Autonomous Replication for High Availability in Unstructured P2P Systems

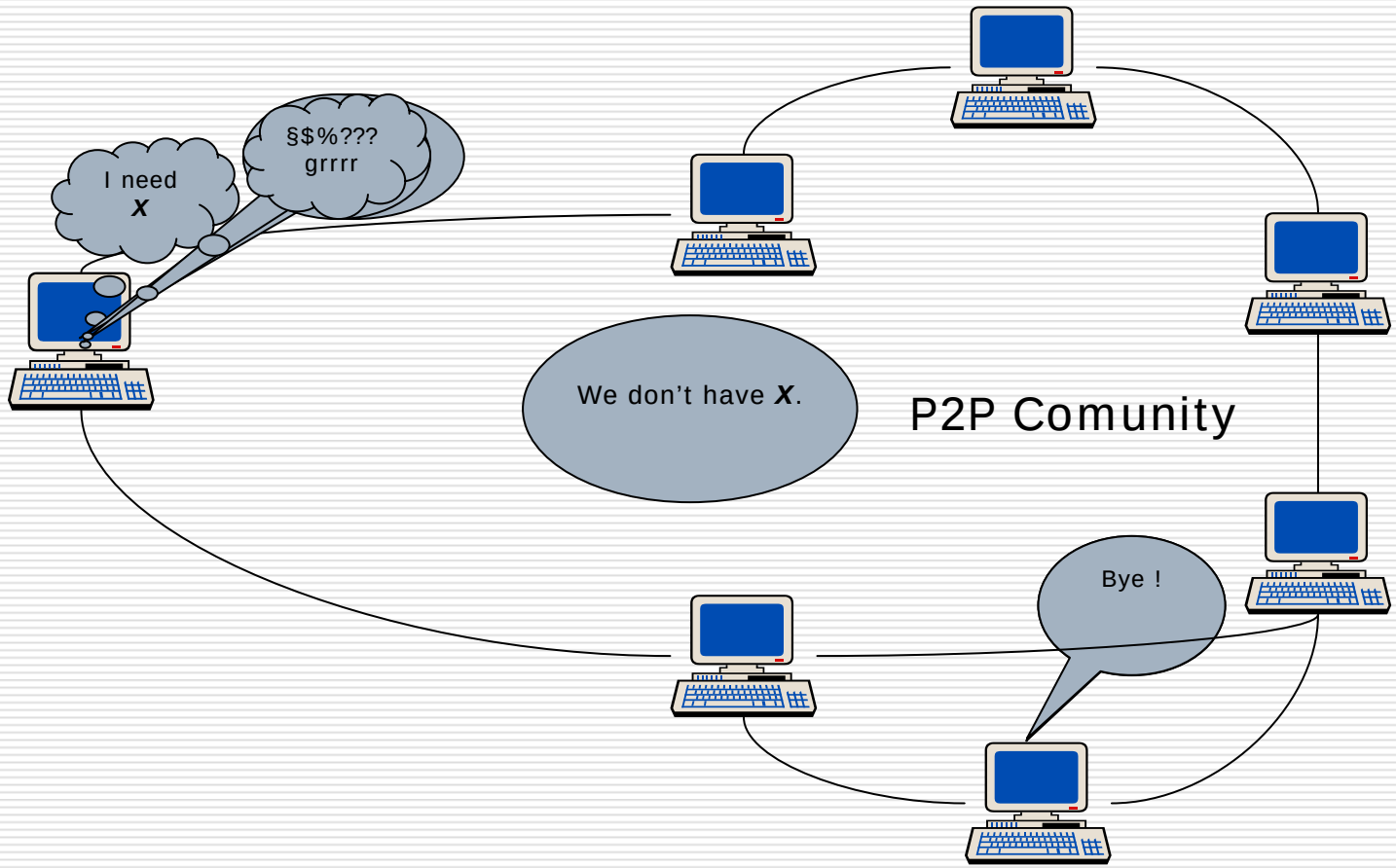
(Paper by Francisco Matias Cuenca-Acuna, Richard P. Martin, Thu D. Nguyen)

Hristo Pentchev

Proseminar

Peer-to-Peer Information Systems

Motivation



Overview

- Introduction
- PlanetP
- Autonomous Replication
- Simulations

- P2P computing is getting important
  - Availability
    - Data availability
    - Peer availability in P2P communities
- A decentralized and autonomous replication algorithm
- Increasing the availability of shared data using this algorithm in weakly organized systems is possible

- Naive idea
  - Replicate a file
  - Send the replicas randomly
- The replication needs
  - Excess storage
  - Bandwidth

Example:

Replicating a file 6 times in P2P community with average peer availability 20% increases the file's availability to 79% but needs lots of bandwidth.

100GB + 600GB = 700GB

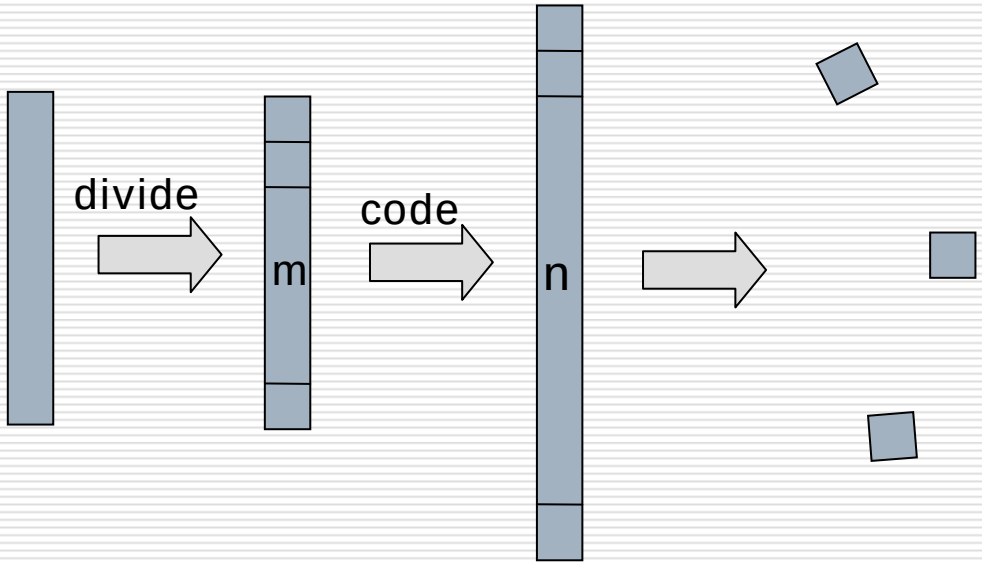
The availability is equal to the probability that all peers having the file are offline.

$$A = 1 - 0{,}8^7 = 1 - 0{,}2097 \approx 0{,}79$$

Conclusion: Naive replication is not effective.

How to improve the benefit of replication?

- Erasure coding
  - Divide a file in *m* fragments
  - Code this *m* fragments in *n* (*n* > *m*), so that the file could be reproduced with any *m* (unique) fragments
  - Send the fragments to random peers



We win with erasure coding:

- More availability
- Less bandwidth

because constant changes to the online membership do not require the movement of replicas any more

Overview

- Introduction
- PlanetP
- Autonomous Replication
- Simulations

- Gossiping-based publish/subscribe infrastructure
  - Maintaining the loosely synchronized global data
  - Gossiping in PlanetP - spreading new information

    Information needed by our algorithm:
    - Replica-to-peer mapping
    - Peer availability

Node

- PlaneP architecture
  - Local index = content of shared files
  - Global index = state of the community

Nickname	Status	IP	Bloom Filter
Bob	ON-LINE	10.25.125.10	BF[.....]
Gera	OFF-LINE	10.67.12.101	BF[.....]
...	...	...	...
Fred	ON-LINE	45.23.78.115	BF[.....]
Juri	ON-LINE	125.37.167.15	BF[.....]
Tan	ON-LINE	10.25.125.11	BF[.....]

Overview

- Introduction
- PlanetP
- Autonomous Replication
- Simulations

- Nodes have hoard set and fragment store
  - Hoard set – used by offline operations
    - Hoarding entire files
  - Fragment set - used by the algorithm
    - The size of the fragment sets is limited

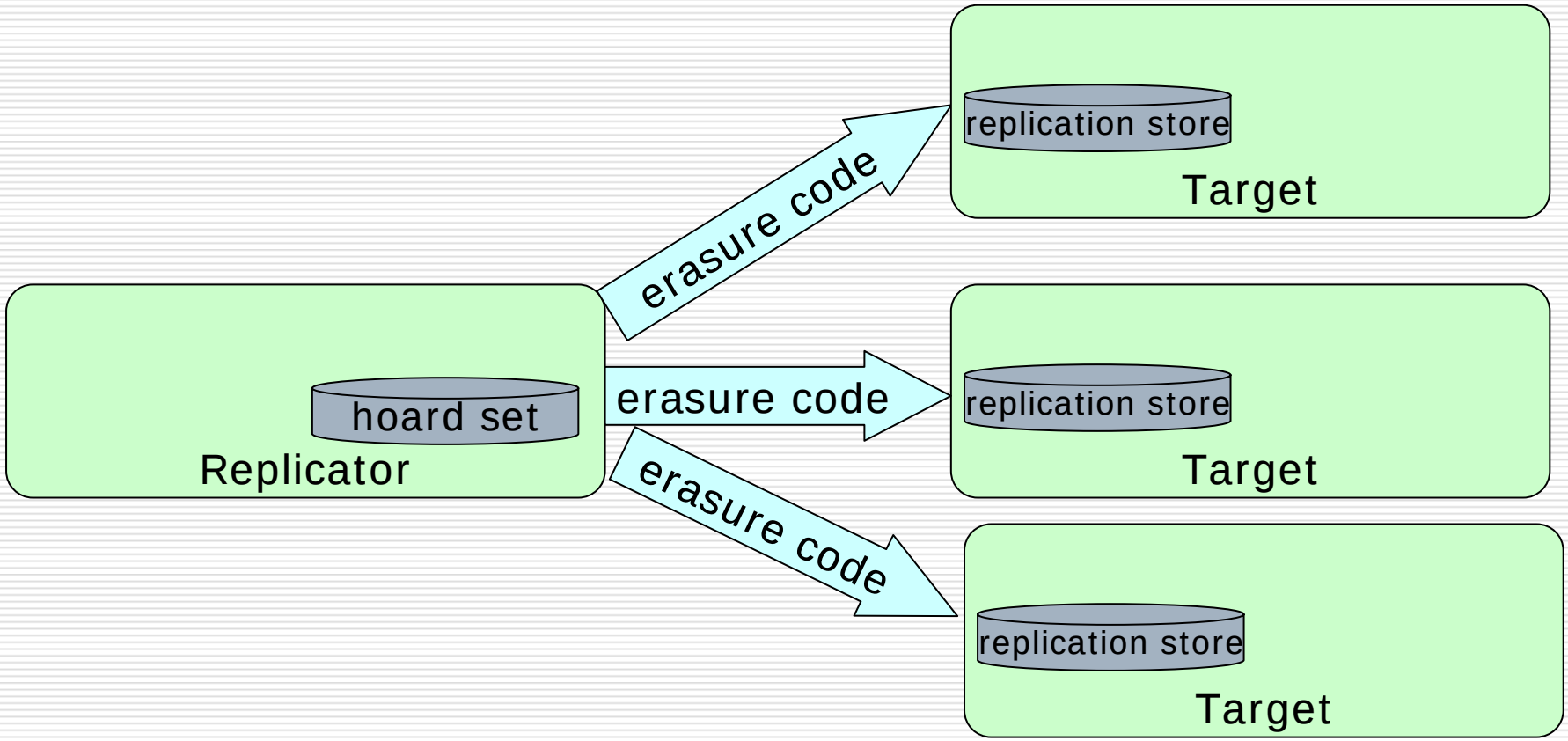
- Files have unique ID
- Availability of fragments

$$x \in Fragments$$

$$y \in Files$$

$$Availability(x) = Availability(y \mid x \subseteq y)$$

Autonomous Replication



- The algorithm
  - Each member advertises IDs in the global index (files + fragments)
  - Each member periodically estimates the availability of its files and fragments
  - Each member periodically generates random fragment of a random file and pushes it to a random target (with its availability)
  - The target saves the fragment or rejects it

Estimating the availability

- Nodes insert/remove $File\_Hash(f) \rightarrow m$ and $Frag\_Hash(f) \rightarrow m$ to/from the global index
- Peers advertise their availability in the global index
- Peers could either hoard a file or store only one fragment of it
- Peers go offline but not permanently
- Peers are dropped from the global index after timeout

$$H(f) = \{m \mid (File\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$F(f) = \{m \mid (Frag\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$P_i = \frac{avg.offlineTime}{avg.onlineTime + avg.offlineTime}$$

$$P_{avg} = \frac{1}{n} \sum_{i \in F(f)} P_i$$

$$A(f) = 1 - \prod_{i \in H(f)} P_i \sum_{j=n-m+1}^{n} \binom{n}{j} P_{avg}^{j} (1 - P_{avg})^{n-j}$$

Note: Don't work with duplicate fragments (small chance)

Fragmentation and replication

- Choose a random file with availability smaller then the wanted
- Produce some fragment of this file using erasure coding ($n >> m$)

Advantages of this kind of erasure coding:

1. Small probability of duplicating fragments
2. Usefulness of fragments
3. Easy to reflect changes in the community

Target's activity

- Receiving a request
  - If the replication store is not full then save the fragment
  - If the replication store is full then

    compute the availability of the fragments in it $average\_A$
    - if

      $$Availability(new\_fragment) > average\_A + \frac{average\_A}{10}$$

      then reject the request
    - else we have the case that

      $$Availability(new\_fragment) \leq average\_A + \frac{average\_A}{10}$$

      then replace some fragment in the store with the new one

    But which one?

Weighted random selection

- We make lottery with tickets divided in two bowls like 80:20.
- Every fragment receives the same portion from the "small" one.
- The tickets in the "big" bowl are shared between the fragments having availability bigger than the average depending on their availability.
- The "winner" must be ejected

Example:

We have 3 fragments *a,b,c* and 100 tickets:

$$No.nines = -\log_{10}(1 - availability)$$

Average availability (in nines) + 10% = 0,76

	Availability	*Availability in nines*	*Share big pool*	*Share small pool*	*Eviction probability*
a	0,99	2	67	6,6	0,73
b	0,9	1	13	6,6	0,19
c	0,5	0,3	0	6,6	0,06

So fragment *a* has the biggest chance to be rejected.

Optimizations on replicators

- Including some weighted random selection of files for replication
- Choosing targets with free space in the replication store

Misbehaving peers

- Corrupting fragments
  - No consequences for the file reproduction
  - Wasted space
- Pushing high available fragments
  - No consequences for the usage of replication store
  - Wasted bandwidth

- Wrong information

  Worst case: a peer advertises very high availability and that it is hoarding a significant part from the shared data

  - The community will not replicate this files
  - Much free storage for greedy peers
  - The peer will receive lot of requests for the files it's "hoarding"

Overview

- Introduction
- PlanetP
- Autonomous Replication
- Simulations

- Assumptions for the simulator
  - Synchronous replication's attempt
  - Simplified message transfer timing simulation
  - Estimation of the availability every 10 min.
  - Erasure coding with $m=10$
- Simulated communities

	File Sharing	**Corporate**	**Workgroup**
No. Members	1 000	1 000	100
No. Files	25 000	50 000	10 000
Node availability	24,9%	80,7%	33,3%

- Results computed using equation 1 as analytical model (Figure 2)

	excess storage	availability
CO	1,75X	3 nines
WG	6X	3 nines
FS	8X	3 nines

  - High members average availability increases the availability

Figure 2

- Results from the simulation

	excess storage	availability
CO	1X	3 nines
WG	9X(uniform)	3 nines
FS	6X	3 nines

Figure 3

- Server like peers increase the availability

- Centralized knowledge

  Compare our algorithm (Rep) with one having centralized perfect knowledge (Omni).

  - Less excess space used by Omni
  - Better minimum availability by Omni

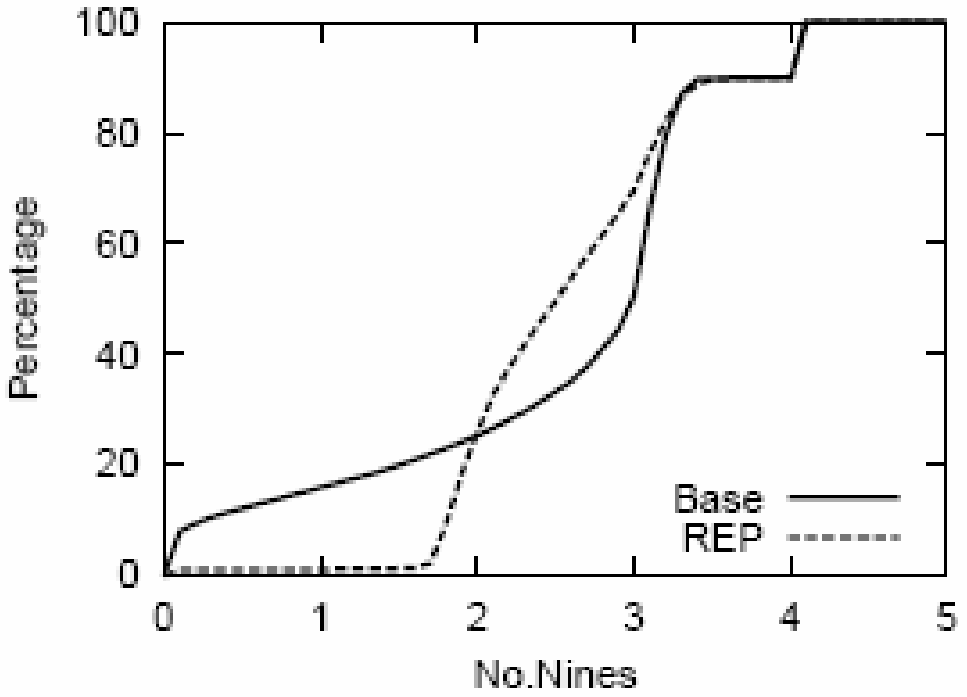
  Figure 3

- Availability-Based Replacement

  Compare our algorithm (Rep) with one replacing the fragments in replication store by FIFO rule (Base).

  - Higher availability with Rep

Simulations



Thank you for your attention !

Figure 2

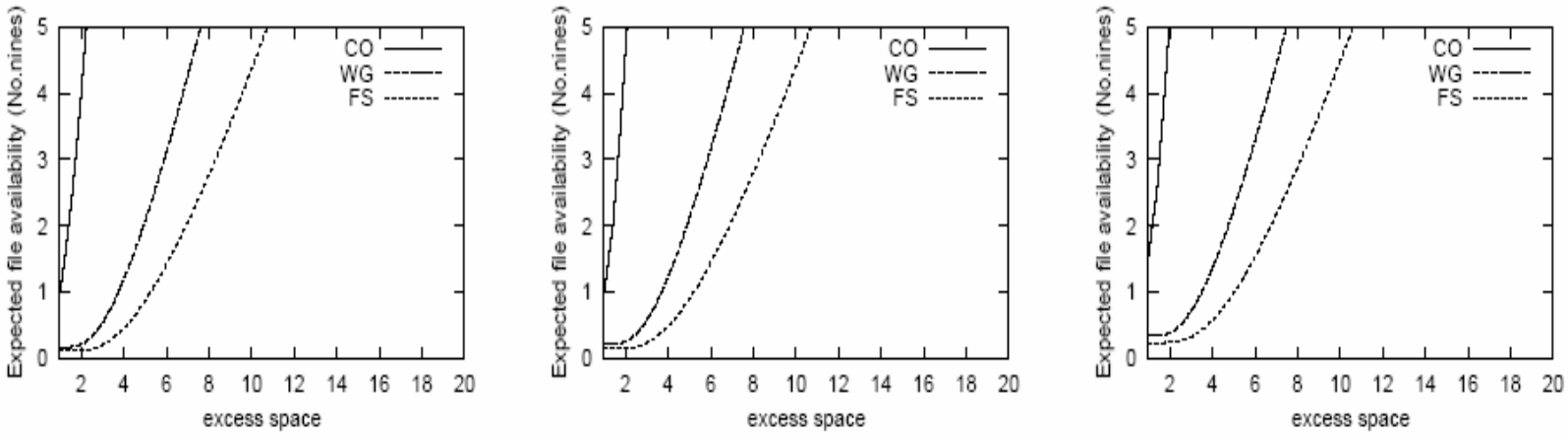


Figure 2: *Achieved file availability in number of nines plotted against excess storage in terms of x times the size of the entire collection of hoarded files for (a) 0 replication from hoarding, (b) 25% of the files have two hoarded replicas, and (c) 25% of the files have five hoarded replicas.*

Simulations

Figure 3

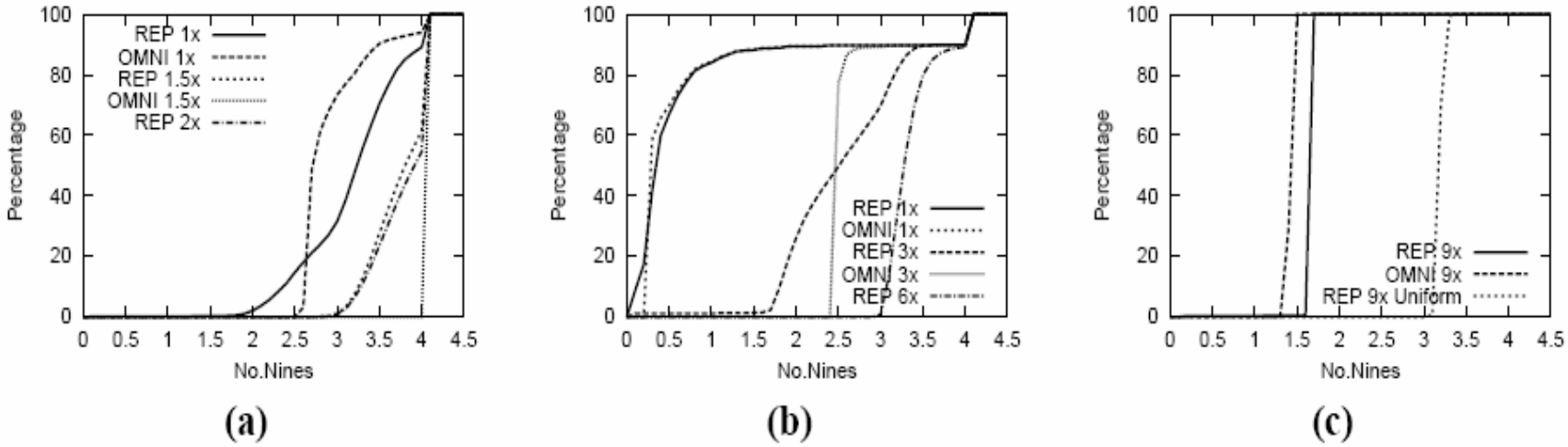


Figure 3: *CDFs for (a) the CO community with 1X, 1.5X and 2X, (b) the FS community with 1X, 3X, and 6X, and (c) the WG community with 9X excess storage capacities. REP refers to our replication algorithm as described in Section 3. OMNI is as described in Section 4.3.*

Simulations

Figure 3

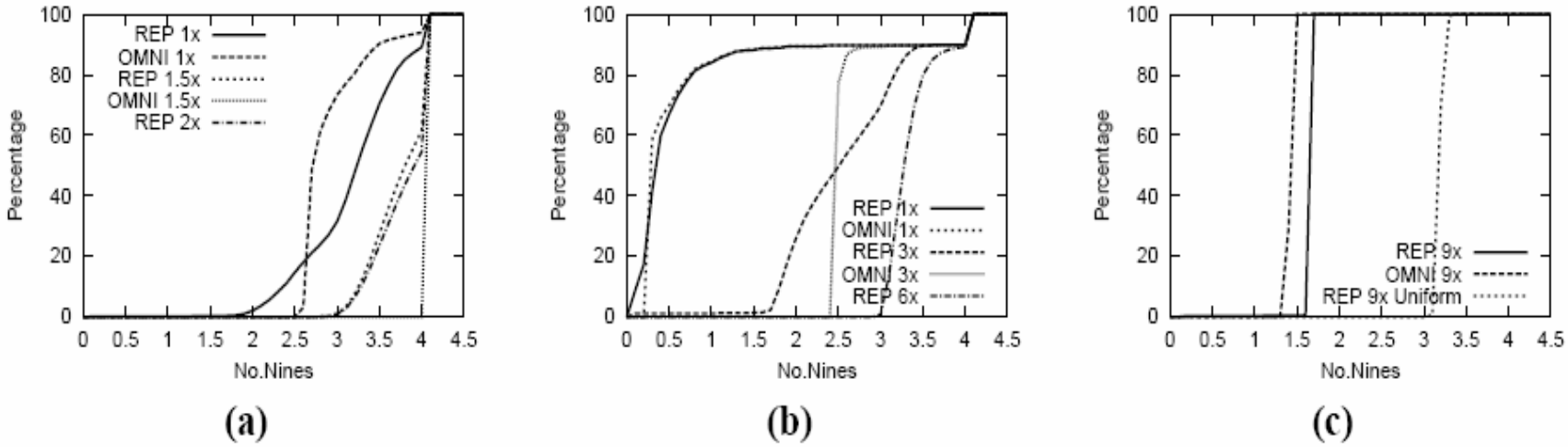


Figure 3: *CDFs for (a) the CO community with 1X, 1.5X and 2X, (b) the FS community with 1X, 3X, and 6X, and (c) the WG community with 9X excess storage capacities. REP refers to our replication algorithm as described in Section 3. OMNI is as described in Section 4.3.*

Simulations

$$H(f) = \{m \mid (File\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$F(f) = \{m \mid (Frag\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$P_i = \frac{avg.offlineTime}{avg.onlineTime + avg.offlineTime}$$

$$P_{avg} = \frac{1}{n} \sum_{i \in F(f)} P_i$$

$$A(f) = 1 - \prod_{i \in H(f)} P_i \sum_{j=n-m+1}^{n} \binom{n}{j} P_{avg}^{j} (1 - P_{avg})^{n-j}$$

Back

Appendix: PlanetP – Membership

NEW

OFF-LINE

Joining of new peers:

- **gossiping**

Leaving of present peers:

- **a peer discovers that another peer is OFF-LINE when an attempt to communicate with it fails:**
  - **the peer status is marked as OFF-LINE**
  - **information in the global index is not dropped**
- **if a peer has been marked as OFF-LINE continuously for a time T_{Dead} , it is assumed that the peer has left the community permanently:**
  - **all information about the peer is dropped**

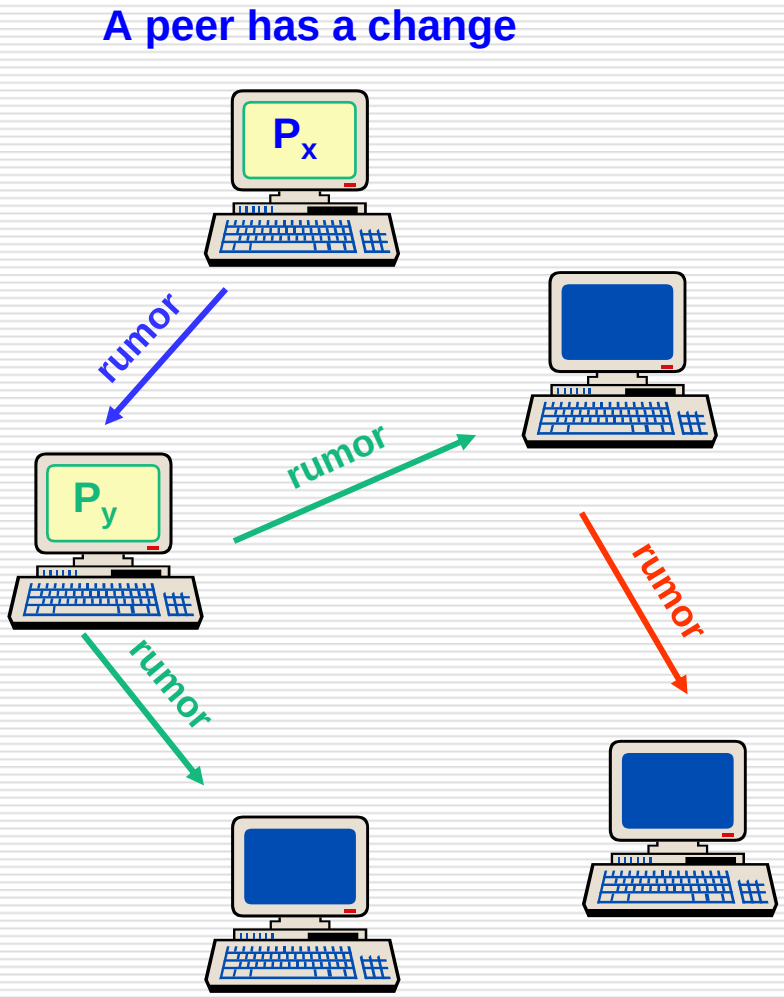
ON-LINE

Rejoining of peers (before T_{Dead}):

- **gossiping (the peer status is marked as ON-LINE)**

Appendix: Rumoring algorithm

The algorithm provides spreading of new information across a P2P community

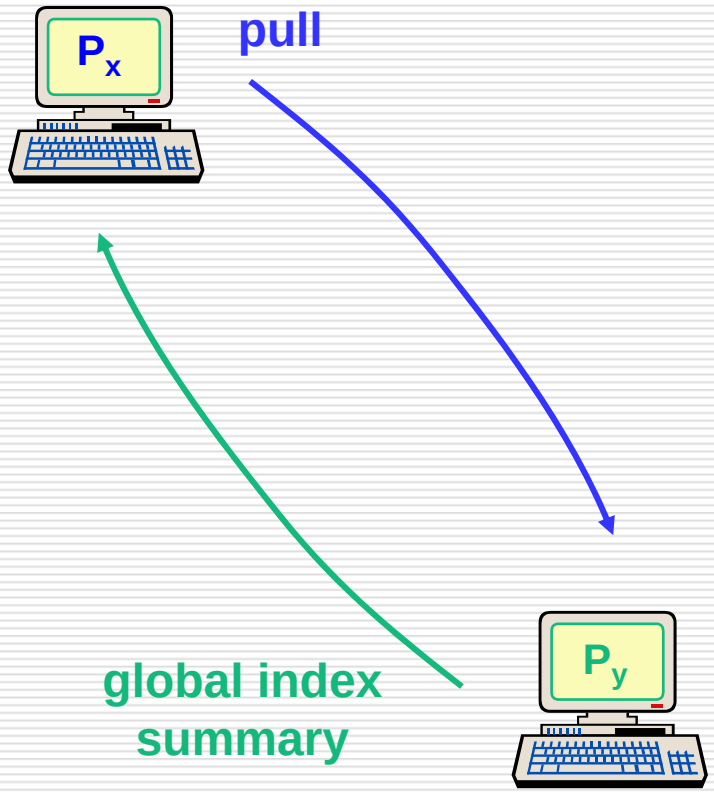


A peer has a change:

- every T_g seconds, a peer P_x pushes this change (a rumor) to a peer P_y chosen randomly from the global index
- if the rumor is new information for the peer P_y, then it starts to push this rumor just like P_x
- the peer P_x stops pushing the rumor after it has contacted n consecutive peers that already heard the rumor

Appendix: Anti-entropy algorithm

The algorithm allows to avoid the possibility of rumors dying out before reaching everyone

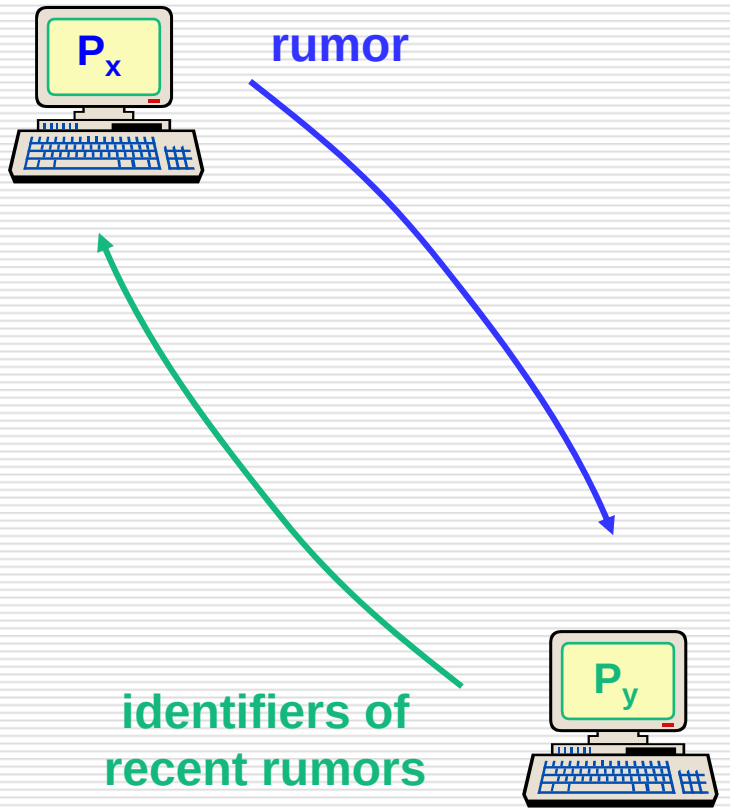


All peers:

- every T_r seconds, a peer P_x attempts to pull information from a peer P_y chosen randomly from the global index
- the peer P_y returns the summary of its version of the global index
- then P_x can ask P_y for any new information that it does not have

Appendix: Partial anti-entropy algorithm

The algorithm allows to reduce the time of new information spreading



Extension of each push operation:

- a peer P_x pushes a rumor to a peer P_y
- the peer P_y piggybacks the identifiers of a small number of the most recent rumors
- then P_x can pull any recent rumor that did not reach it

The process requires only one extra message exchange in the case that P_y knows something that P_x does not