Lightweight Probabilistic Broadcast

P. Th. Eugster [1], R. Guerraoui [1], S. B. Handurukande [1], A.-M. Kermarrec [2], P. Kouznetsov [1]

[1] Swiss Federal Institute of Technology, Lausanne
[2] Microsoft Research Cambridge, UK

Roadmap

- **Context**
- **Background**
- ***lpbcast***
- **Analysis**
- **Practical Results**
- **Optimizations/Future Work**
- **Conclusions**

Context

- **DACE middleware platform**
  - Distributed Asynchronous Computing Environment
  - Targeted at large scale asynchronous systems
- **Event-based interaction**
  - Publish/subscribe paradigm
  - Basic subscription criterion: types
- **Implemented as a « pure » library**
  - Perfectly distributed setting
  - No centralized event brokers etc.
  - Peer-to-peer computing
  - Different primitives for different QoS requirement

Background

- **« Traditional » algorithms**
  - *Reliable Broadcast* [HT93]
  - Strong reliability
  - Scale badly
- **Network-level protocols**
  - Scale better
  - Best-effort
  - E.g., *RMTP* (sender-reliable), *LBRM* (receiver-reliable): ack flow
- **Peer-based protocols**
  - Every process has same « role », can handle retransmission requests
  - E.g., *SRM*: peer-based, but re-broadcasting

Gossip-based (probabilistic) algorithms

- Not deterministic
  - No acks/nacks
- There is a probability of (1-*x*) that *all* processes deliver a given message
- And/or there is a probability of (1-*y*) for *any* given process to deliver a given message
- Ideally, *x* and/or *y* are quantifiable and << 1

- **Scalability**
  - Every process sends a limited number of messages
- **Reliability**
  - Every process receives copies of same message from different processes
- **Parameters**
  - *Period T* : each process period. gossips
  - *Fanout F* : at each gossip *round*, a process gossips to several processes
  - *Hops/Forwards* : same information is forwarded a limited number of times in total, or by same process
  - Adjusted to satisfy scalability and reliability (*x*, *y)* requirements

- **Variants**
  - *Push*, *pull*, *anti-entropy* [Demers et al.87]
  - Propagation of payload itself
  - Or identifiers (explicit retransmission requests)
  - E.g., *pbcast* (*Bimodal Multicast*) [Birman et al.99], *rpbcast* [SS00]
- **Usually based on « complete » views**
  - Though only weak consistency
  - Costly in terms of
    - Memory resource consumption
    - Message exchanges

- **Scalability**
  - Every process knows only a limited subset of the system
- **Reliability**
  - Every process is known by several other processes
- **Deterministic approaches**
  - Hierarchy, possibly based on network topology, e.g., [LM99]
  - Analysis?
- **Probabilistic approach**
  - *Period* : each process gossips periodically an *exerpt* of its view
  - *Fanout* : at each gossip *round*, a process gossips to several processes

lpbcast

- **Every process only knows *l* within *n* processes**
  - Probabilistic broadcast *and* membership
- **Gossip messages serve**
  - Membership information exchange
  - Transporting events
  - Event knowledge exchange
- **A gossip message carries**
  - A set of subscriptions (not nec. « new » ones)
  - A set of unsubscriptions
  - A set of events received since the last outgoing gossip
  - A digest of received events (ids)

- **Data structures**
  - Events
  - Event ids
  - View (+ unsubscriptions)
- **Upon receiving a gossip message**
  - Deliver new events/update event ids
  - Add to event buffer/truncate buffer
  - Ask for retransmission
  - Remote unsubscribed processes from view/add to unsubs
  - Add new subscriptions to view/truncate view
- **When sending**
  - Add subset of events, event ids, view, unsubs

Analysis

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

  $p=(l/n)(F/l)(1-e)(1-f)$

  $=(F/n)(1-e)(1-f)$

  $q=1-p$

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

  $p_{i\,j} = B(n-i,j-i)\ (1-q^{i})^{j-i}(q^{i})^{n-j}$

- ***P(j infected at round r) = S $_{i\ ?\ j}$ P(i infected at round r-1) p$_{i\,j}$***
- **Throughput independent of *l***
  - Provided that views are <u>uniformly</u> distributed

Membership stability

- Probability of creation of a partition of size $i > l$

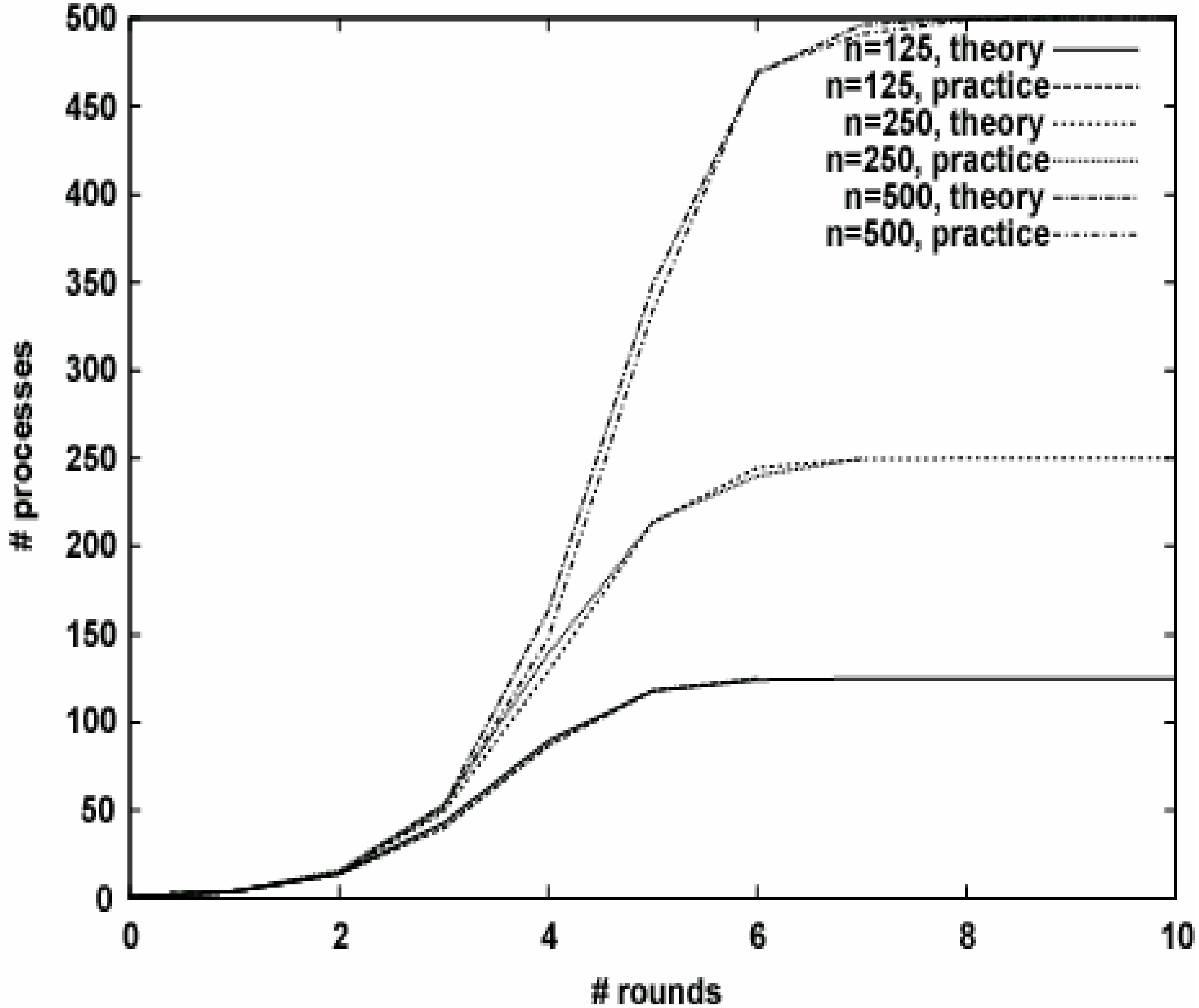
  $B(n,i) \, (B(i,l) / B(n,l))^{i} \, (B(n-i,l) / B(n,l))^{n-i}$

- 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

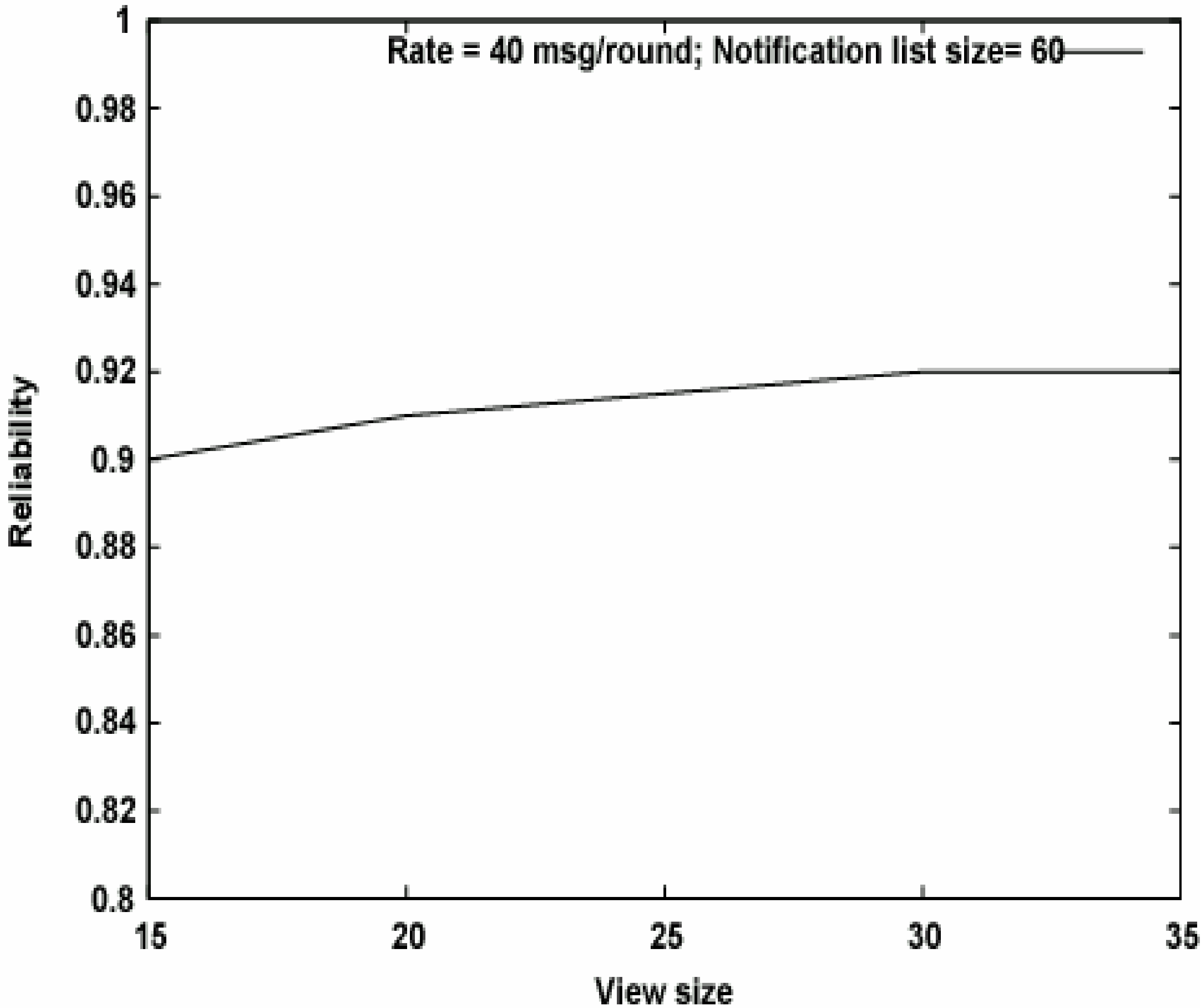
Practical Results

- **Simulation/measurements**
- **Distribution of views**
  - Throughput *does* depend (very little) on *l*
  - Dependency
    - Gossiping process adds parts of its view
    - Receiving process mixes with its view and forwards
    - Redundant messages
  - Reliability
    - Throughput decreases, and buffers are limited
    - Probability that a given notification is removed from all buffers before being delivered by all increases

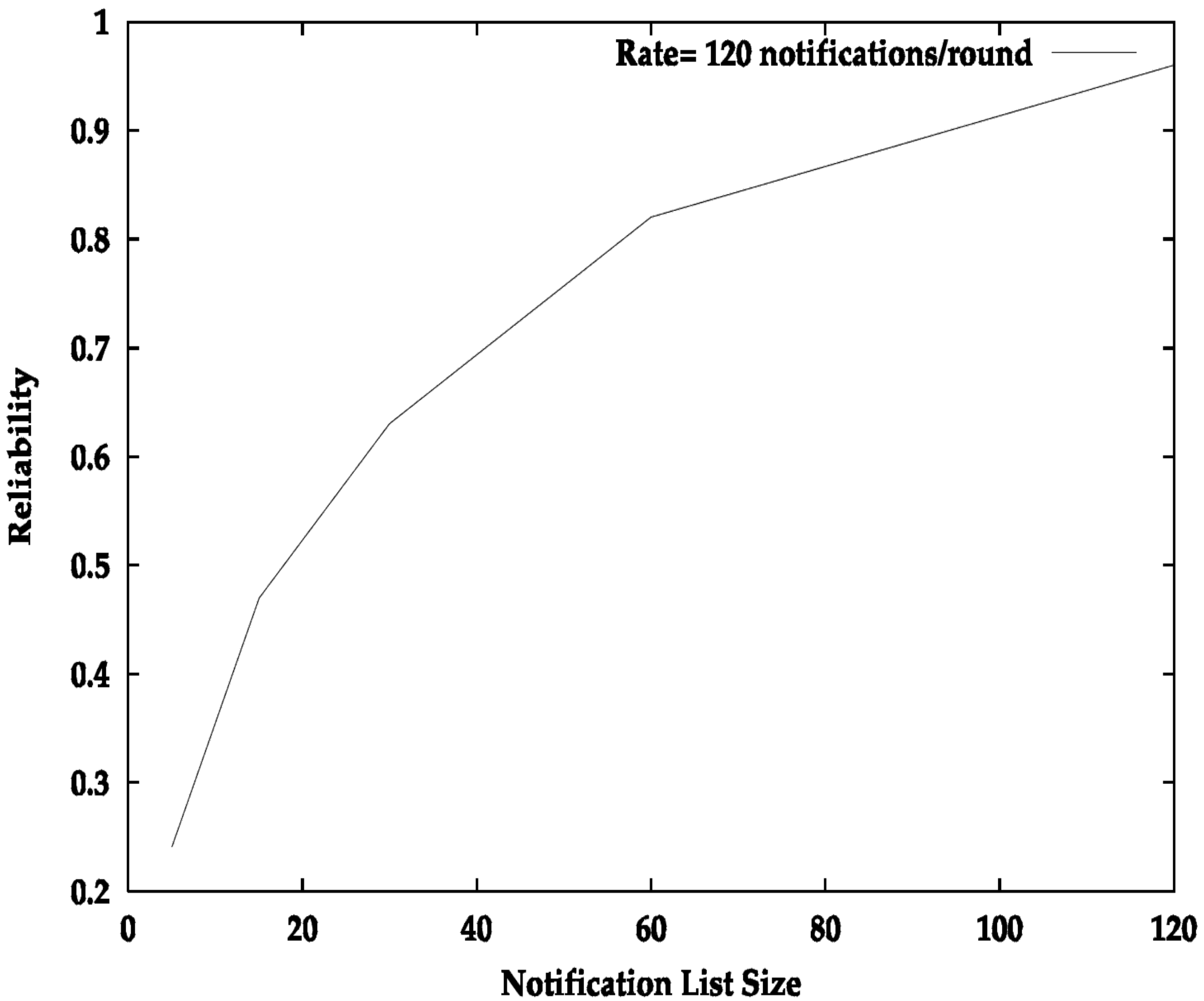
- Analysis vs simulation
- Fanout 3
- 1 msg injected
- Varying system size



- 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



Optimizations / Future Work

- **Towards « perfect » views**
  - Remove dependencies
    - By adding weights to subscriptions
    - By reducing period for membership gossiping
- **Garbage collection**
  - Remove old messages first
- **Add rapid dissemination phase**
  - à-la *pbcast*
  - Increase throughput
  - Use gossip messages solely for digests (ids)

- **Optimal Value for *l* ?**
- **Expected value for l_{eff}**
  - Number of processes which know a given process
  - Obviously *l*
- **Variance of l_{eff}**
  - $l\ (1-l\ /n)$
  - Good for small, and big *l*
  - Maximum (worst) for $n\ /2$
- **Must be at least *F***
  - $Log(n) < n/2$

Conclusions

- **Preciser analysis would also depend on**
  - Concrete compositions of individual views
  - Sizes of buffers for events, ids, ...
- **Membership can be separated from broadcast**
- **Weaknesses**
  - Does not exploit locality
  - Joining/leaving (failure detection)
- **Deterministic schemes (hierarchy)**
  - Based on (network) topology knowledge
  - Better in the case of genuine multicast (filtering)