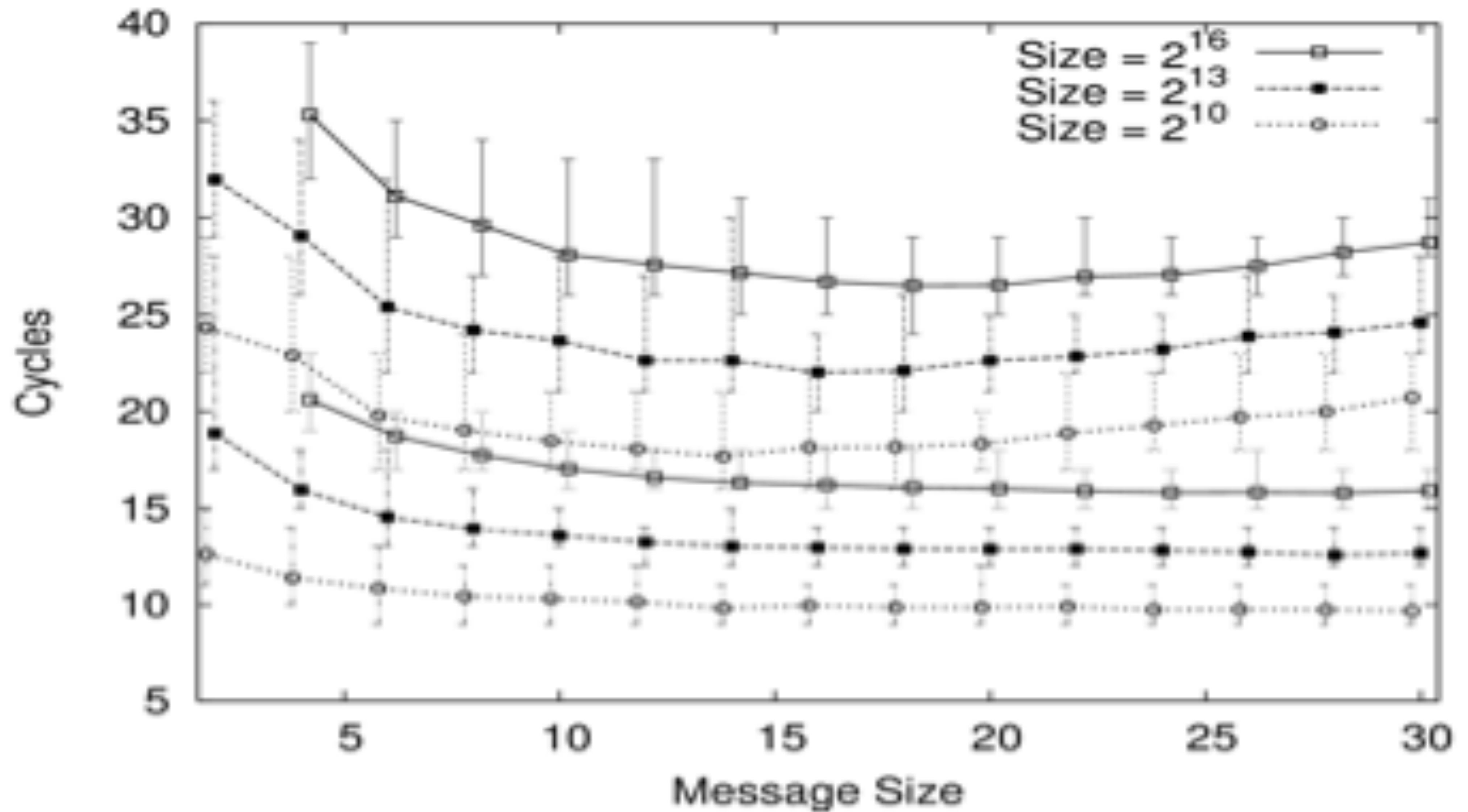
© Alberto Montresor

Communication costs



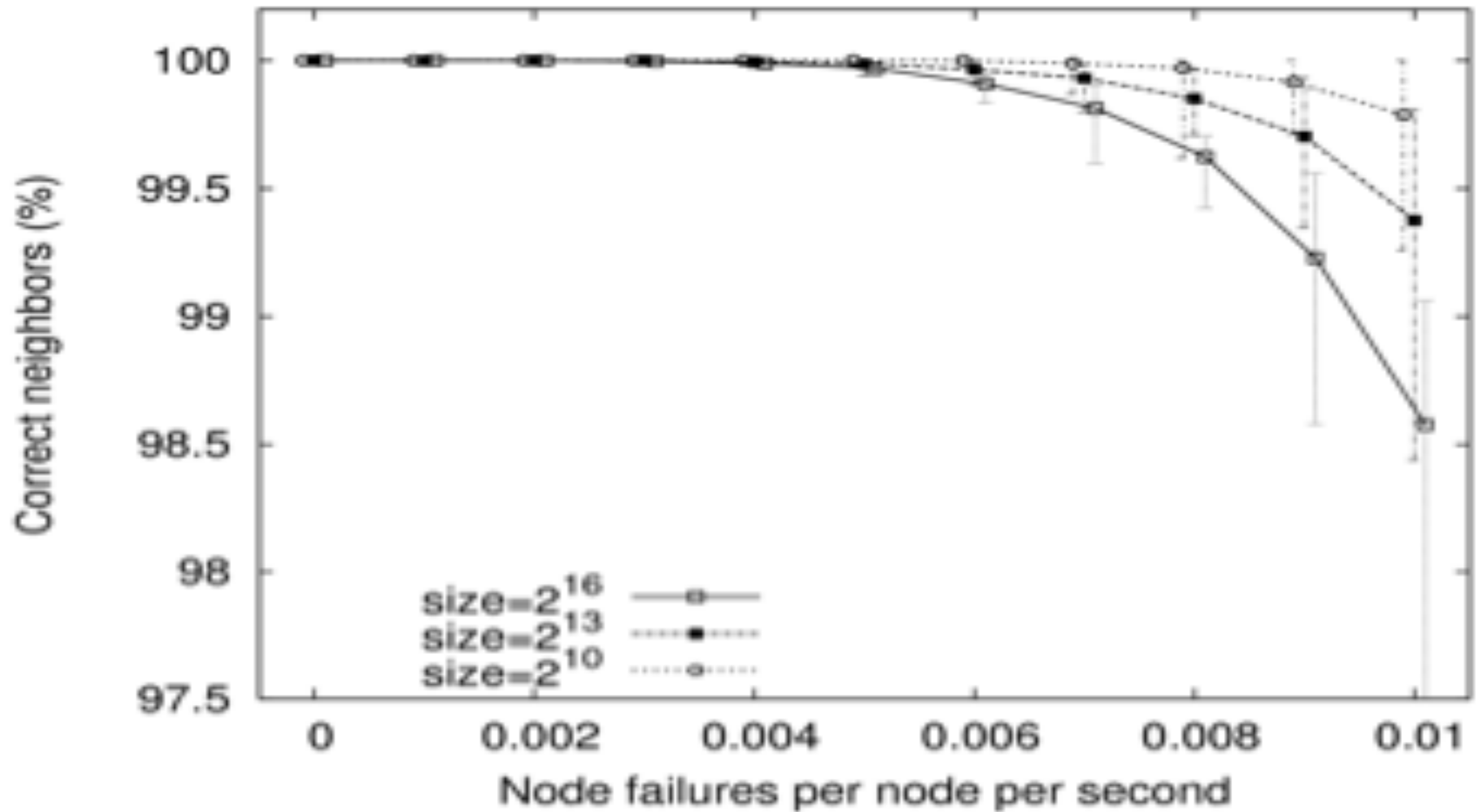
© Alberto Montresor

Parameter tuning: message size



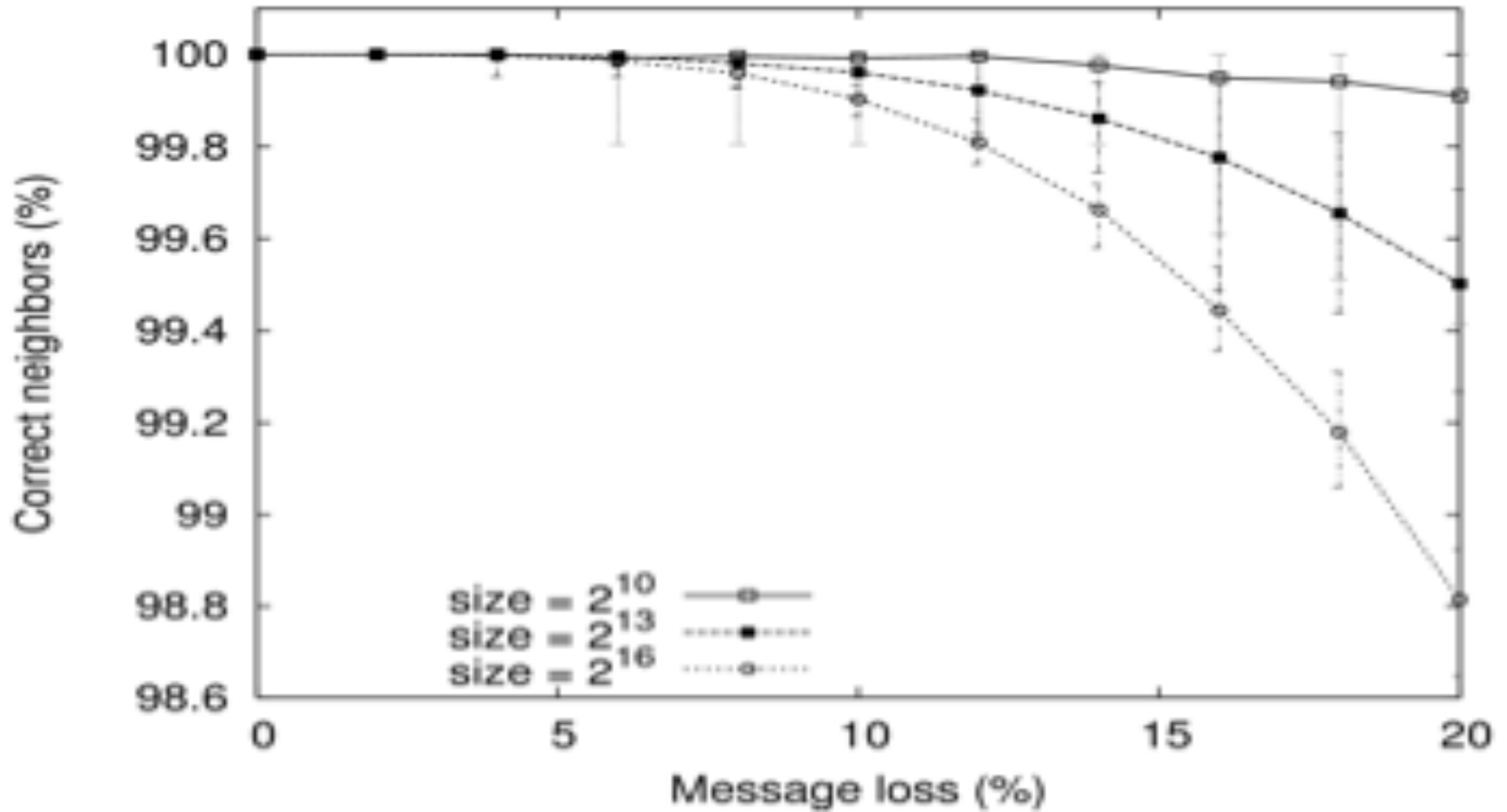
© Alberto Montresor

Robustness to failures



© Alberto Montresor

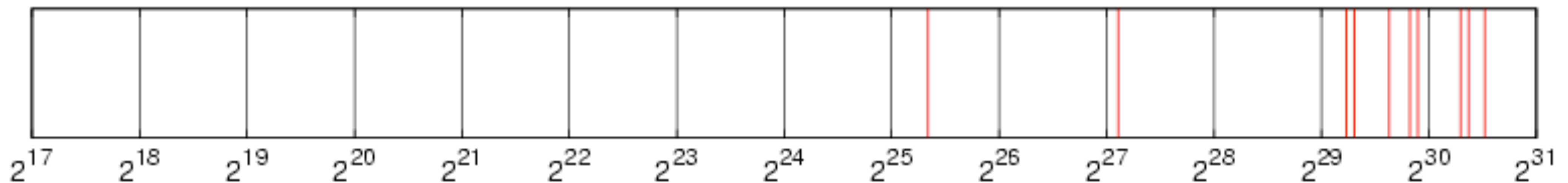
Robustness to message losses



© Alberto Montresor

T-Chord

- **How it works?**
  - Node descriptor contains node ID in a *[0..2^t[* space
  - Nodes are sorted over the ring defined by IDs
  - Final output is the Chord ring
  - As by-product, many other nodes are discovered
- **Example:**
  - t=32, size=2^{14}, msg size=20



© Alberto Montresor

T-Chord

- **Run the T-Man until the ring is formed**
- **Use the closest neighbors to select successors**
- **Use the entire view to fill the finger table**
  - Let id_p be the identifier of p
  - For each $i = 0 \dots m-1$ (m is the number of bits)
  - search in the view the node with identifier id' s.t.
    - $id' \in [(id_p + 2^i) \bmod 2^m, (id_p + 2^{i+1}-1) \bmod 2^m]$
    - id' is the closest value in that range

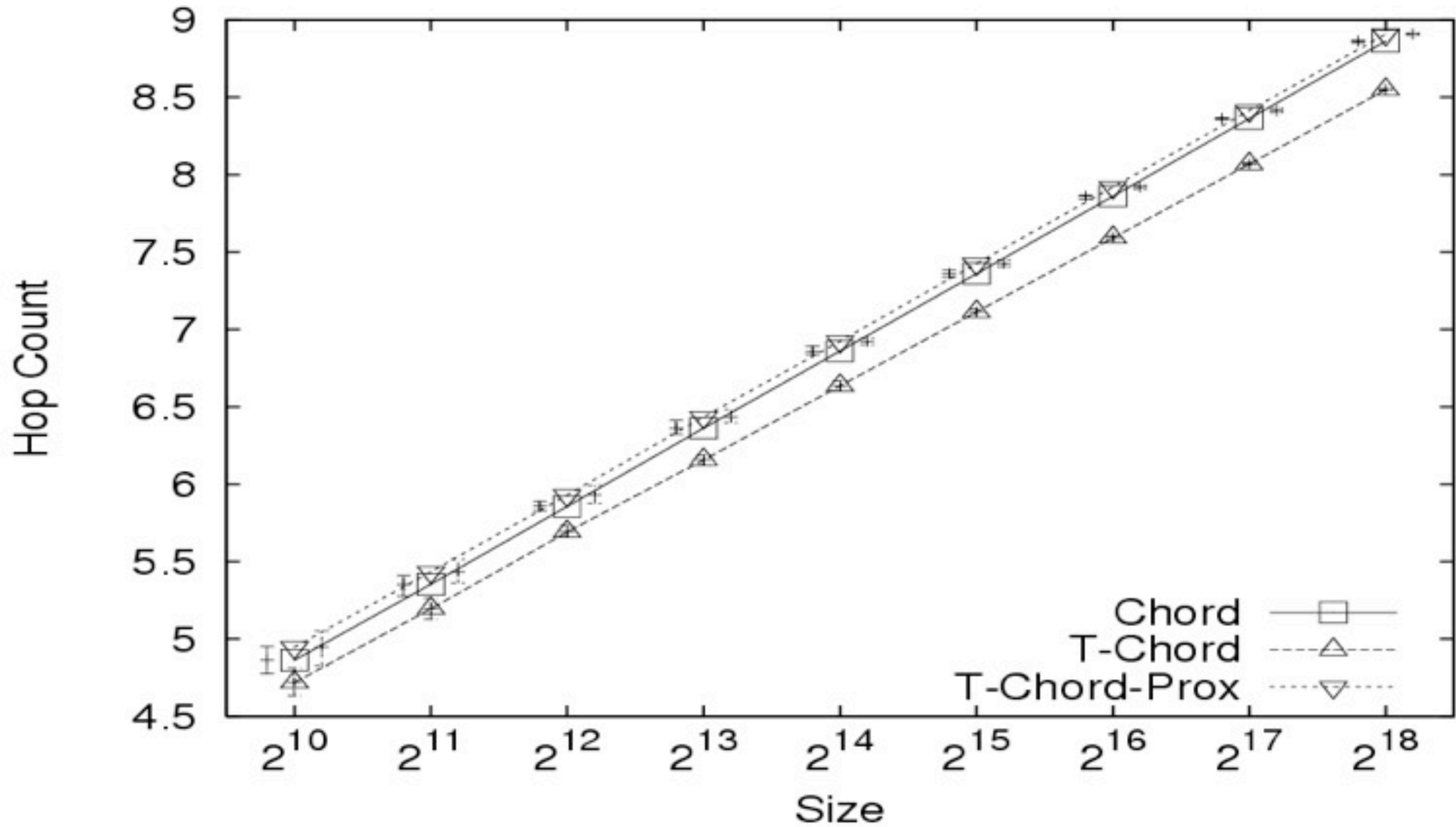
© Alberto Montresor

T-Chord-Prox

- **Run the T-Man until the ring is formed**
- **Use the closest neighbors to select successors**
- **Use the entire view to fill the finger table**
  - Let id_p be my identifier
  - For each $i = 0 \ldots m-1$ (m is the number of bits)
  - search in the history the node with identifier id' s.t.
    - $id' \in [(id_p + 2^i) \bmod 2^m, (id_p + 2^{i+1}-1) \bmod 2^m]$
    - the node is the closest (in terms of latency) among "probed" nodes (randomly selected)

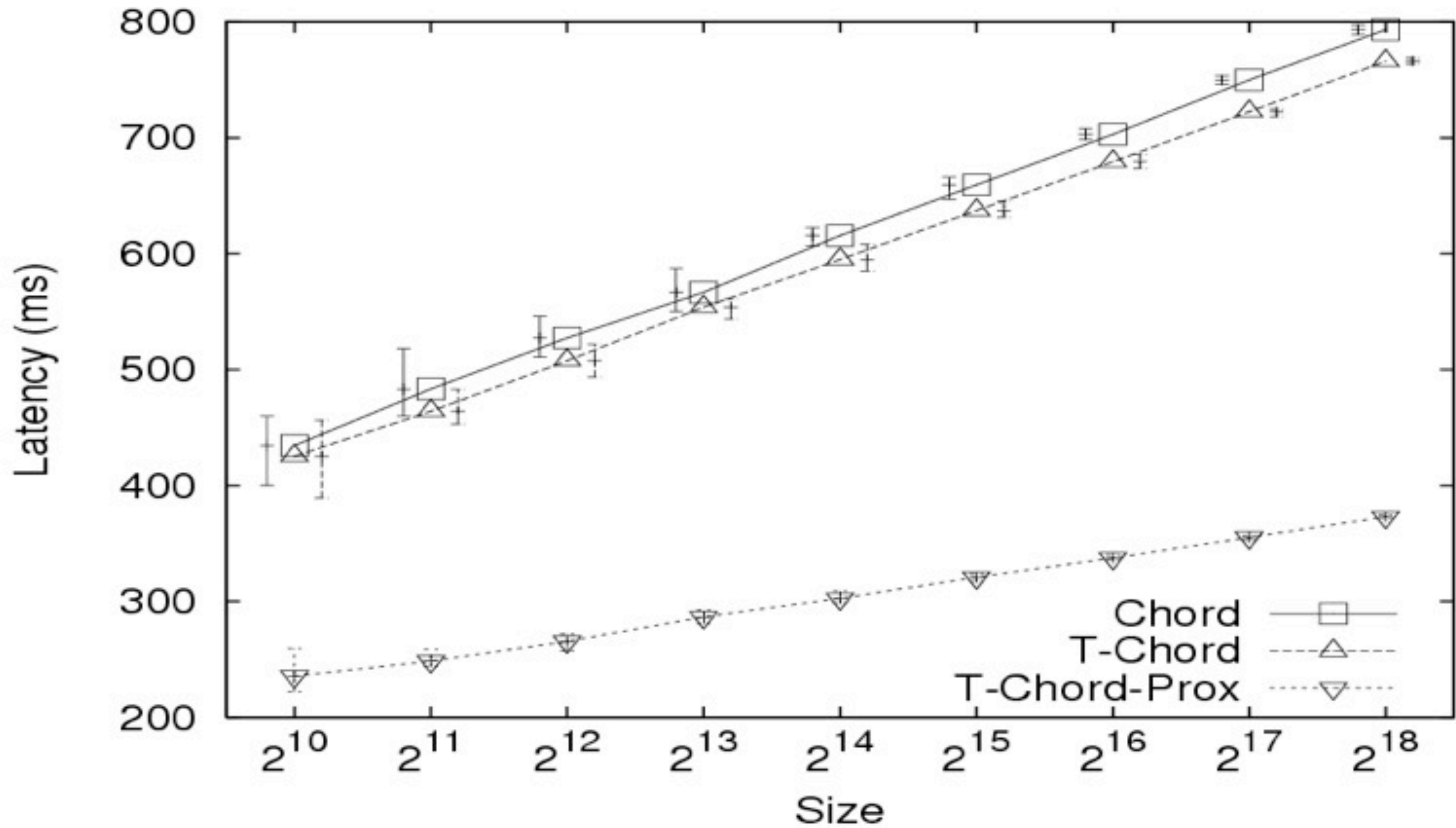
© Alberto Montresor

Scalability of T-Chord – Hop count



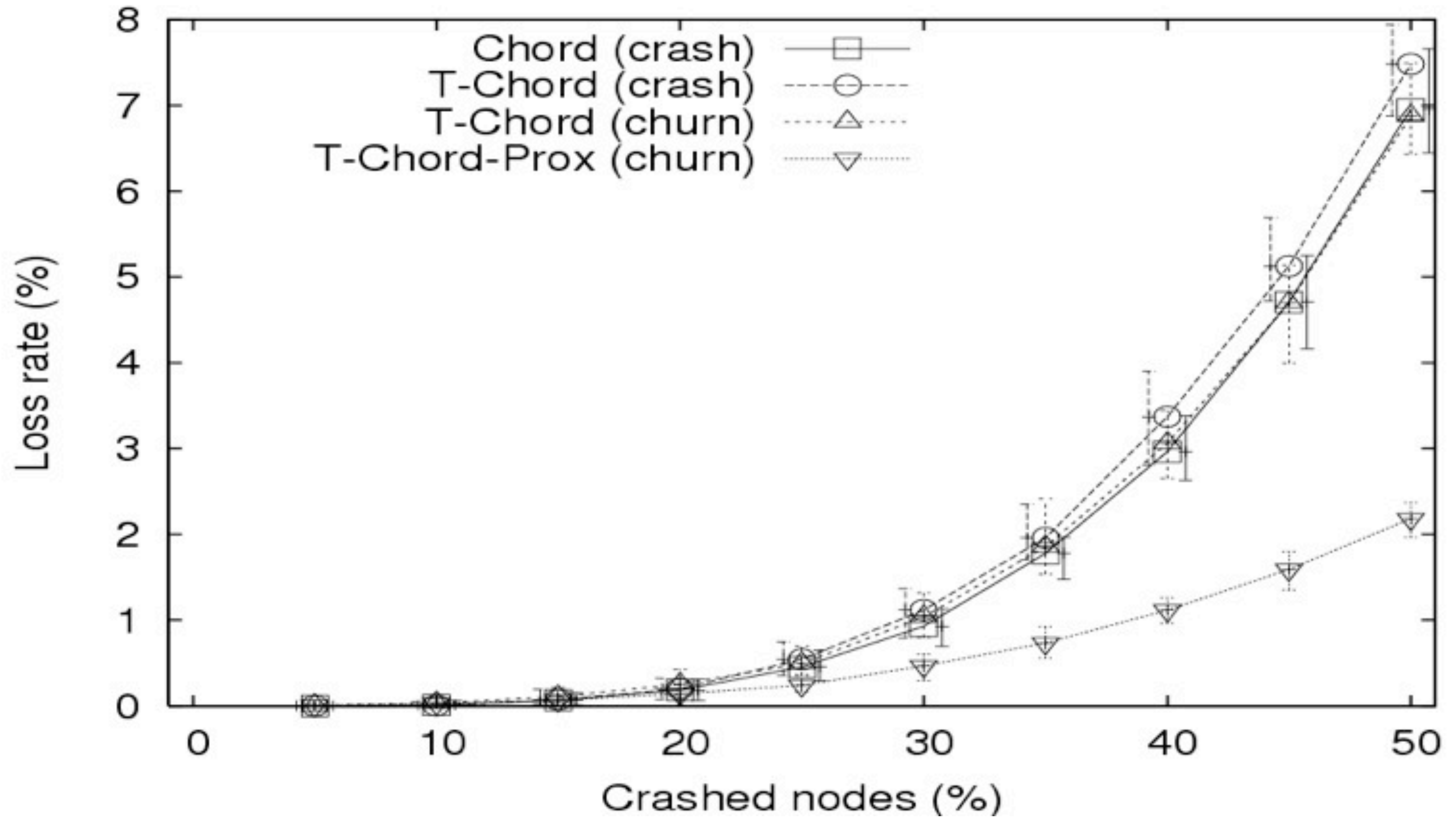
© Alberto Montresor

Scalability of T-Chord – Latency



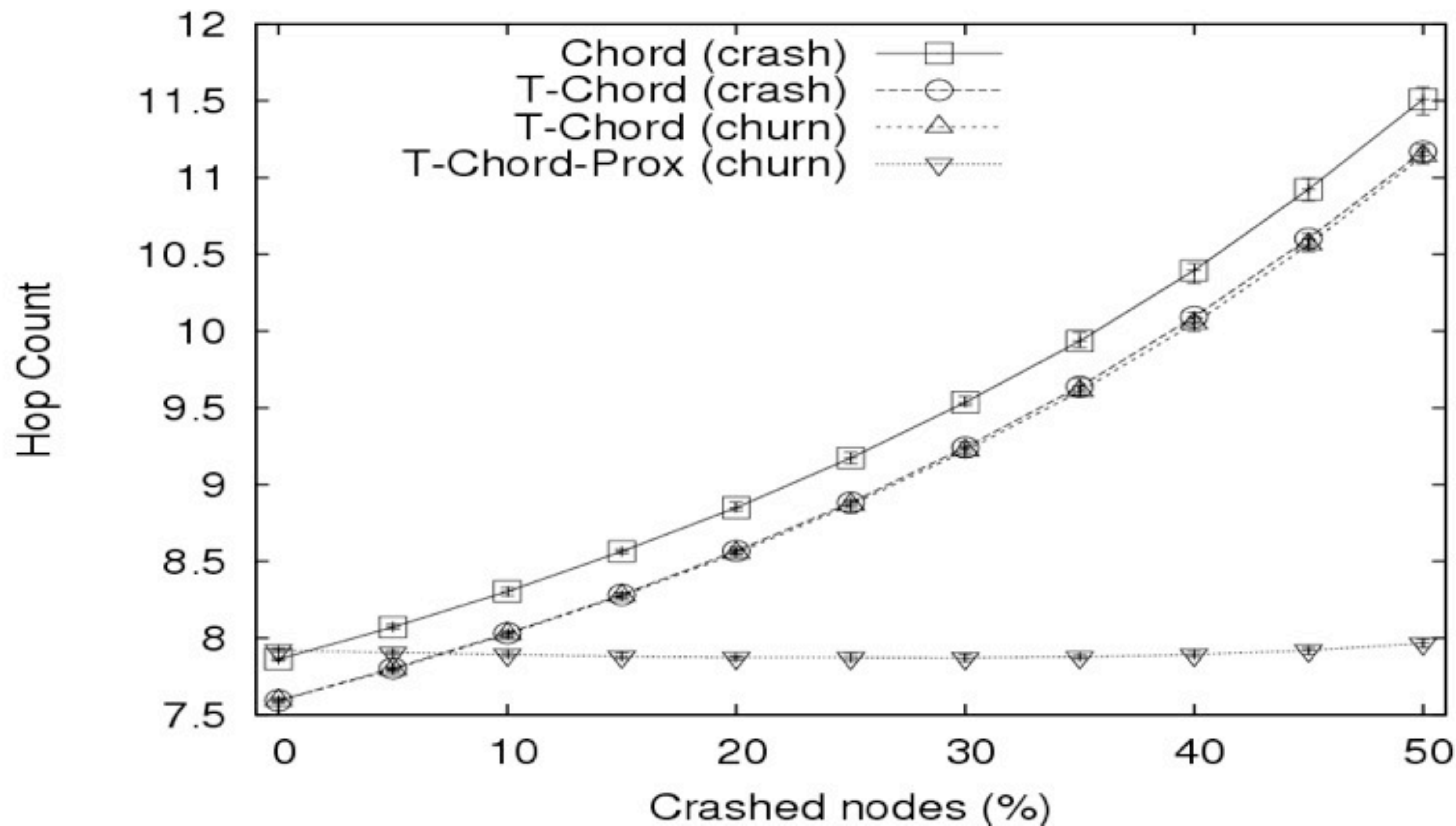
© Alberto Montresor

Robustness to failures



© Alberto Montresor

Robustness to failures



© Alberto Montresor

Conclusions

- **This mechanism to build Chord is tightly tailored on the particular structure of Chord**
- **A more generic approach:**
  - Define a ranking function where nodes have a preference for successors and predecessors AND fingers
  - Approx. same results, only slightly more complex to explain
- **Can be used for Pastry, for example:**
  - Define a ranking function where nodes have a preference for successors and predecessors AND nodes in the prefix-based routing table

© Alberto Montresor

Bibliography

- M. Jelasity, A. Montresor, and O. Babaoglu. *The bootstrapping service*. In Proc. of Int. ICDCS Workshop on Dynamic Distributed Systems (ICDCS-IWDDS'06), Lisboa, Portugal, July 2006. IEEE Computer Society.
- G.P. Jesi, A. Montresor, and O. Babaoglu. *Proximity-aware superpeer overlay topologies*. In Proc. of SelfMan'06, LNCS 3996, Dublin, Ireland, June 2006. Springer-Verlag. Best paper award.
- A. Montresor, M. Jelasity, and O. Babaoglu. Chord on demand. In Proc. of the 5th Int. Conf. on P2P Computing. Konstanz, Germany, August 2005.

© Alberto Montresor