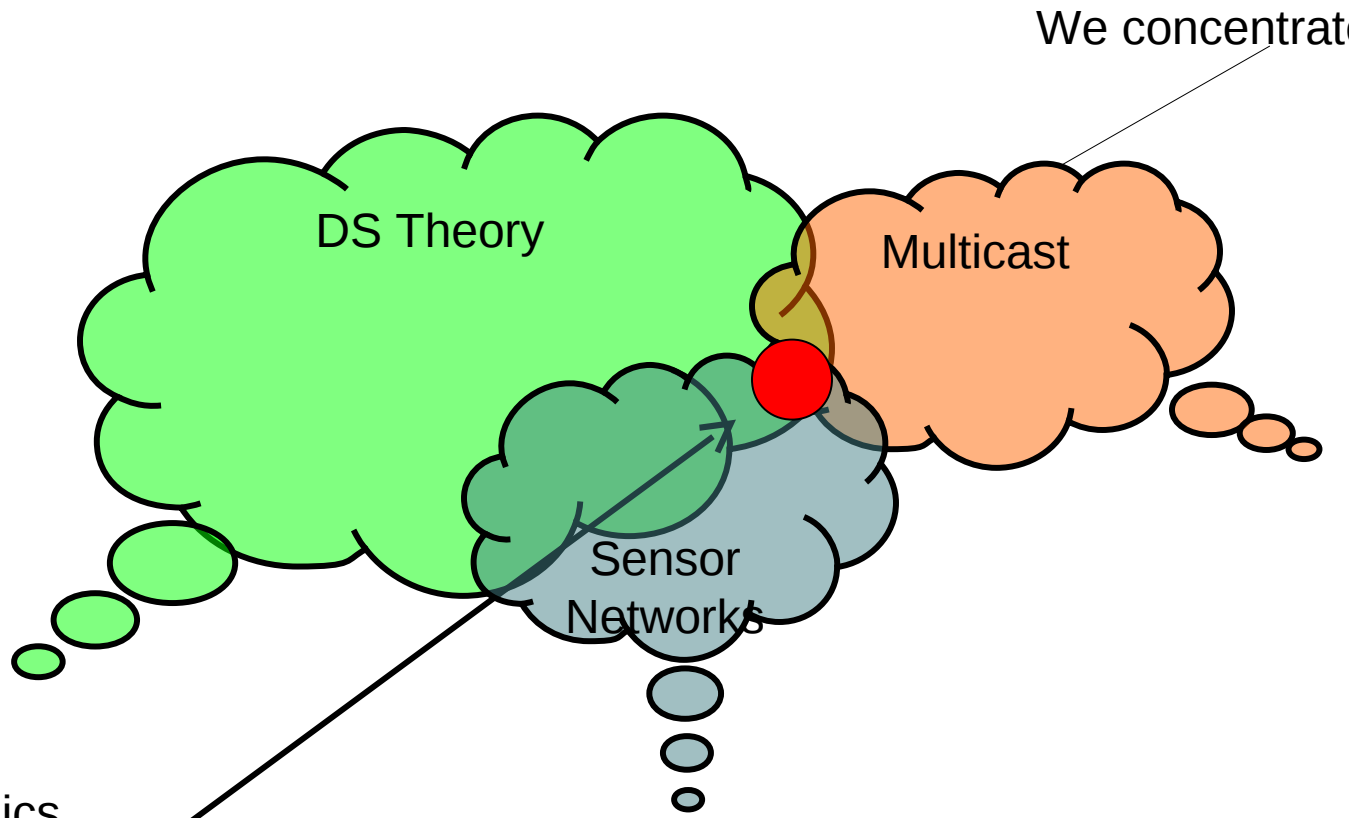
Epidemic Algorithms

Presented by: Kurchi Subhra Hazra
CS 525
University of Illinois at Urbana - Champaign

Interesting: Area Overlaps



From Professor Gupta's Slides of 1/19/2010

Recap! Gossip vs Flooding

- Flooding has advantages
  - Universal Coverage
  - Minimal State Information
- Goal of Gossip?
  - Reduce the number of redundant transmissions while retaining the above advantages
- Nature of Gossip?
  - Probabilistic
  - Takes a localized decision but results in a global state.
  - Lightweight
  - Fault Tolerant
- Why use the term epidemics for gossip?
  - Gossip spreads data in a manner similar to spread of virus in living beings.

Today's Agenda

- Epidemic Algorithms for Replicated Database Maintenance – Demers et al.

  (Talks of several epidemic algorithms)

- Bimodal Multicast - K.P. Birman et al

  (Epidemics in Multicast)

Epidemic Algorithm for Replicated Database Maintenance

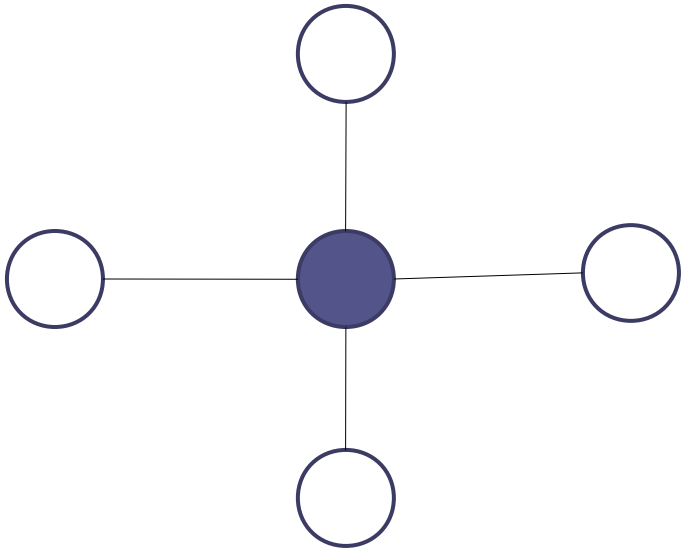
A. Demers et al.
Xerox Palo Alto Research Center

PODC 1987

Motivation

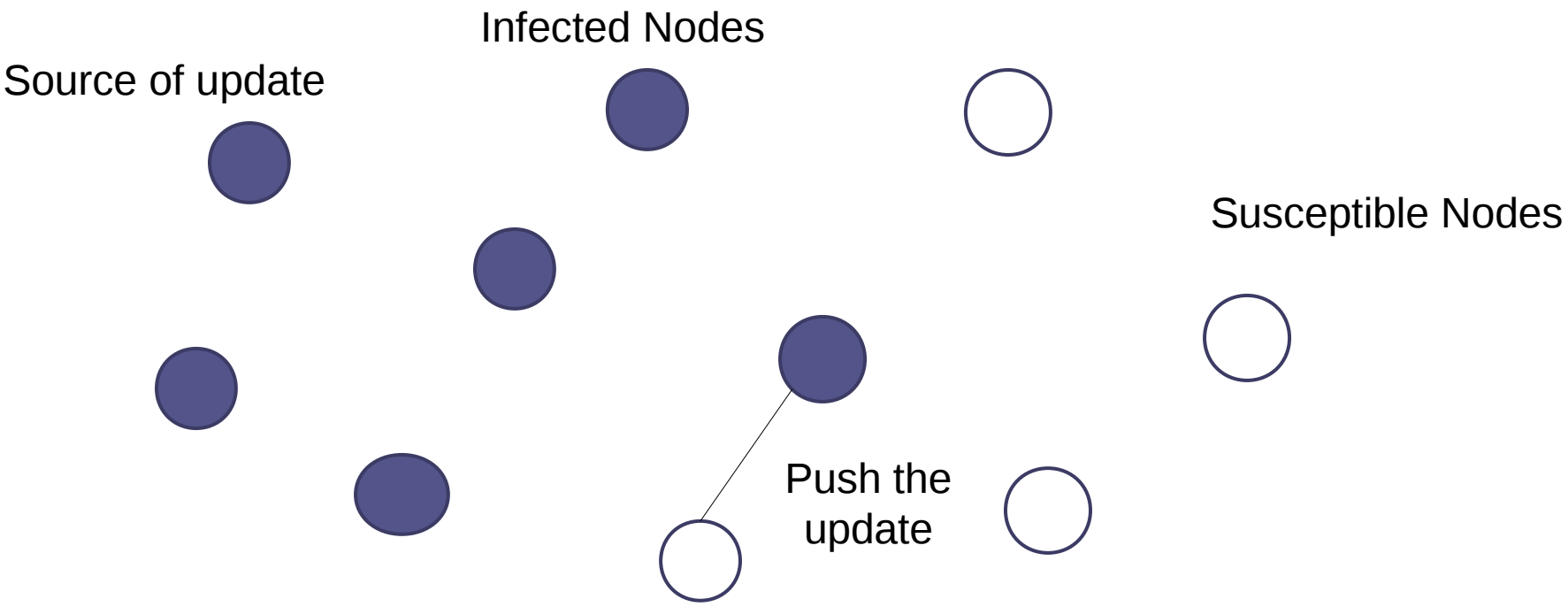
- Origins in study of Clearinghouse Servers on Xerox Corporate Internet (CIN)
- Updates needed to be distributed to drive the database toward consistency.
- Strategies for Spreading Updates
  - Direct Mail – Each new update is immediately mailed to other sites
  - Anti Entropy – Exchange database contents with another random site
  - Rumor-mongering – A site with “hot rumour” will periodically infect other sites.

Direct Mail

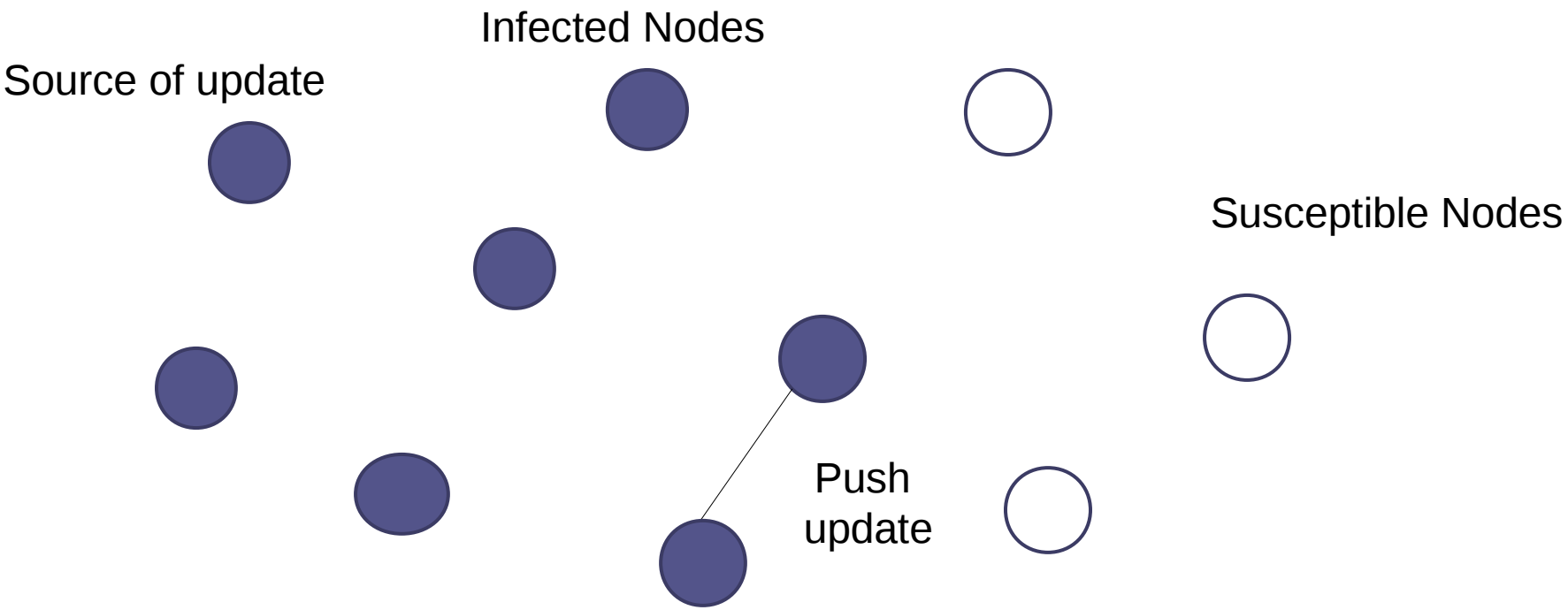


Problems?
-> Message losses
-> Need accurate knowledge of network
-> Bottleneck at originating site

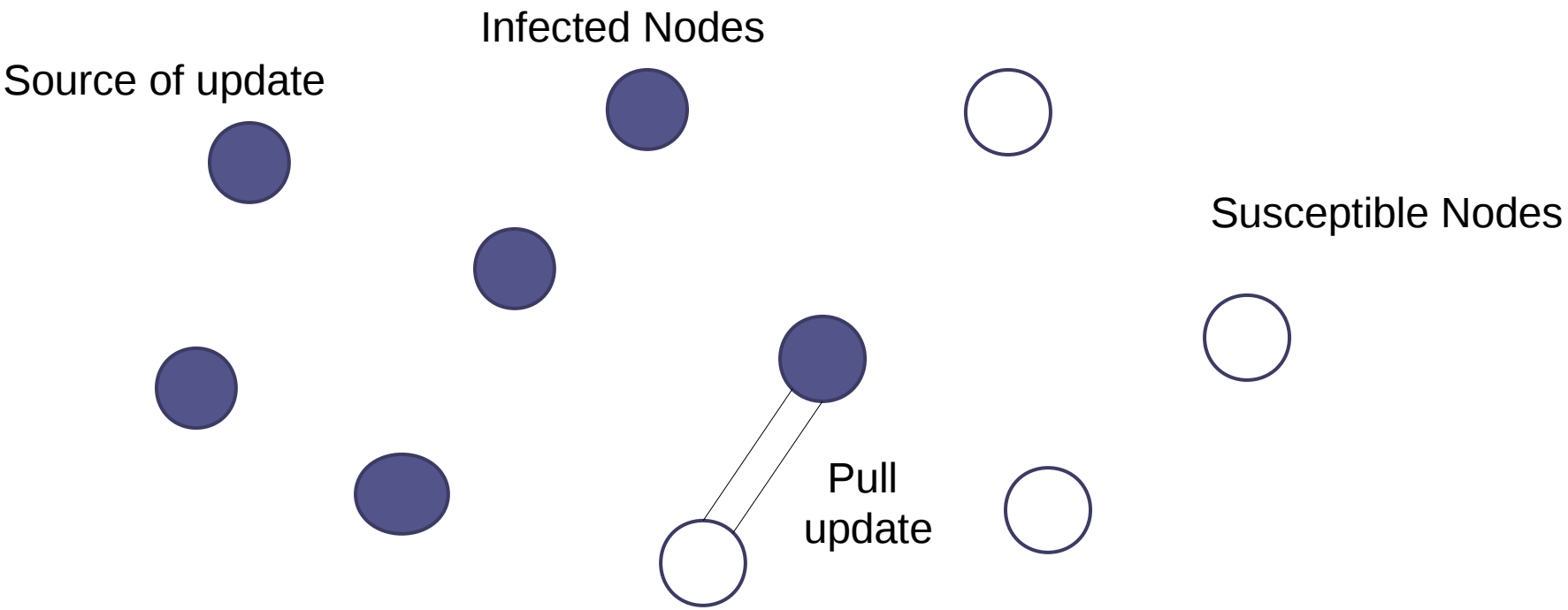
Anti Entropy - Push



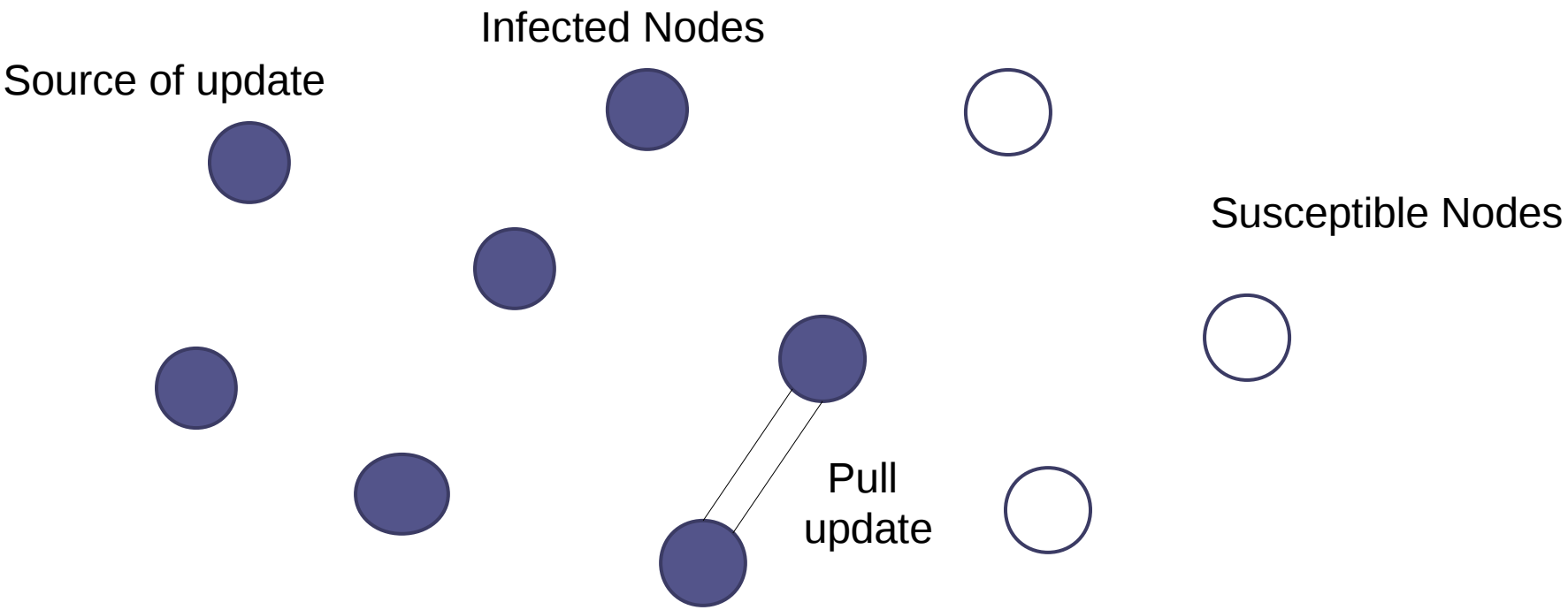
Anti Entropy - Push



Anti Entropy - Pull



Anti Entropy - Pull



Push vs. Pull

- Pull

Probability that site s1 was susceptible in i^{th} cycle $= p_i$

Probability that the s2 that contacted s1 in $i+1^{th}$ cycle was susceptible in i^{th} cycle $= p_i$

Probability of susceptibility of s1 in i+1th cycle $= p_i^2$

- Push

Probability that site s1 was susceptible in i^{th} cycle $= p_i$

Probability that s2, an infectious site does not contact s1 in $i+1^{th}$ cycle $= 1-(1/n)$

No. of infected sites in i^{th} cycle $= n(1-p_i)$

Probability that no infectious site chose to contact me in $(i+1)^{th}$ cycle $= (1-1/n)^{n(1-pi)}$

Probability of susceptibility of s1 in $i+1^{th}$ cycle $= p_i\,(1-1/n)^{n(1-pi)}$

Push vs. Pull

	Pull	Push
Probability of Susceptibility in $(i+1)^{th}$ cycle	$(p_i)^2$	$p_i \quad (1- 1/n)^{n(1-p_i)}$
Convergence	Fast	Slow
Traffic overhead	Less	More
Good for	Numerous independent updates	Quiescent Database

But what's the problem with both?

Anti Entropy - Some Optimizations

- Use checksums

  Sites maintain checksums of their database and exchange these. When checksums disagree, they compare full database

- Recent Update List

  Maintain a list of updates whose age is less than Ʊ. Exchange list first.

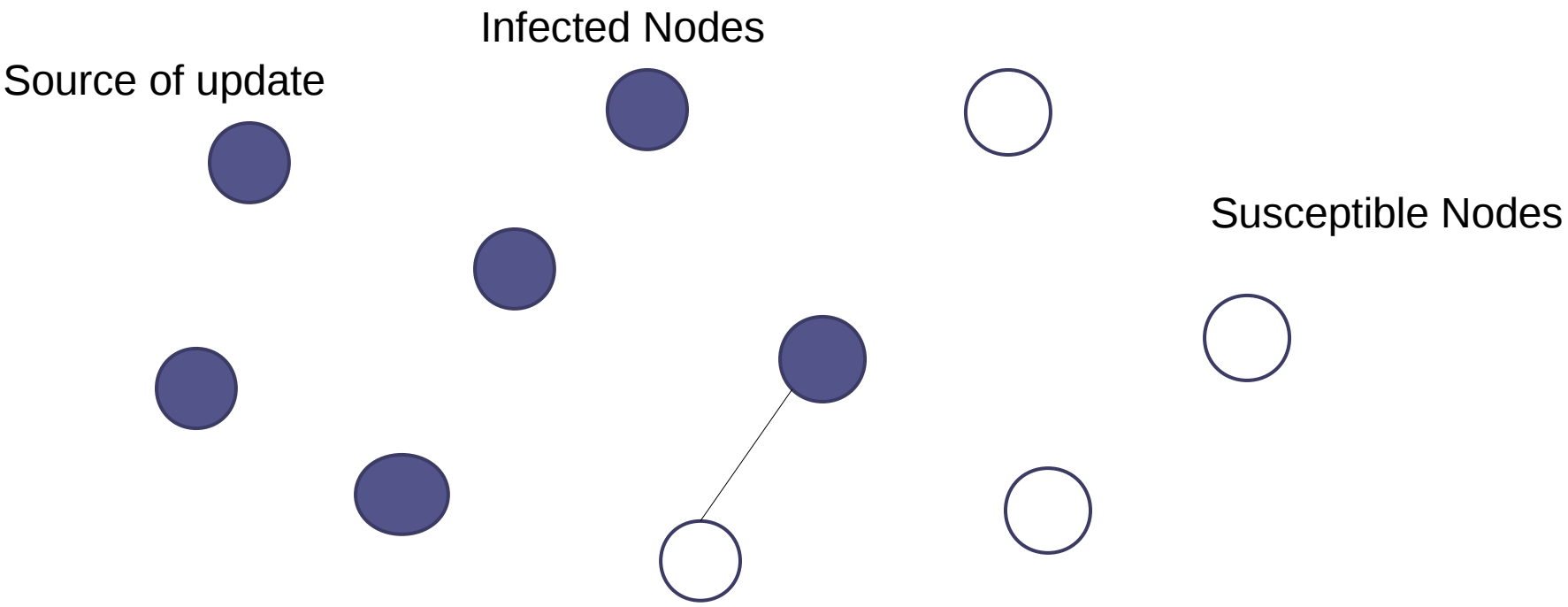
- Inverted Index

  Sites maintain inverted index of database by timestamp. Quite costly!

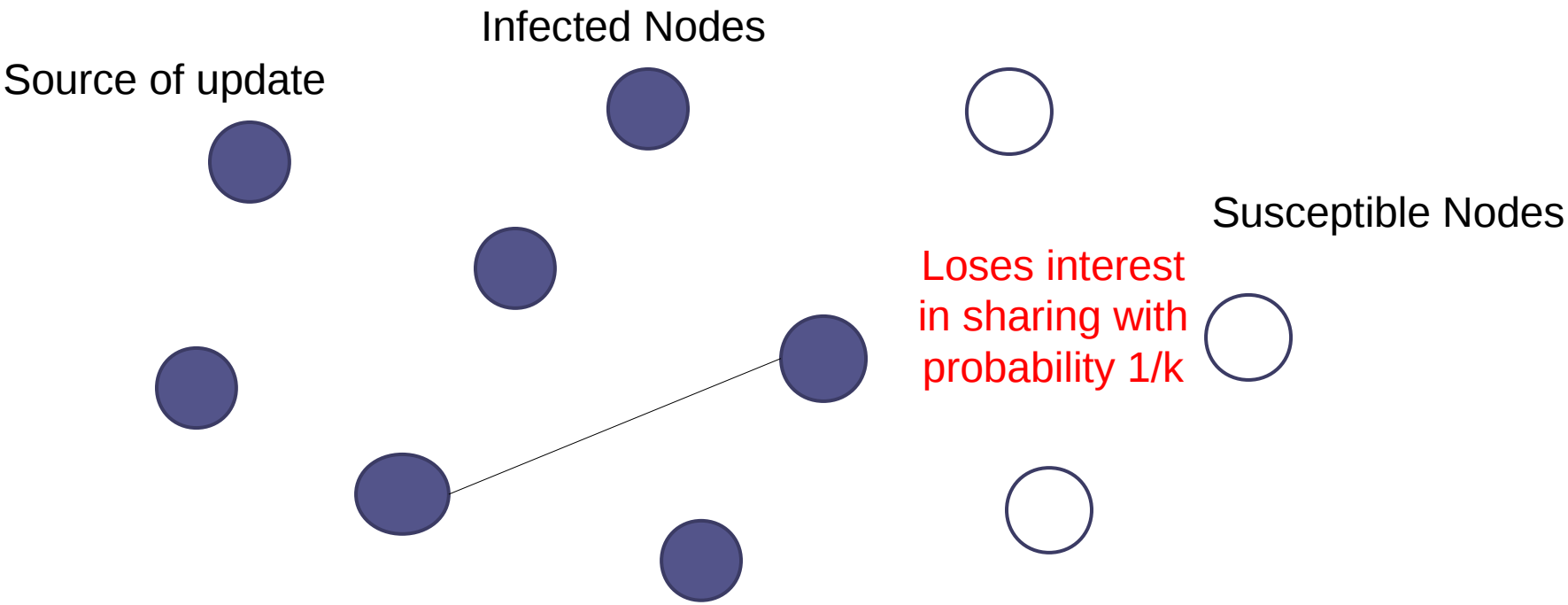
Complex Epidemics!

- Anti Entropy was “simple”. Why?
  - All nodes are either susceptive or infective
- What’s new in Complex Epidemics?
  - “Removed” nodes
  - Example: Rumor spreading
- So how does the algorithm work?

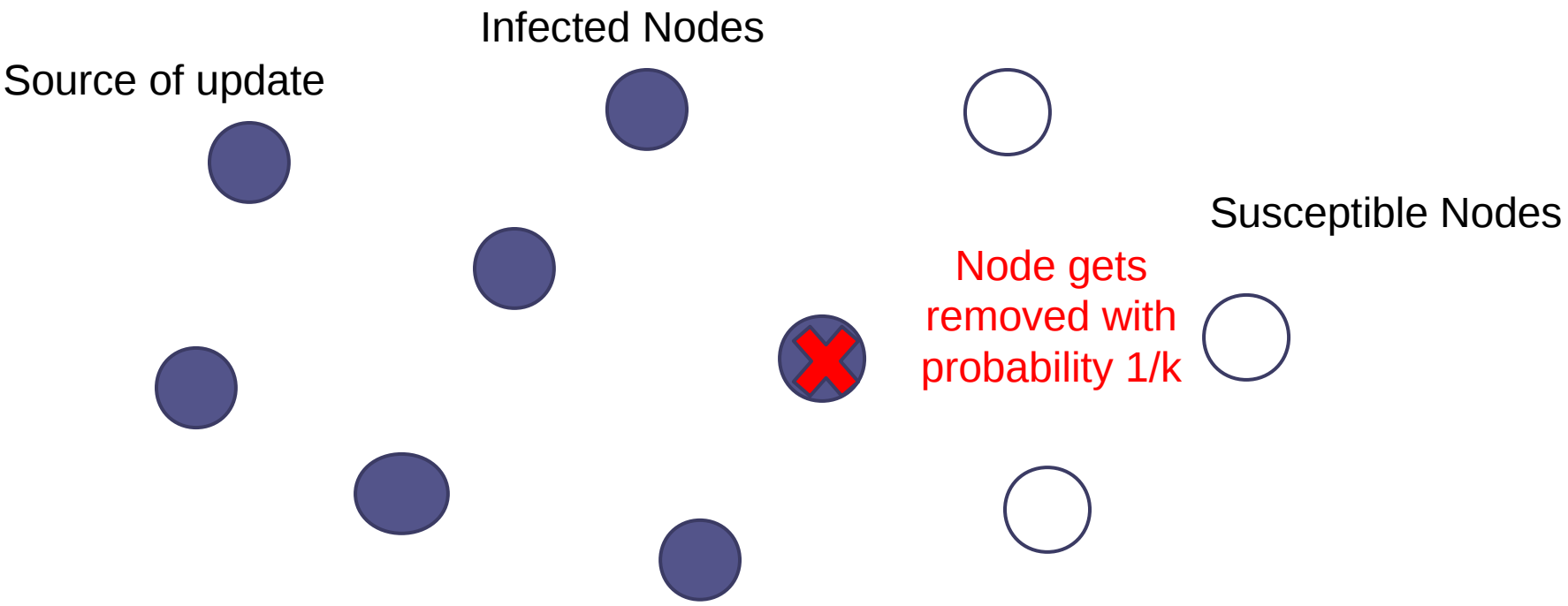
Rumor Spreading



Rumor Spreading



Rumor Spreading



What if k=1?

Increase in k ensures that almost all nodes hear the rumor.

Variations of Rumor Spreading

- Feedback – What we just saw!
- Blind – Does not care if recipient is infected
- Coin – Uses a probability value as in previous case
- Counter – Loose interest after k unnecessary contacts
- Counter + Blind = ?
- Push and Pull is valid here too!

Performance metrics

- Residue – Number of nodes that do not receive updates when no infected nodes remain
- Traffic - In terms of number of messages sent from a typical site

$$m = \frac{\text{Total update traffic}}{\text{No. of sites}}$$

- Delay/ Convergence –
- t_{avg}
- t_{last}

Some Results

Table 1. Performance of an epidemic on 1000 sites using response and counters.

Counter k	Residue s	Traffic m	Convergence t_{ave}	t_{last}
1	0.176	1.74	11.0	16.8
2	0.037	3.30	12.1	16.9
3	0.011	4.53	12.5	17.4
4	0.0036	5.64	12.7	17.5
5	0.0012	6.68	12.8	17.7

Using Push

Table 3. Performance of a pull epidemic on 1000 sites using response and counters†.

Counter k	Residue s	Traffic m	Convergence t_{ave}	t_{last}
1	3.1×10^{-2}	2.70	9.97	17.63
2	5.8×10^{-4}	4.49	10.07	15.39
3	4.0×10^{-6}	6.09	10.08	14.00

Using Pull

Using response and counters improve delay

Some optimizations

- Minimization – Compare counters of both parties. Increment smaller counter value.
- Connection Limit – Two sites may choose to contact one site in the same cycle. Push performs better than pull.
- Hunting – When a connection is rejected, choosing site can hunt for alternate sites.

Death Certificates

- Absence of an item does not spread with epidemics
- Resurrection happens instead!
- Solution: Death Certificates carrying timestamps
- When copies of deleted items meet death certificates, delete them

- Problem : When do we delete the certificate itself?

Dormant Death Certificates

- Two thresholds : τ_1 and τ_2
- Retain death certificates till τ_1
- Retain dormant death certificates at r retention sites till $\tau_1 + \tau_2$
- Anti Entropy – What if a deleted item was reinstated?
- Rumor Mongering

Some discussion points

- Which one is better?

Direct Mail, Anti Entropy, Rumor Mongering

- How about push-pull?

- Which neighbor to pick?

Should we have any preference here?

- When can Anti Entropy be better than Rumor Mongering?

Bimodal Multicast

By K.P. Birman et al

ACM Transactions on Computer Systems, 1999

Presented by:
Kurchi Subhra Hazra

Multicast?

Messages are sent from exactly one process to several processes on the network

Pros: Conserves Bandwidth, Need not know the identity of group members

Image Source: www.dkfactor.com

Reliable Multicast?

- Integrity – Every process delivers the original message sent out at most once
- Validity - A correct process that sends out message m will eventually deliver m
- Agreement - If one correct process delivers m, all other correct processes will eventually deliver m
- ->Overall Liveness of System

Two Kinds of "Reliable" Multicast Protocols

Type I

•Strong Atomicity Guarantees

• Limited scalability

• Unstable performance under stress

•Eg. Virtually Synchronous Protocols

Type II

•Best Effort Reliability

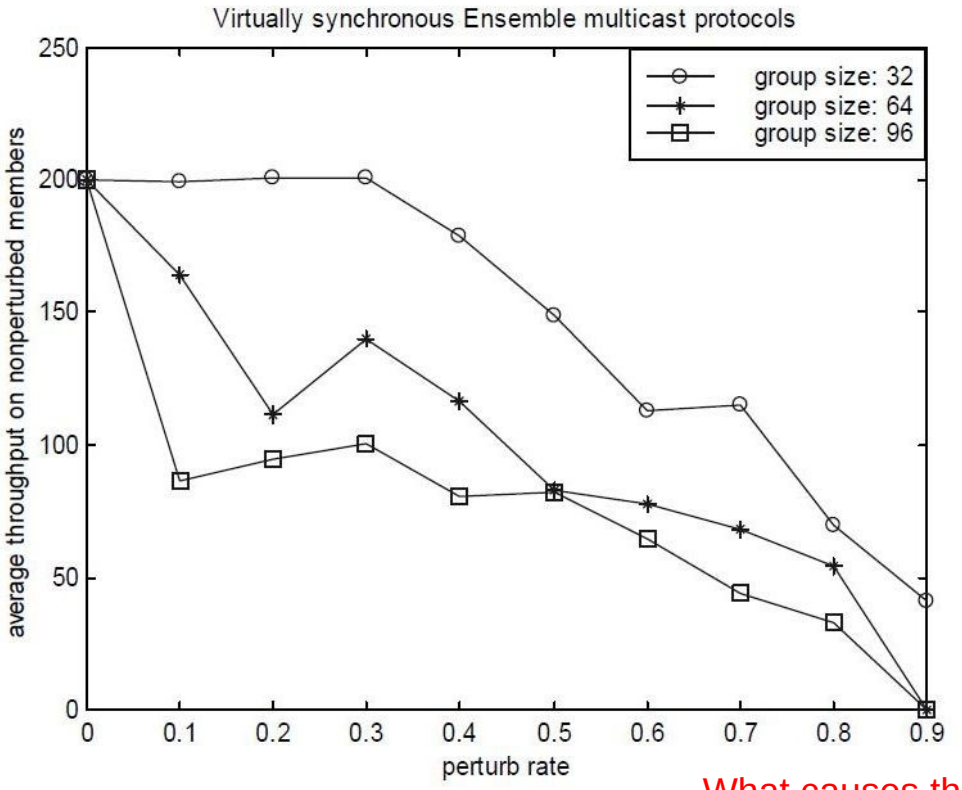
• Better Scalability

•Impossible to reason about system performance

•Eg. SRM, IP Multicast

Throughput collapse in Type 1

- 7 KB multicast messages at a rate of 200 messages/sec
- Single Member perturbed for the percentage of each second (as in x axis)



What causes this problem?

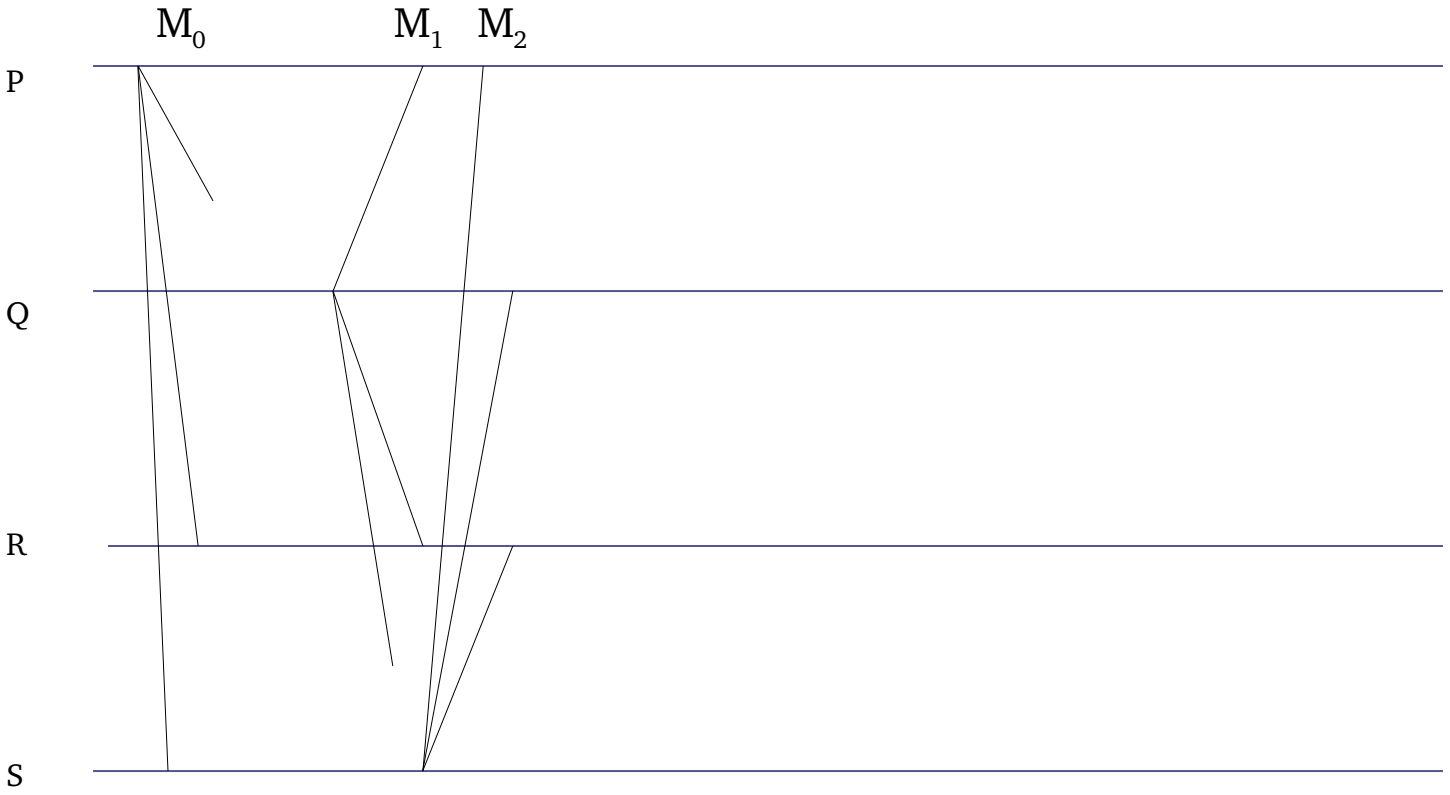
What is the problem?

- Limited Scalability
- Applications with notion of “freshness”
- Safety Critical Applications that cannot withstand unstable throughput

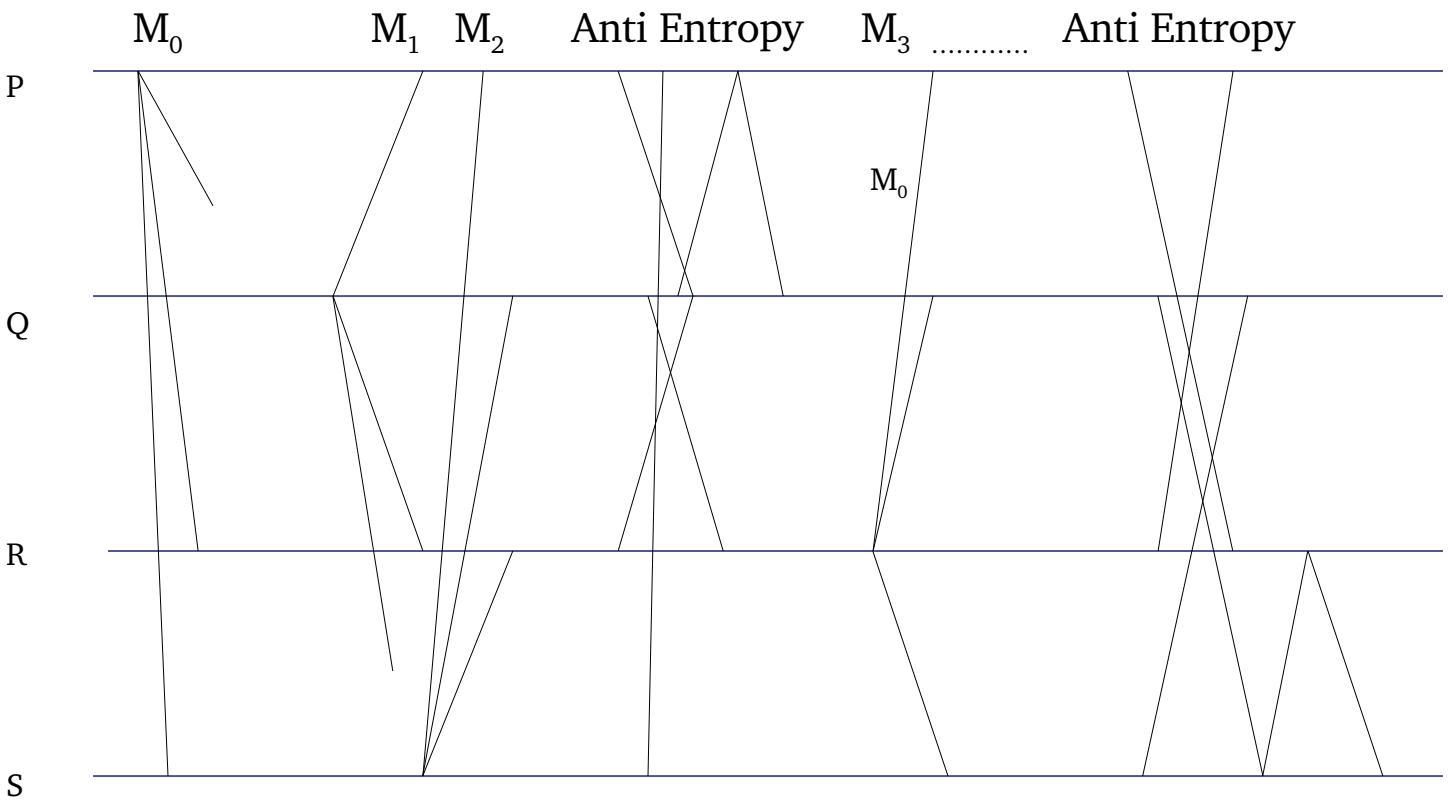
Bimodal Multicast or Probabilistic Broadcast(Pbcast)

- Two Sub-Protocols
- Redefines Atomicity – Almost all or Almost none
- Throughput stability – Expected variation in throughput can be characterized
- Scalability – Throughput variation grows slowly with log of group size

Bimodal Multicast – Optimistic Dissemination Protocol

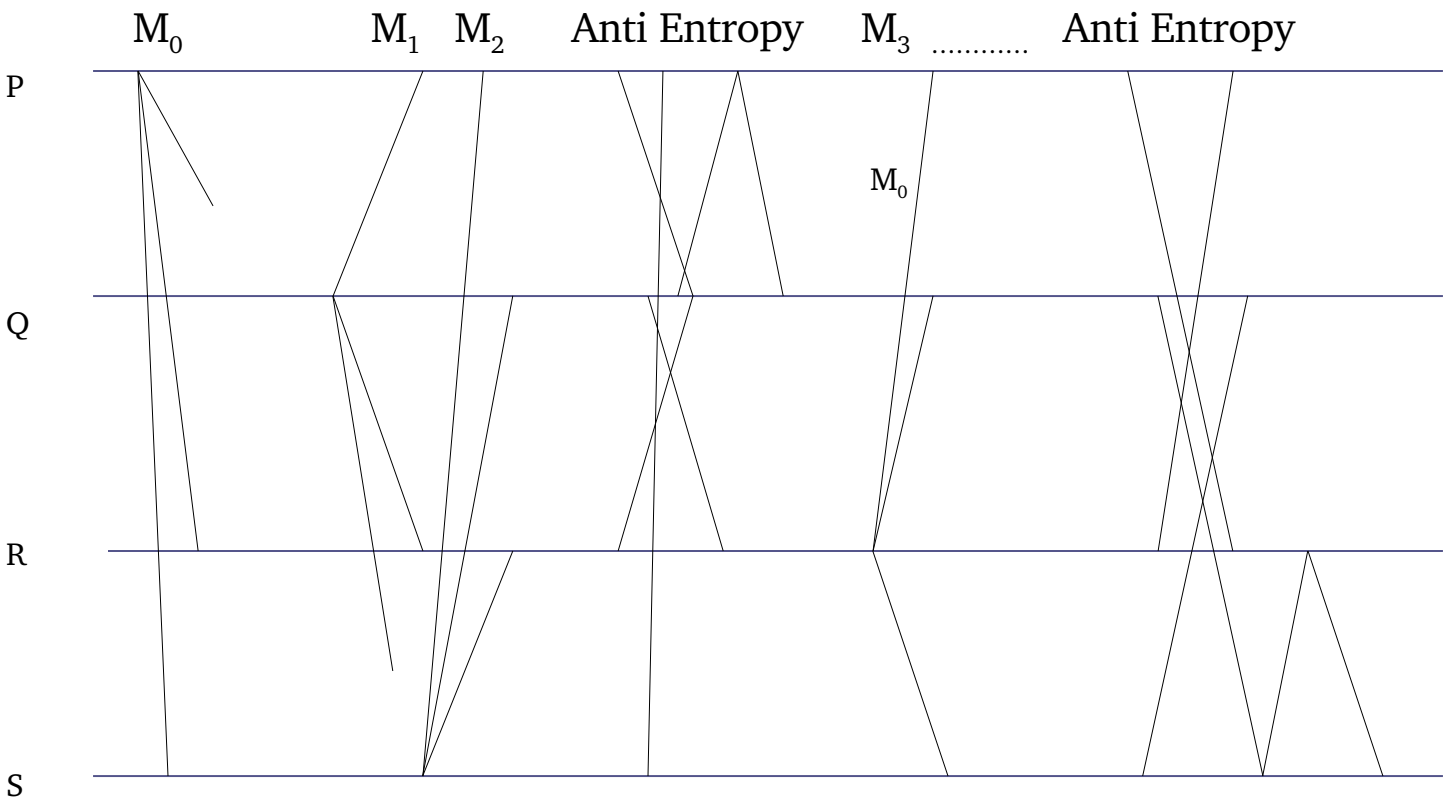


Bimodal Multicast – Anti-Entropy Protocol



Fanout = no. of rounds X no. of processes with which to gossip

Bimodal Multicast – Anti-Entropy Protocol

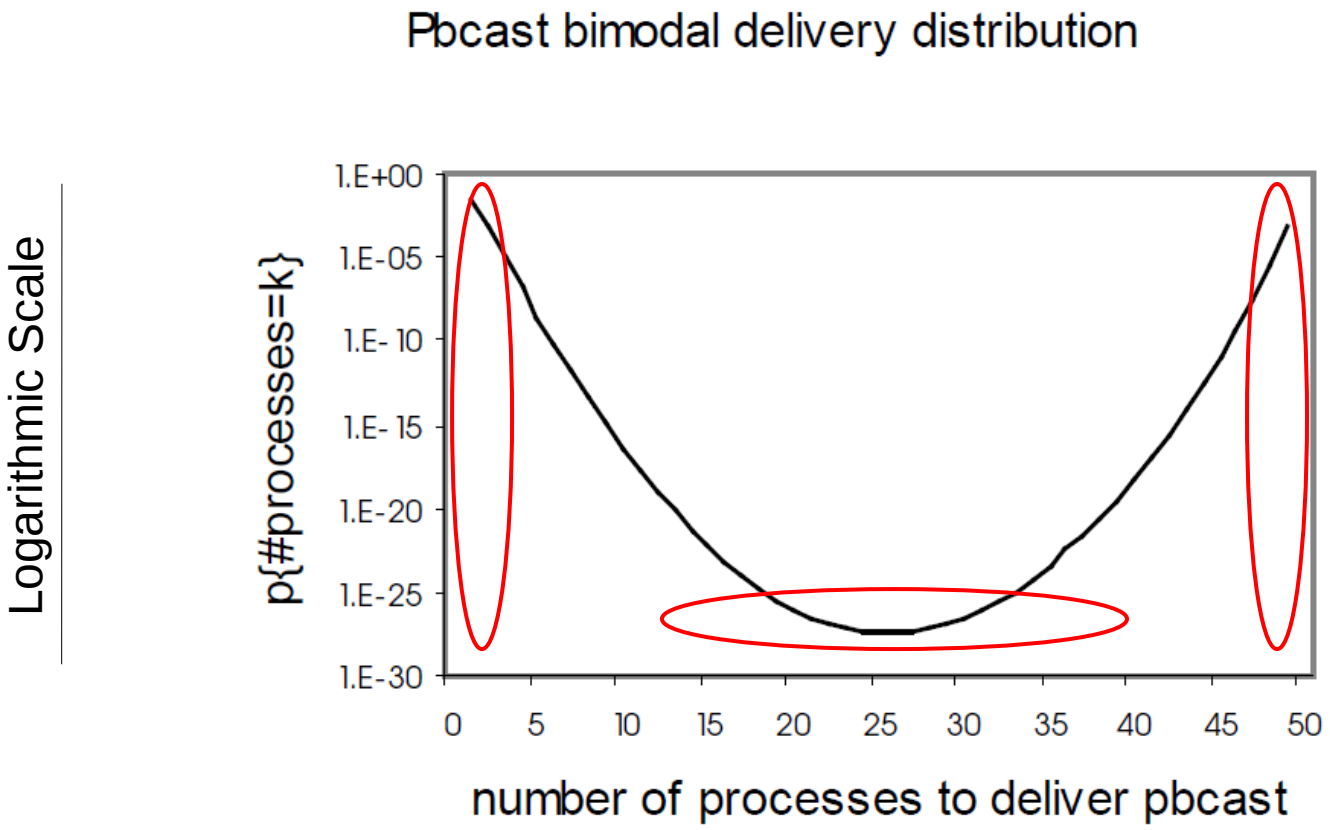


Fanout = no. of rounds X no. of processes with which to gossip

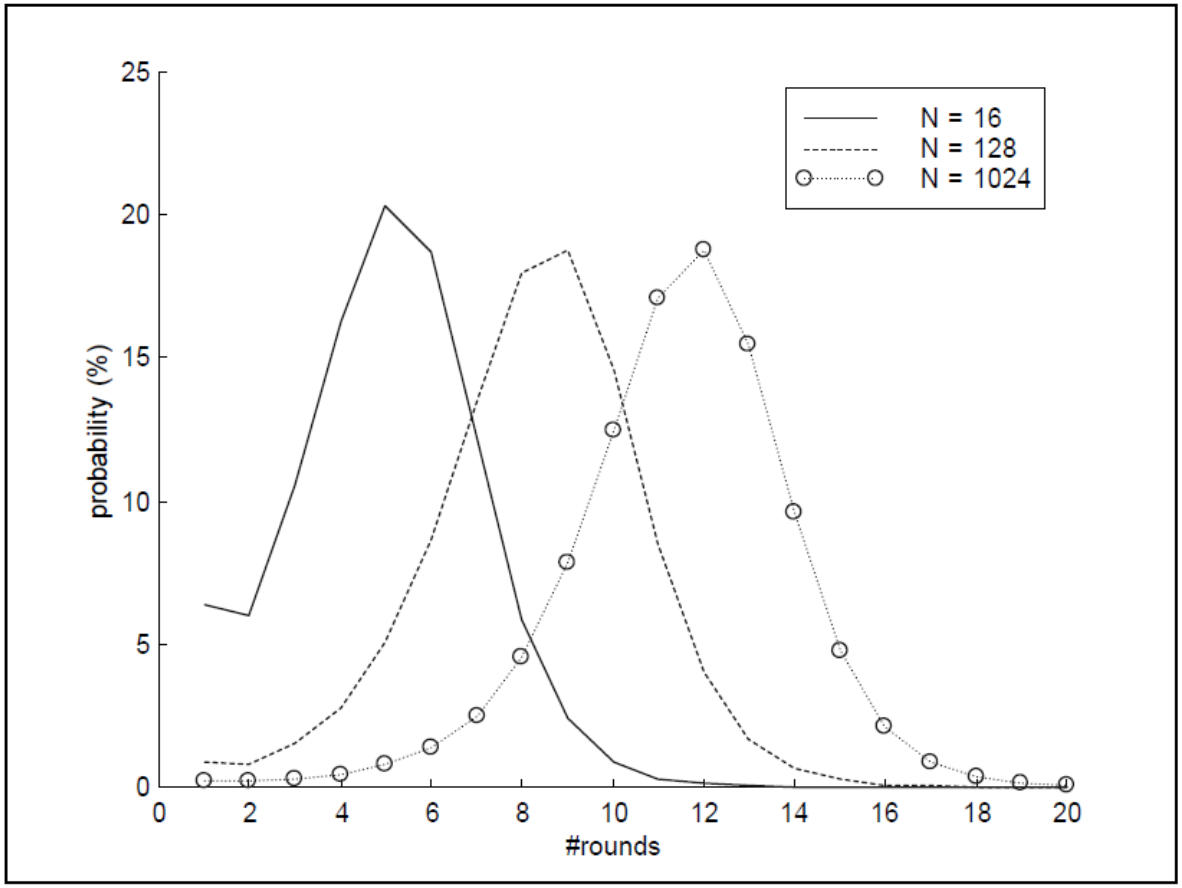
Optimizations

- Soft Failure Detection - Retransmission requests serviced only in the same round as solicitation (Failure)
- Round Retransmission Limit – Limit the no. of retransmissions by one process to spread overhead in space and time
- Most Recent First Retransmission – Maintains Freshness concept; Prevents a temporary faulty process from lagging behind
- Independent Numbering of Rounds – Makes garbage collection a local decision
- WAN and LAN gossip
- Multicast for some Retransmissions - If the same message is solicited twice

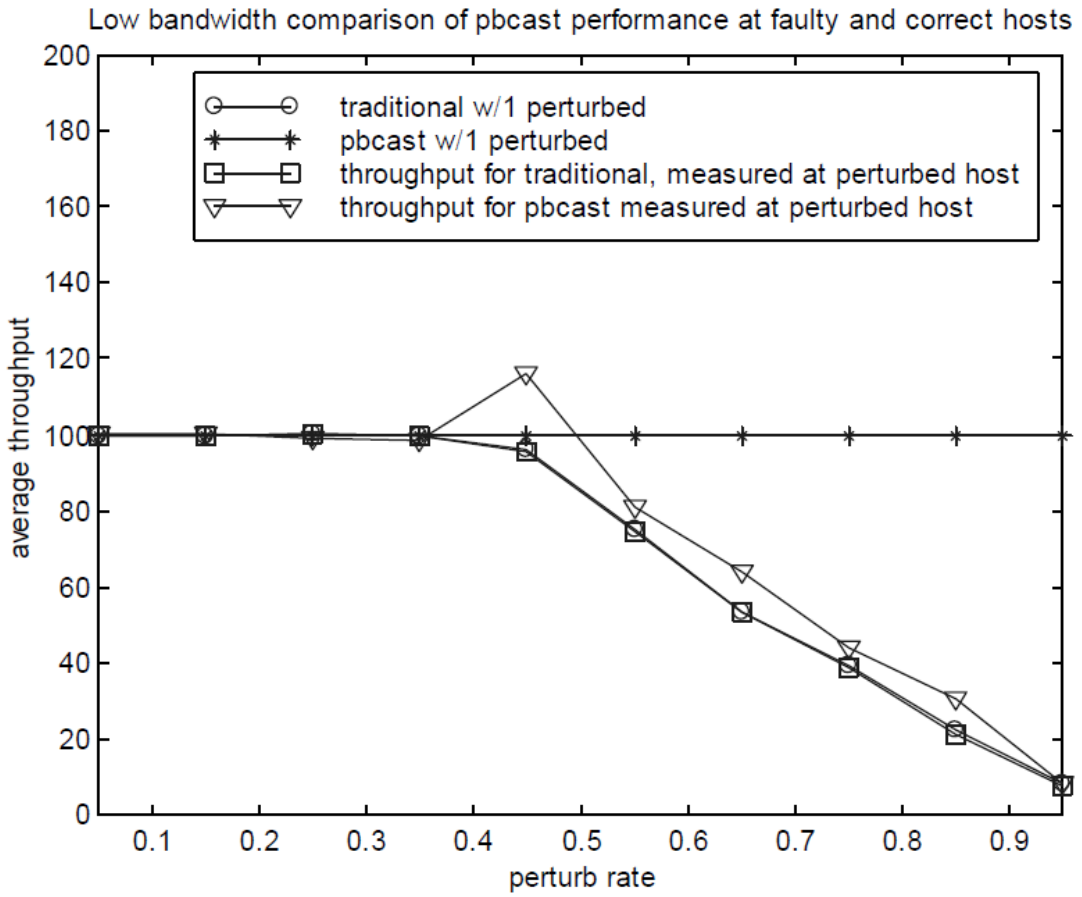
Why Bimodal?


Pbcast bimodal delivery distribution
Logarithmic Scale
p{#processes=k}
1.E+00
1.E-05
1.E- 10
1.E- 15
1.E-20
1.E-25
1.E-30
0
5
10
15
20
25
30
35
40
45
50
number of processes to deliver pbcast

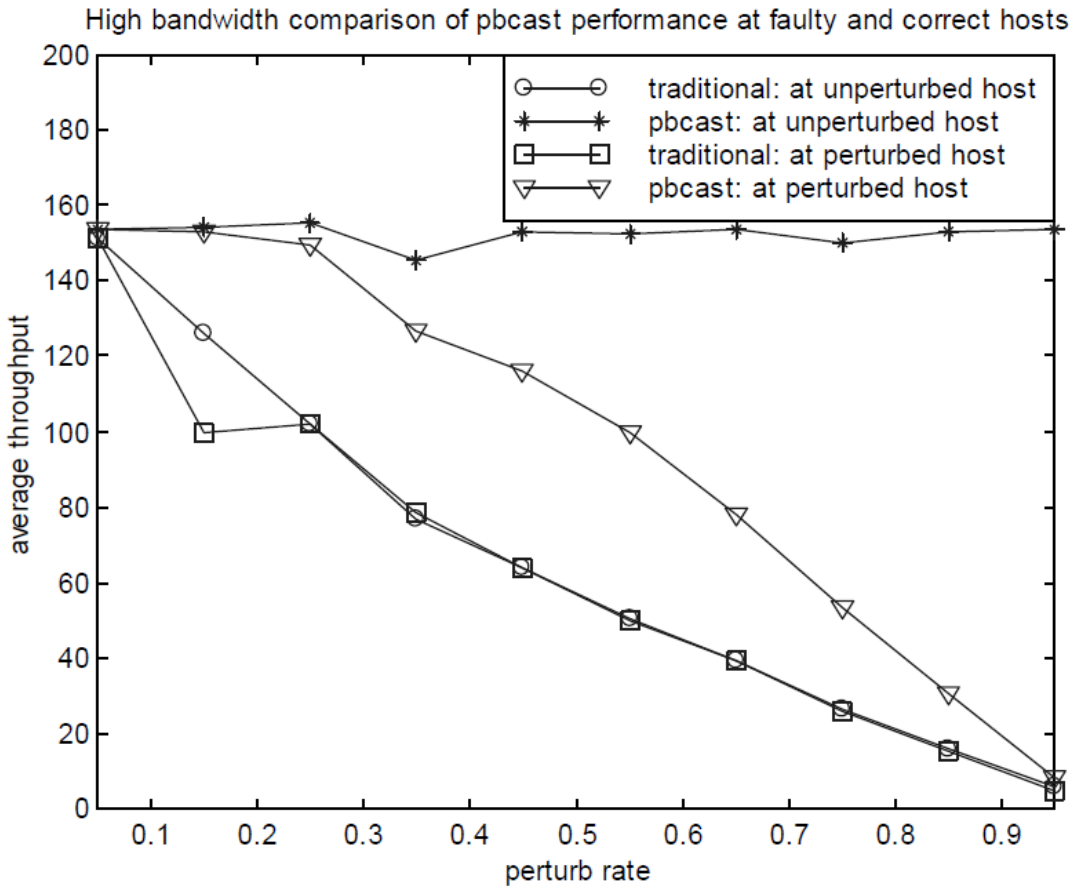
Latency


25
20
15
10
5
0
probability (%)
0
2
4
6
8
10
12
14
16
18
20
#rounds
N = 16
N = 128
N = 1024

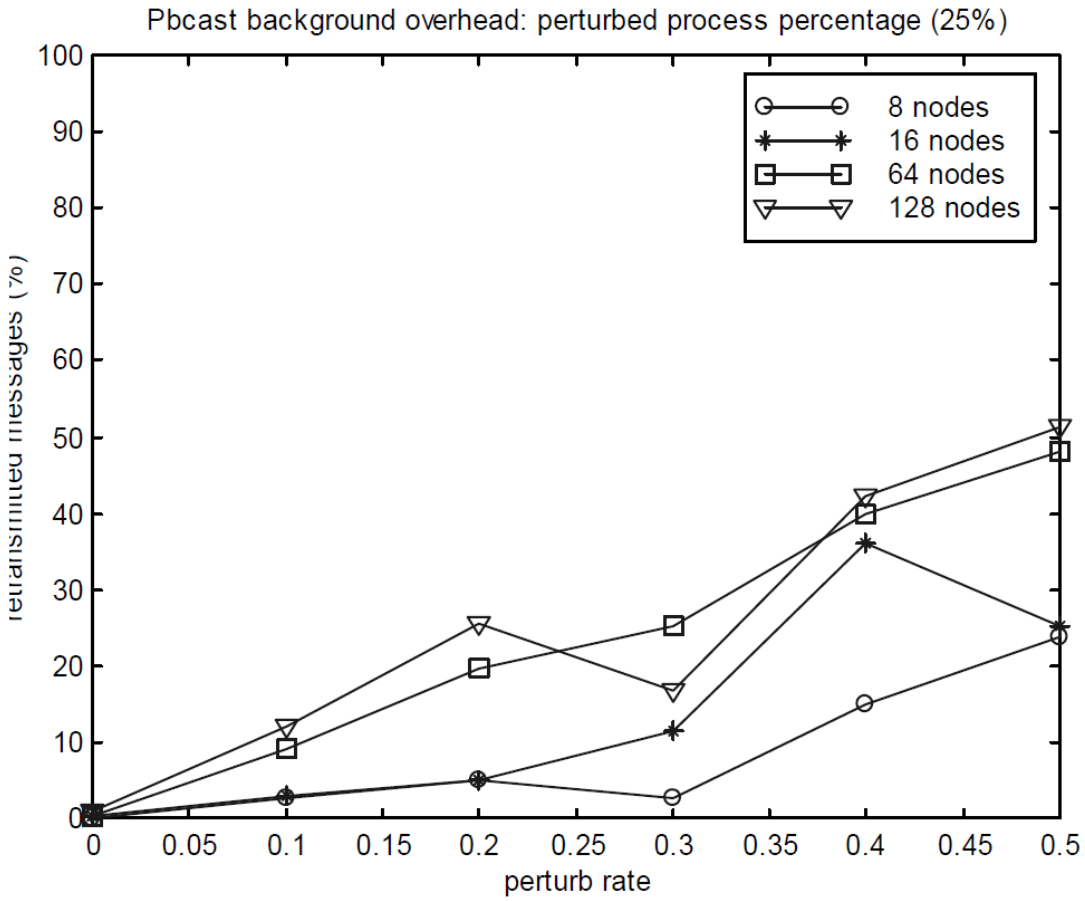
Throughput Stability



Throughput Stability



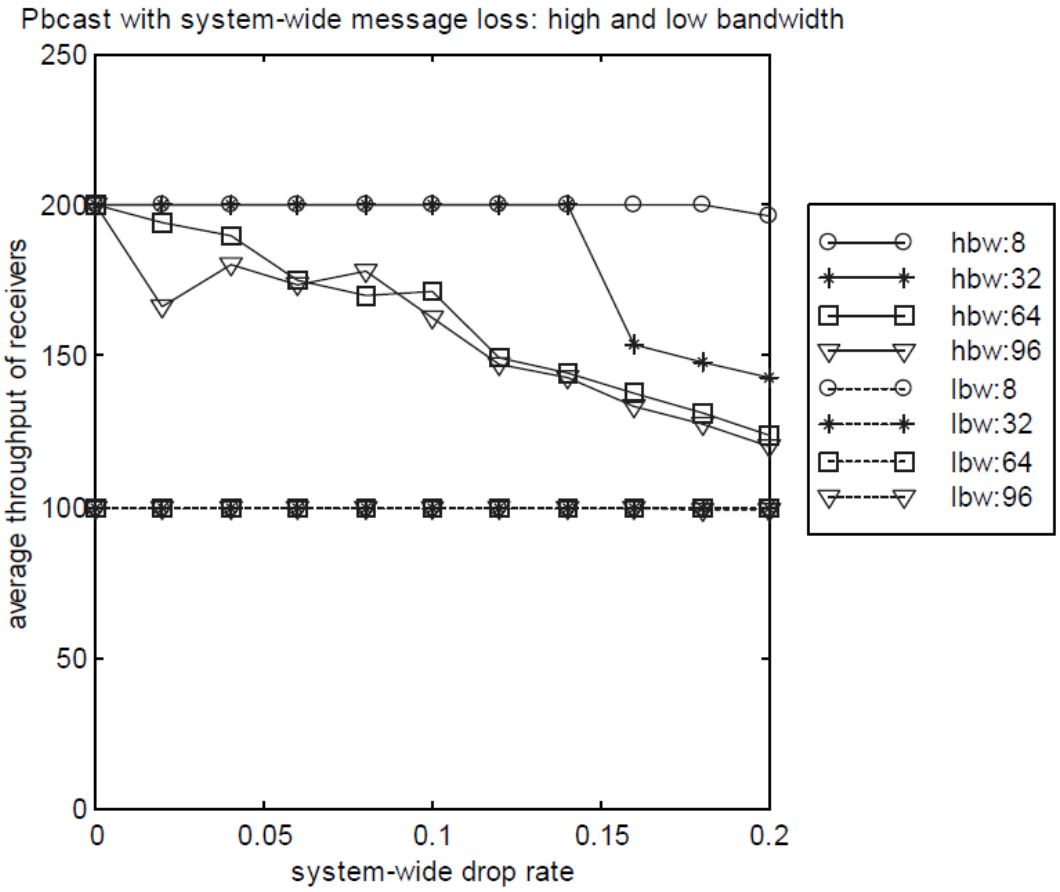
Overhead


Pbcast background overhead: perturbed process percentage (25%)
8 nodes
16 nodes
64 nodes
128 nodes
retransmitted messages (%)
perturb rate
0
10
20
30
40
50
60
70
80
90
100
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

Difference with previous paper?

- Stability of Throughput
- Process failure

Dark Side of The Protocol



Some Discussion points

- What about IP Multicast?
- Most Recent First Transmission – Is it always ideal?
- How efficient is gossip in general in wired networks ?

Some Discussion Points

- Are there any scenarios when Pbcast sends gossip messages unnecessarily?

  -> What if Pbcast message injection rate is very low

Some Discussion Points

- Is Pbcast really required if IP multicast is very very reliable?

-> Leads to wastage of messages again!

Some Discussion Points

- What if IP multicast is both reliable and unreliable?

-> Expected delivery latency has different models. Throughput also fluctuates

Soln?

-> Buffering

Some Discussion Points

- What about recovery from delivery failures?

->A temporarily failed process will experience a gap in the messages delivered to it.

Soln?

-> Repositories

-> State transfer

Questions?