

Department of Computer Science

PURDUE
UNIVERSITY

Probabilistic Group Communication

Outline

1. From Reliable to Probabilistic Broadcast
2. Probabilistic Broadcast Algorithms
3. Multicast and Ordering

Background

▶ Asynchronous and heterogenous systems

- Varying processor speeds, network latencies
- Failures can not be distinguished from latency, cf. [Fisher et al.'85]
- Reasoning with sequences of atomic actions?

▶ Group communication

- Reliable Broadcast et al. [Hadzilacos&Toueg'93]
- Reflect one-to-many interaction scenarios
- Fundamental building blocks of distributed systems

Issues

- ▶ **Deterministic group communication**
 - Sometimes impossible (scale...)
 - Sometimes overconstrained

- ▶ **Randomization**
 - E.g., alternative to failure detectors
 - Leads to probabilistic properties/specifications

- ▶ **Probability proliferation**
 - Probabilities “sum up”
 - Correlation

Reliable Broadcast

I. No duplication

- No message is **rdelivered** more than once

II. No creation

- No message is **rdelivered** unless it was **rbroadcast**

III. Validity

- If p is correct, then a message **rbroadcast** by p is eventually **rdelivered** by p

IV. Agreement

- If one correct process **rdelivers** a message m , every correct process eventually **rdelivers** m

Simple Algorithm

upon `rbroadcast(m)` :

`rsend(m)` to self

upon `rreceive(m)` :

 if (see `m` for first time)

`rsend(m)` to every process

`rdeliver(m)`

Assessment

► Complexity

- $O(n^2)$ messages

► Prerequisites

- Reliable channels (r-channels)
- Global views $O(n)$

Probabilistic Broadcast: An Intuition

```
upon pbroadcast(m) :
```

```
  rsend(m) to self
```

```
upon rreceive(m) :
```

```
  if (see m for first time)
```

```
    rsend(m) to random set of F processes
```

```
    pdeliver(m)
```

► Guarantees?

Probabilistic Reliable Broadcast

I. No duplication

- No message is **pdelivered** more than once

II. No creation

- No message is **pdelivered** unless it was **pbroadcast**

III. Validity

- If p is correct, then a message **pbroadcast** by p is eventually **pdelivered** by p

IV. Probabilistic agreement

- If one correct process **pdelivers** a message m , all correct processes eventually **pdeliver** m with known probability β

Assessment

- ▶ β tends to $e^{e^{-c}}$ with $F(n) = (\ln n + c + 1)$ [Kermarrec et al.'03]
 - $O(n \ln n)$ messages
- ▶ With probability of process failure ψ
 - $F'(n) = 1/(1-\psi) F((1-\psi)n)$
- ▶ $\beta = 1?$
- ▶ Channels?

Probabilistic Channels

p-channel

I. No duplication

- No message is **preceived** unless some process did **psend** it

II. No creation

- No message is **preceived** unless some process did **psend** it

III. Probabilistic validity

- If a *correct* process p_i **psends** a message m to a *correct* process p_j , then p_j **eventually preceives** m with known probability α

In Perspective

▶ Unreliable channels (u-channels)

- Weaker than probabilistic channels

▶ Reliable channels (r-channels)

III. Validity

- If a *correct* process p_i **sends** a message m to a *correct* process p_j , then p_j *eventually receives* m

- Stronger than p-channels

▶ Fair-lossy channels (f-channels)

III. Finite losses

- ▶ If a *correct* process p_i **sends** a message m to a *correct* process p_j an infinite number of times, then m gets lost only a finite number of times before being **received** by p_j

- Maximum losses X_{max}
- Stronger than probabilistic channels

Stubborn Probabilistic Channels

▶ *s-channel*

- Keep resending

III. 1-Validity

- If a *correct* process p_i *sends* a message m to a *correct* process p_j x times, then p_j *eventually receives* m with probability $P(x) = 1 - (1 - \alpha)^x$

▶ Still weaker than fair-lossy channel

- $P(x \leq x_{max}) = \dots$

- $P(x > x_{max}) = 1$

- x_{max} *unknown, but existing!*

- With $x \rightarrow \infty$ (and independent runs) a *s-channel* tends towards an *r-channel*, but never becomes one

- “Probability 1”

Reliable Broadcast with p-/s-channels?

I. No duplication

- No message is **rdelivered** more than once

II. No creation

- No message is **rdelivered** unless it was **rbroadcast**

III. Validity

- If p is correct, then a message **rbroadcast** by p is eventually **rdelivered** by p

IV. 1-Agreement

- If one correct process **rdelivers** a message m , all correct processes eventually **rdeliver** m with probability 1

Further

▶ ***pbcast*** with p-channels?

– $F'(n) = F(n)/\alpha = (\ln n + c + 1)/\alpha$

▶ Reliable Broadcast = Stubborn Probabilistic Reliable Broadcast?

– Keep **pbroadcasting** until **pdelivered**?

▶ Consensus?

Characteristics of Gossip-based Algorithms

▶ TTL

- *Forwards g* : a same process sends same information g times
- *Hops h* : same information is forwarded at most h times (causal chain of length h)

▶ Information

- Messages: payload
- Digests: identifiers

▶ Interaction

- Push: process sends information to arbitrary processes/digest sender
- Pull: process asks for information
- Mix, cf.[Karp et al.'00]

Example: *pbcast*

- ▶ Probabilistic Broadcast (*pbcast*) [Birman et al.'99]
 - Seminal work
- ▶ Two phases
 1. Best-effort broadcast based on spanning tree
 2. Gossip-based propagation of histories and according updates
 - Periodically every process chooses a random set of F processes and sends digest
 - Missing messages are sent
- 2. Based on complete membership $O(n)$
 - ▶ $O(n \ln n)$ messages
 - ▶ p-channels assumed

Example: *lpbcast*

- ▶ Lightweight Probabilistic Broadcast (*lpbcast*) [Eugster et al.'03]
- ▶ Based on partial membership
 - Assumes *uniform* views of size l
- ▶ Messages convey
 1. Application messages
 2. Digests
 3. Membership subsets
- ▶ Requires $O(n \ln n)$ messages

Analysis (Markov)

- ▶ Probability that a given gossip message infects a given (uninfected) process:

$$p = (l/n)(F/l)(1-\alpha)(1-\psi)$$

$$= (F/n)(1-\alpha)(1-\psi)$$

$$q = 1-p$$

- ▶ Probability of stepping from i infected processes to j infected processes at the next round:

$$p_{ij} = B(n-i, j-i)(1-q)^i (q)^{j-i} (q)^{n-j} \quad \text{with } B(x,y) = \binom{x}{y} = \frac{x!}{y!(x-y)!}$$

- ▶ $P(j \text{ infected at round } r) = \sum_{i \leq j} P(i \text{ infected at round } r-1) p_{ij}$

Membership Stability

► Probability of partitioning

- Partition of size $i > l$

$$B(n,i) (B(i,l) / B(n,l))^i (B(n-i,l) / B(n,l))^{n-i}$$

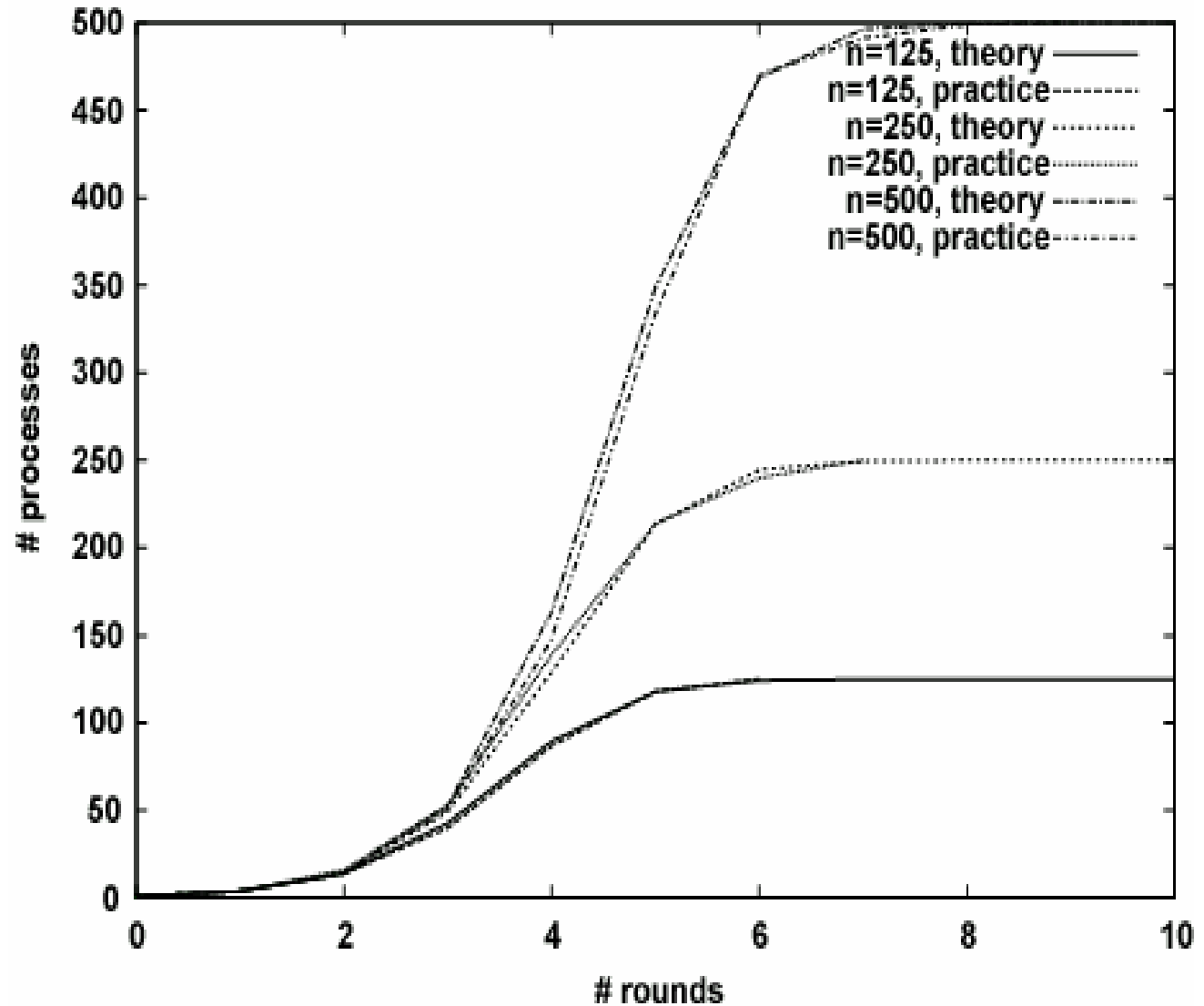
- Upper bound
 - Several partitions can be seen as recursive partitions
- Decreases with increasing l
- But also n
 - Becomes more stable with increasing system size
 - Total amount of membership information in the system increases

Uniform Views

- ▶ Every process has the same probability of appearing in any other process' view
 - On average l appearances; probability of p_i being in p_j 's view: l/n
- ▶ Performance **does** depend on l
 - Dependency
 - Causal dependencies between messages and destinations
 - Reliability
 - Latency increases, and buffers are limited
- ▶ Random walk, e.g., [King&Saia'04]
 - $O(\log n)$ messages, $O(\log n)$ latency
 - Still approximation (with very high probability)
- ▶ More pragmatic see [Jelasity et al.'04]

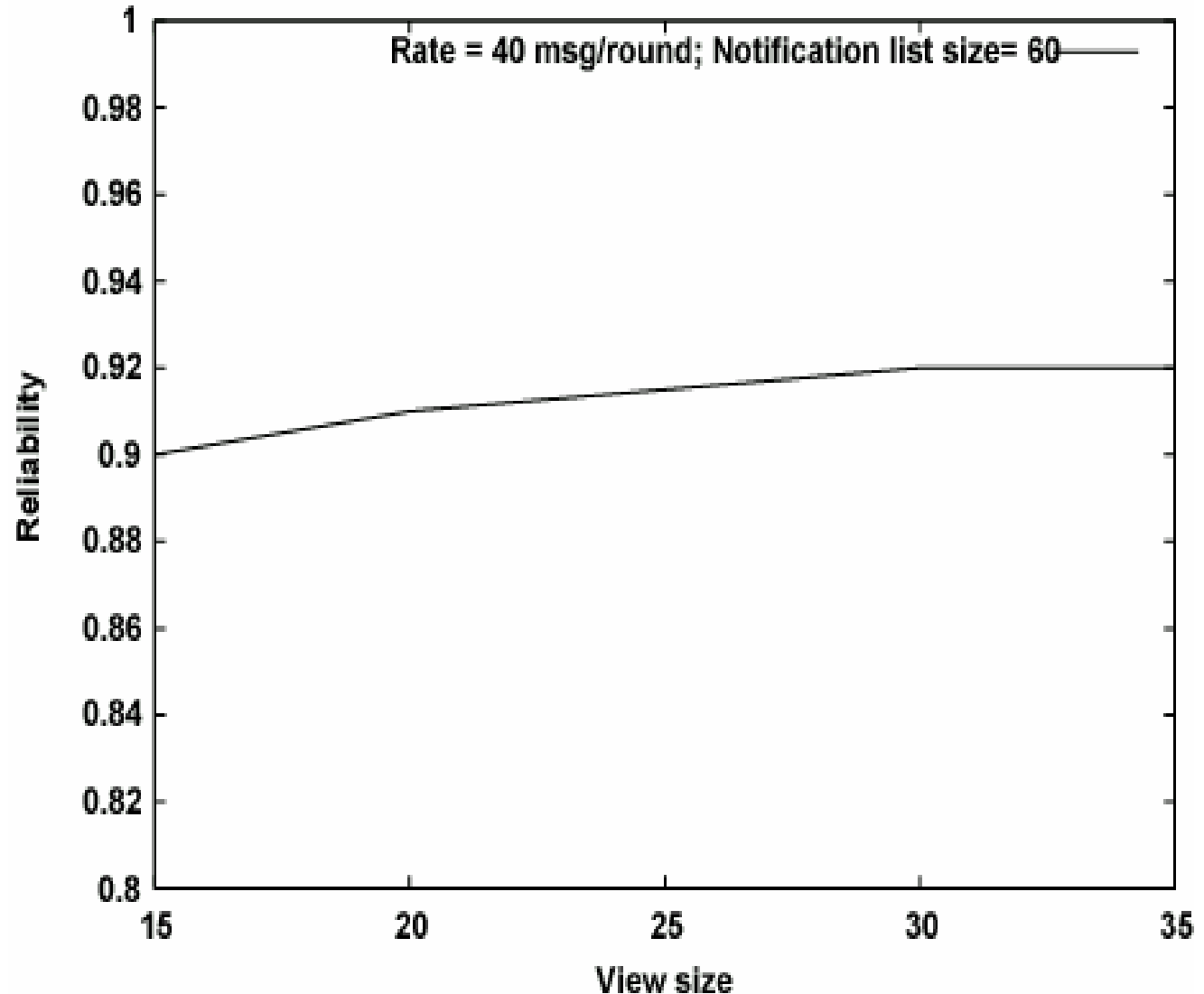
Analysis vs Simulation

- ▶ Fanout 3
- ▶ 1 msg injected
- ▶ Varying system size



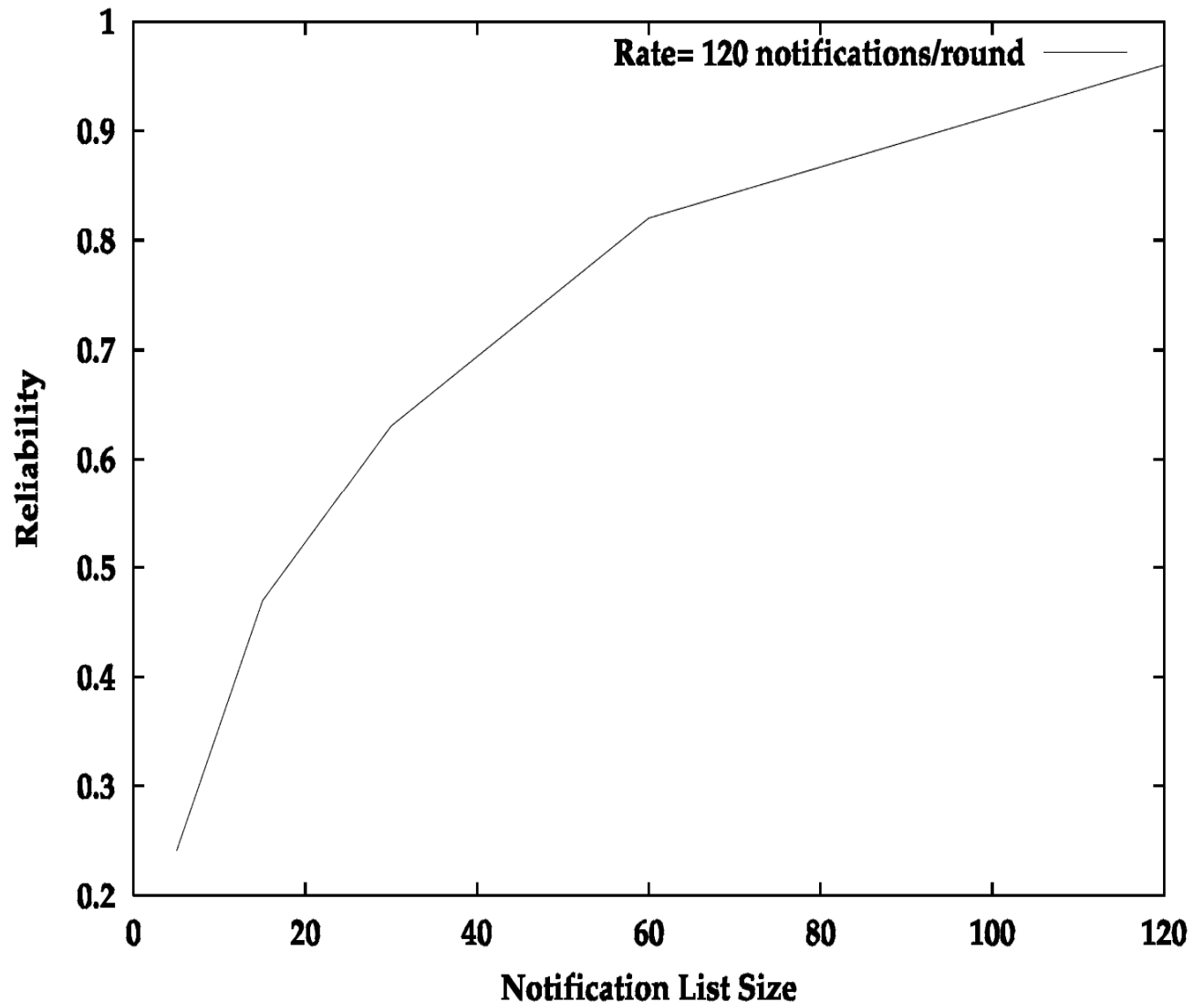
View Size and Reliability

- ▶ System size of 125
- ▶ Fanout 3
- ▶ 40 msgs/round are injected
- ▶ 60 msgs are in buffer
- ▶ Varying view size



Buffer Size and Reliability

- ▶ System size of 125
- ▶ Fanout 3
- ▶ 120 msgs/round are injected
- ▶ Varying buffer size



Guarantees

I. No duplication

- No message is **1pdelivered** more than once

II. No creation

- No message is **1pdelivered** unless it was **1pbroadcast**

III. Validity

- If p is correct, then a message **1pbroadcast** by p is eventually **1pdelivered** by p

IV. Probabilistic agreement

- If one correct process **1pdelivers** a message m , then with known probability β any given correct process eventually **1pdelivers** m

Comparison

▶ Atomicity

- A process has no means of knowing (unilaterally) whether “good” or “bad” run
 - What is “good” or “bad” run? Who are *all* the processes?
- Would require Reliable Broadcast primitive, global views

▶ Thus information is limited

- β is lower bound for β'
 - Reflects case where with probability β *all* processes `pdeliver`, and with probability $(1-\beta)$ *no single* process `pdelivers`

▶ Application-specific

Reliable Multicast

I. No duplication

- No message is **rdelivered** more than once

II. No creation

- No message is **rdelivered** unless it was **rmulticast**

III. Validity

- If p is correct, then a message **rmulticast** by p is eventually **rdelivered** by p

IV. Agreement

- If one correct process **rdelivers** a message m , every correct process interested in m eventually **relivers** m

Simple Algorithm

```
upon rmulticast(m, Dest):
```

```
    rsend(m, Dest) self
```

```
upon rreceive(m, Dest):
```

```
    if (see m for first time)
```

```
        rsend(m) to every process in Dest
```

```
        rdeliver(m)
```

Probabilistic Multicast: Simple Algorithm?

```
upon pmulticast(m) :
```

```
  rsend(m) to self
```

```
upon rreceive(m) :
```

```
  if (see m for first time)
```

```
    rsend(m) to F random interested processes
```

```
  pdeliver(m)
```

Total Order Broadcast

I. No duplication

- No message is **toDelivered** more than once

II. No creation

- No message is **toDelivered** unless it was **toBroadcast**

III. Validity

- If p is correct, then a message **toBroadcast** by p is eventually **toDelivered** by p

IV. Agreement

- If one correct process **toDelivers** a message m , every correct process eventually **toDelivers** m

V. Total order

- No two correct processes **toDeliver** two messages in different orders

Simple Algorithm

```
unordered := empty;
```

```
wait := false;
```

```
sn := 1;
```

```
ordered := empty;
```

```
tobroadcast(m) :
```

```
  rbroadcast(m) ;
```

```
rdeliver(m) :
```

```
  if id(m) !in ordered
```

```
    unordered = unordered  $\cup$  {m} ;
```

Simple Algorithm

```
upon(unordered !empty) && (!wait):  
    wait := true;  
    propose(sn, unordered);
```

```
upon(msgs = decided(sn++)):  
    for all m in msgs in det. order  
        unordered := unordered \ {m};  
        ordered := ordered  $\cup$  {id(m)};  
        todeliver(m);  
wait := false;
```

▶ ***topbcast* = *pbcast* + consensus?**

- Consensus
 - Builds on Reliable Broadcast
 - Requires failure detector
- ***topbcast* < *tobcast***
 - Weaker agreement (if $\beta < 1$)
 - Use it!

▶ **No global views**

- No global agreement
- In particular no Reliable Broadcast

Example: *pabcast*

[Felber&Pedone'02]

I. No duplication

- No message is **pade1ivered** more than once

II. No creation

- No message is **pade1ivered** unless it was **pabroadcast**

III. Probabilistic validity

- If p is correct, then a message **pabroadcast** by p is eventually **pade1ivered** by p with known probability φ

IV. Probabilistic agreement

- If one correct process **pade1ivers** a message m , any correct process eventually **pade1ivers** m with known probability β

V. Probabilistic total order

- If processes p and q **pade1iver** messages m_1 and m_2 , then with known probability ρ they do so in the same order

Assessment

▶ Probabilistic agreement captures

- Large scale (reachability)
- Large scale (awareness)
- Liveness

▶ But ordering?

- Safety
 - Hampers consistency
- Total order
 - E.g., concurrent updates

Probabilistic + Ordering Guarantees?

I. No duplication

- No message is **topdelivered** more than once

II. No creation

- No message is **topdelivered** unless it was **topbroadcast**

III. Validity

- If p is correct, then a message **topbroadcast** by p is eventually **topdelivered** by p

IV. Probabilistic agreement

- If one correct process **topdelivers** a message m , any given correct process eventually **topdelivers** m with a known probability β

V. Total order

- No two correct processes **topdeliver** two messages in different orders

Simple Algorithm?

▶ Determinism for message ordering

- $H(\text{messages}, \text{broadcasters})$

▶ Timestamps

- m_1 from p_1 has to be delivered before m_1 from p_2
- Failures?
 - Delay delivery of m_1 from p_2 until m_1 received from p_1 or timeout reached
 - If m_1 of p_1 received thereafter, omit delivery
- Unfeasible if **every** process can broadcast

Intuition: Hybrid Approach

- ▶ Need to balance between determinism (structure) and randomization (chaos)
- ▶ Observations
 1. In large-scale broadcasts only few broadcasters
 2. Internet is not fully symmetric. Built of interconnected “clouds”
- ▶ Hierarchical approach

Example: *pmcast*

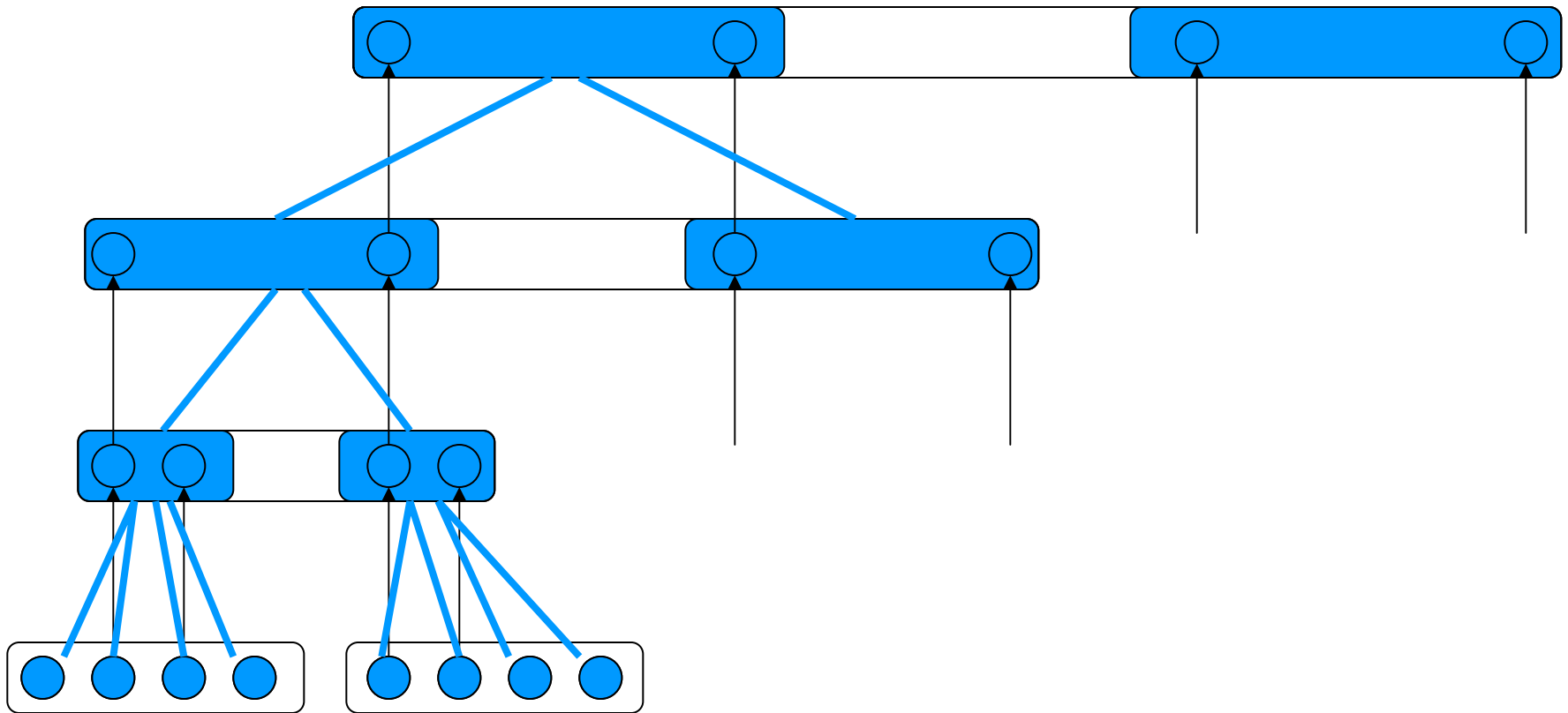
▶ Bottom-up

- Processes have precise knowledge about „close“ neighbors
- Knowledge decreases with the „distance“
- Processes are orchestrated in subgroups, according to distance
- *r delegates* are chosen for each subgroup, manifest interests of all represented processes
- Each process knows the *r* delegates of each (known) subgroup

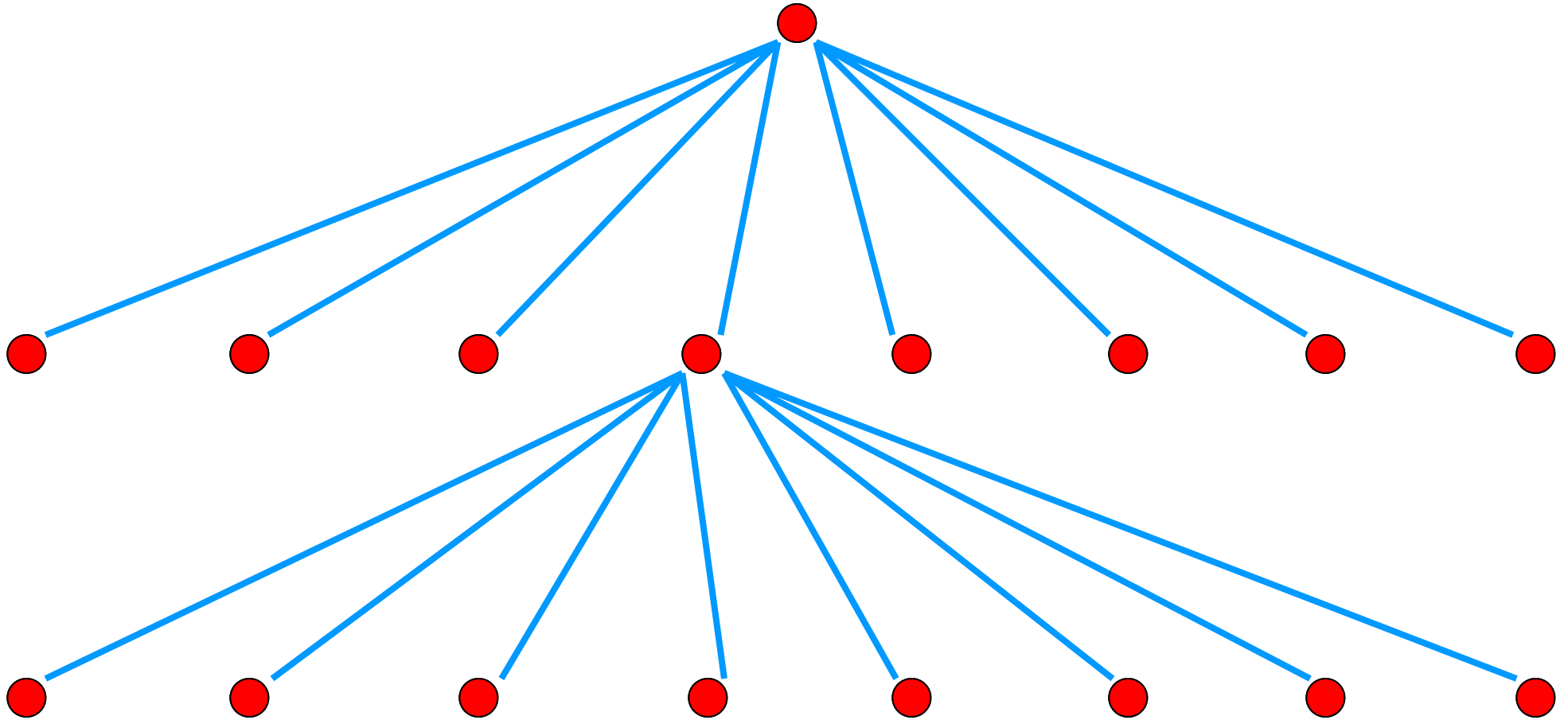
▶ Recursively

- Put several subgroups of level 1 together: form a group of level 2
- Elect *r* delegates for every such supergroup
- etc.

Illustration



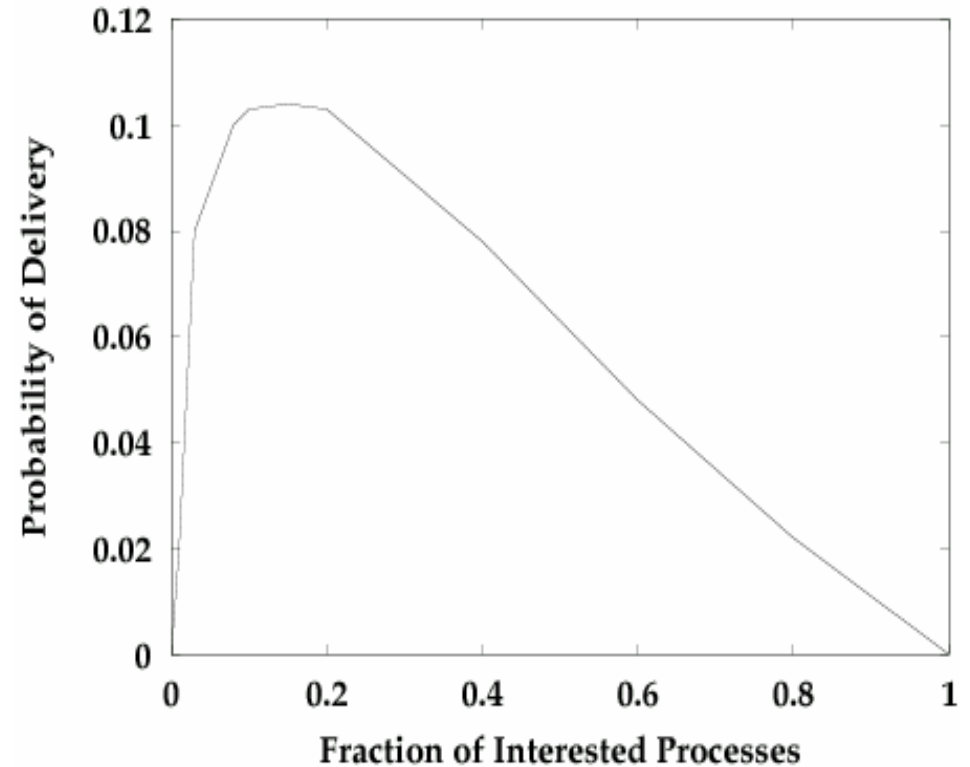
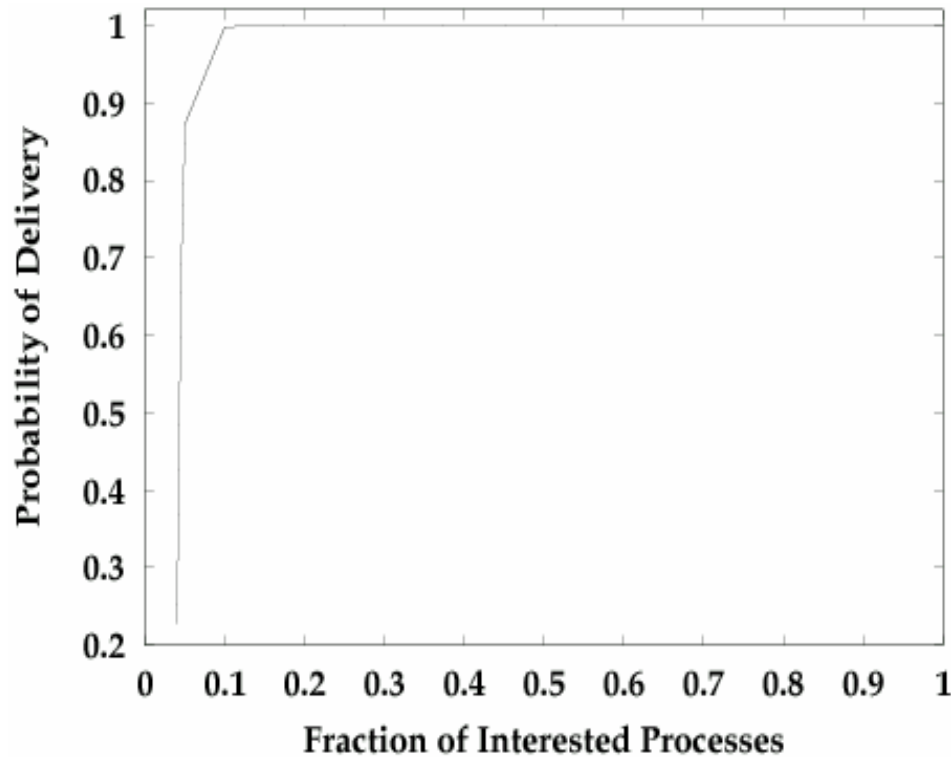
Illustration



Analysis (Diff')

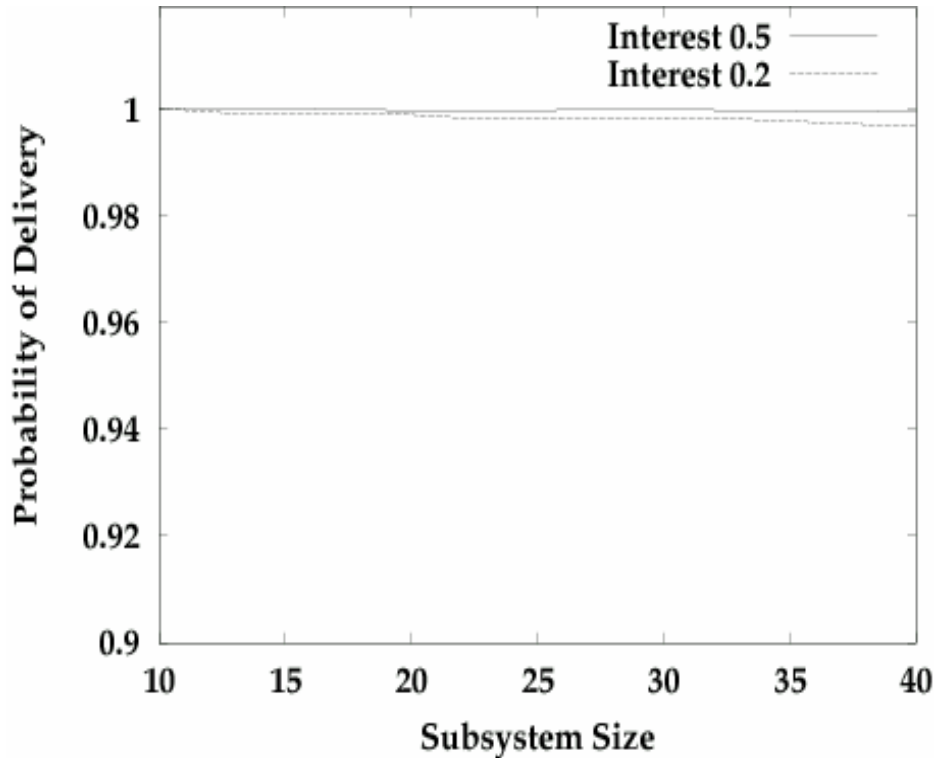
- ▶ Number of processes that a given process knows
 - $O(\ln n)$
- ▶ $I(t)$: infected entities at time t [Diekmann&Heesterbeek'00]
 - $dl/dt = F/n I(n - I)$, and hence $I = n / (1 + ne^{-Ft})$
 - Limited number of hops
- ▶ „Worst“ case: broadcast
 - Same „time“ to complete as when every process knows every other process: $O(\ln n)$
 - Same message complexity: $O(n \ln n)$
- ▶ Multicast
 - Probability ω of **pmdelivering** a message to an *interested* process
 - Comes very close to 1
 - Small reception probability for non-interested processes

Delivery and Unvoluntary Delivery

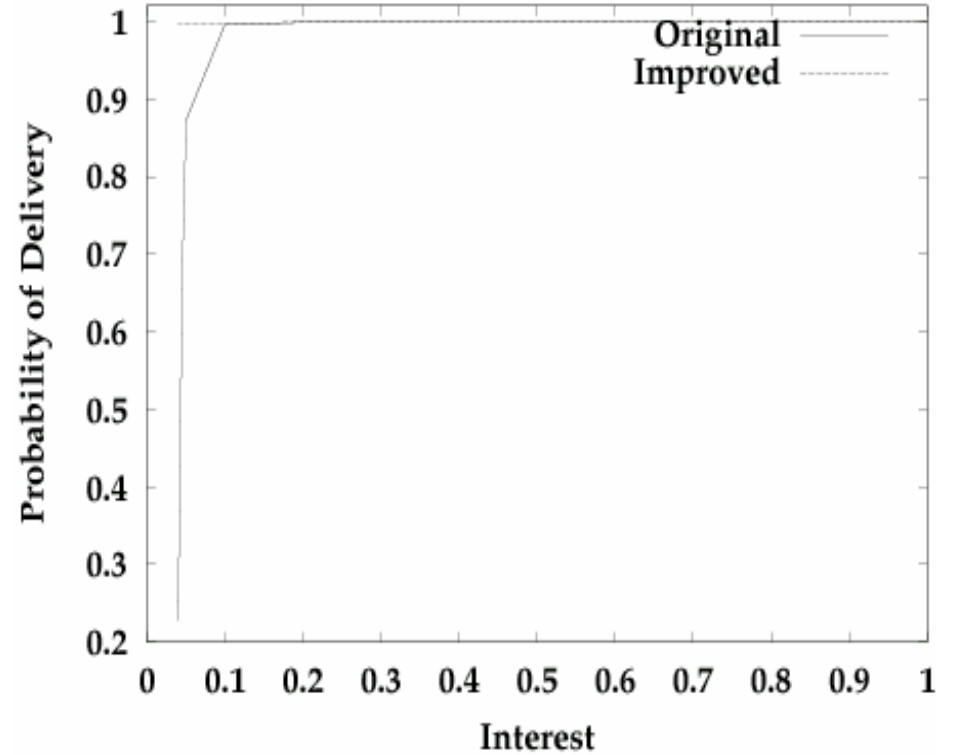


10000 processes, 3 levels, redundancy and fanout of 3

Scalability and Tuning



3 levels, redundancy 4, fanout 3



≈10000 processes, 3 levels,
redundancy and fanout of 3

Total Order

▶ **Roots**

- Delegates of top level

▶ **Root group**

- Roots of same level-1 subdivision

▶ **Roots order messages deterministically**

- Based on timestamps and broadcast rate, cf. [Aguilera&Strom'00]
 - $H=H(\text{time of last msg ordered from bs, rates of bs, max of bs rates})$
 - Proven optimal for memory-less Poisson processes
- Roots of same root group generate **identical** sequence numbers
 - Agree on sets of broadcasters
- Roots of distinct root groups generate **compatible** sequence numbers
 - Sets of broadcasters may overlap
 - $H=H(\text{time of last msg ordered from bs, rate of bs, absolute max rate})$

Broadcasters

- ▶ Reliable Broadcast to *individual* root groups
- ▶ Broadcaster failure/lag
 - Exclude from root group
- ▶ Broadcaster (re-)joining
 - Avoid considering any messages which would have been ordered prior to any already ordered messages

Conclusions

▶ „Natural“ tradeoffs

- Increasing the number of levels
 - Reduces the membership knowledge each process has
 - Increases the average filtering load, etc.
- Approximations for the number of rounds at each level
 - Small vs large fractions of interested processes

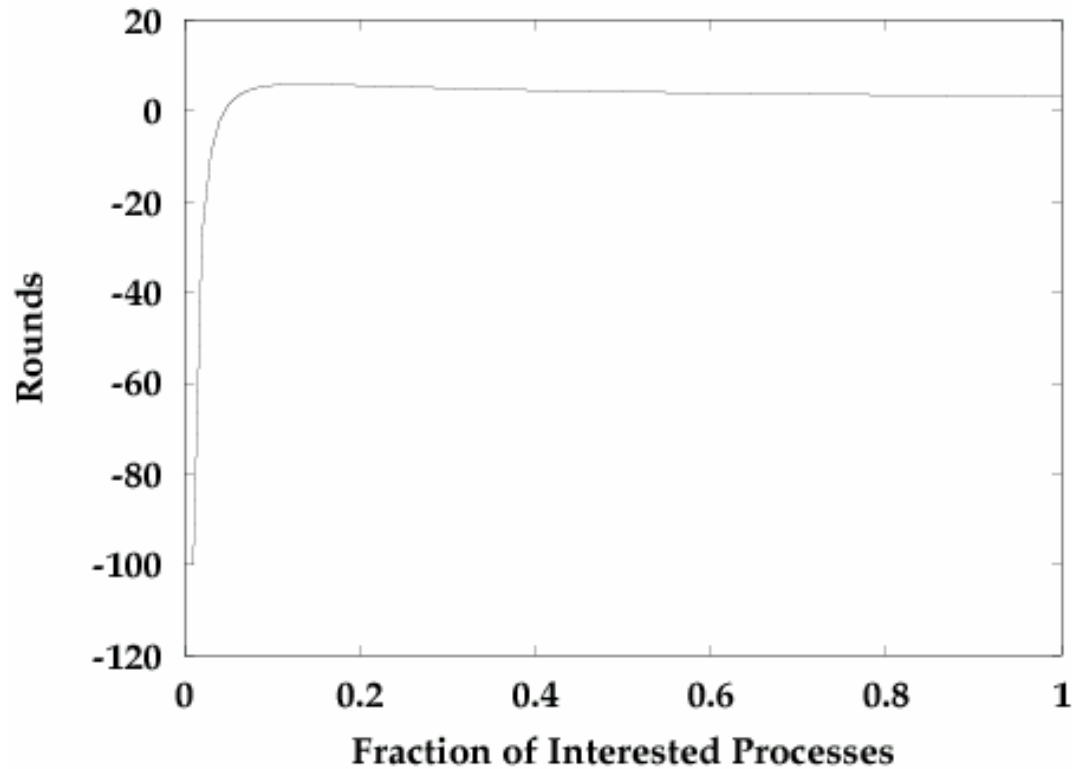
▶ Root processes

- Require more computing power
 - Use approximate matching to reduce filtering load
- Require more memory
 - Hash functions to compact space for „interests“

Questions



Gossip Round Approximation



1 level, 100 processes, fanout 2