Multicast

Qi Huang

CS6410 2009 FA

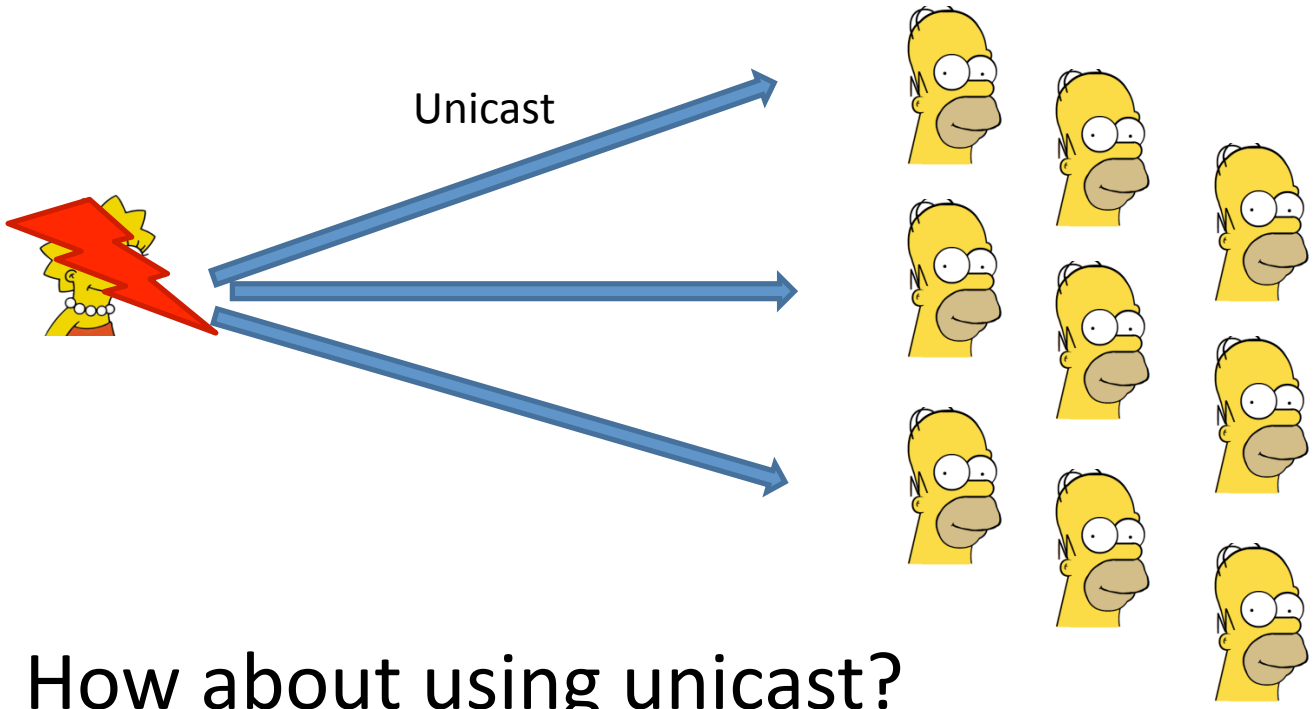
10/29/09

Cornell University
Computer Science

What is multicast?

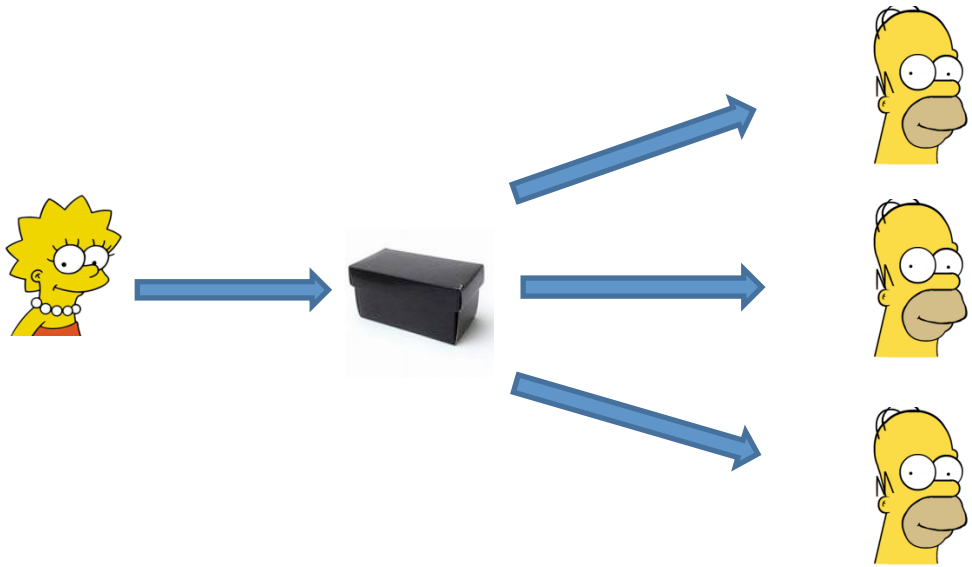
- Basic idea: same data needs to reach a set of multiple receivers
- Application:

Why not just unicast?



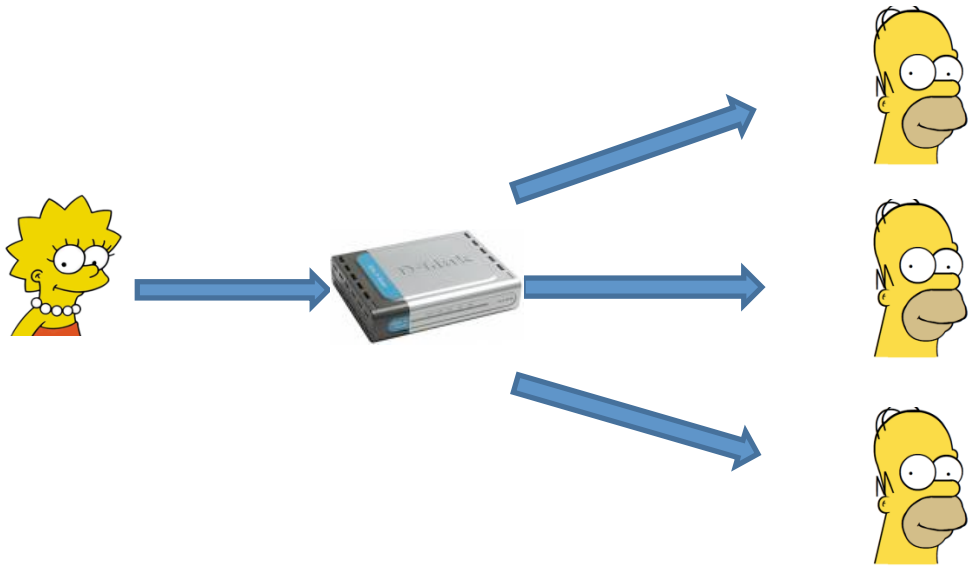
- How about using unicast?
- Sender's overhead grows linearly with the group size
- Bandwidth of sender will exhaust

How does multicast work?



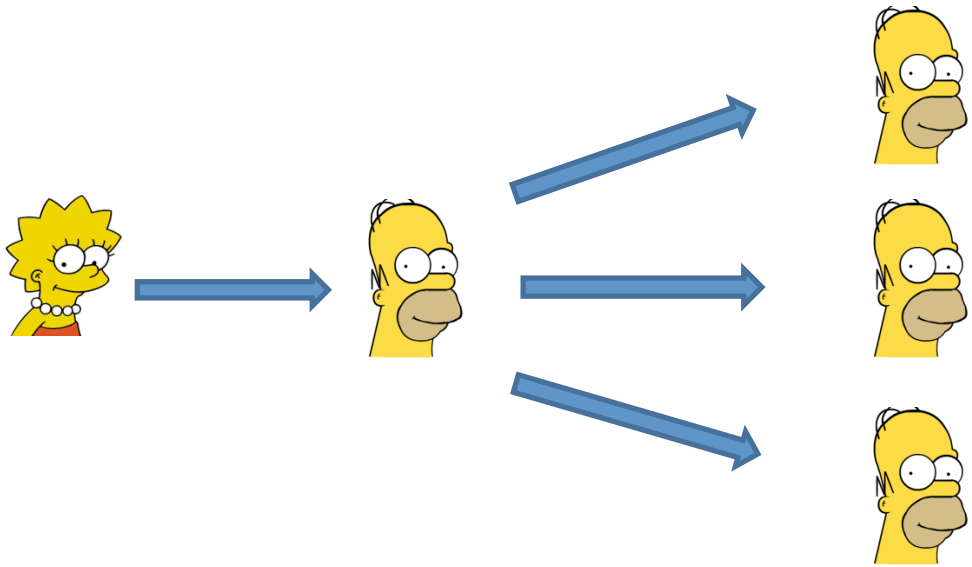
- Use intermediate nodes to copy data

How does multicast work?



- IP multicast (IPMC)
  - Use router to copy data on the network layer, Deering proposed in 1990

How does multicast work?



- Application layer multicast (ALM)
  - Use proxy node or end host to copy data on application layer,

State of the art

- IPMC is disabled in WAN
  - Performance issues (Karsruhe, Sigcomm)
  - Feasibility issues
  - Desirability issues
  - Efficient in LAN
- ALM has emerged as an option
  - Easy to deploy and maintain
  - Based on internet preferred unicast
  - No generic infrastructure

Research focus

- Scalability
  - Scale with group size
  - Scale with group number
- Reliability
  - Atomicity multicast
  - Repair best-effort multicast

Trade-off is decided by the application scenarios

Agenda

- Bimodal Multicast
  - ACM TOCS 17(2), May 1999
- SplitStream: High-Bandwidth Multicast in Cooperative Environments
  - SOSP 2003

Bimodal Multicast

Ken Birman
Cornell University

Mark Hayden
Digital Research Center

Oznur Ozkasap
Koç University

Zhen Xiao
Peking University

Mihai Budiu
Microsoft Research

Yaron Minsky
Jane Street Capital

Application scenario

- Stock exchange, air traffic control needs:
  - Reliability of critical information transmission
  - Prediction of performance
  - 100x scalability under high throughput

Outline

- Problem
- Design
- Analysis
- Experiment
- Conclusion

Problem

- Virtual Synchrony
  - Offer strong relialbility guarantee

Costly overhead, can not scale under stress

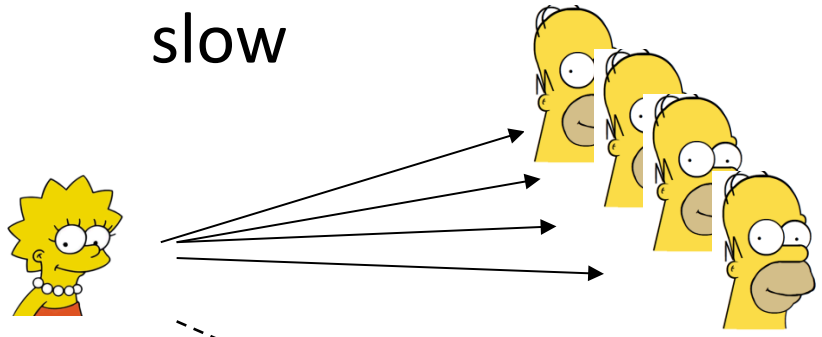
- Scalable Reliable Multicast (SRM)
  - Best effort reliability, scale better

Not reliable, repair can fail under bigger group size and heavy load

Let's see how these happen

Problem

- Virtual Synchrony
  - Under heavy load, some one may be slow



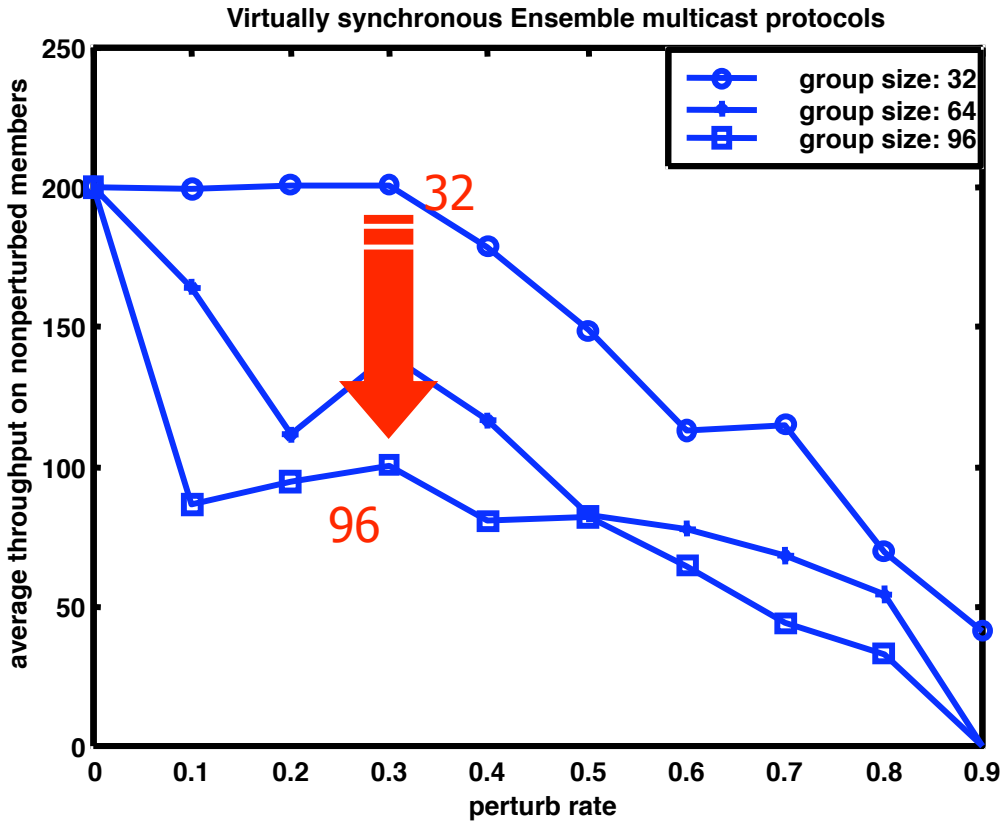
Most members are healthy….

… but one is slow

i.e. something is contending with the receiver, delaying its handling of incoming messages…

Problem

- Virtual Synchrony
  - Slow receiver can collapse the system throughput



Problem

- Virtual Synchrony (reason?)
  - Data for the slow process piles up in the sender's buffer, causing flow control to kick in (prematurely)
  - More sensitive failure detector mechanism will rise the risk of erroneous failure classification

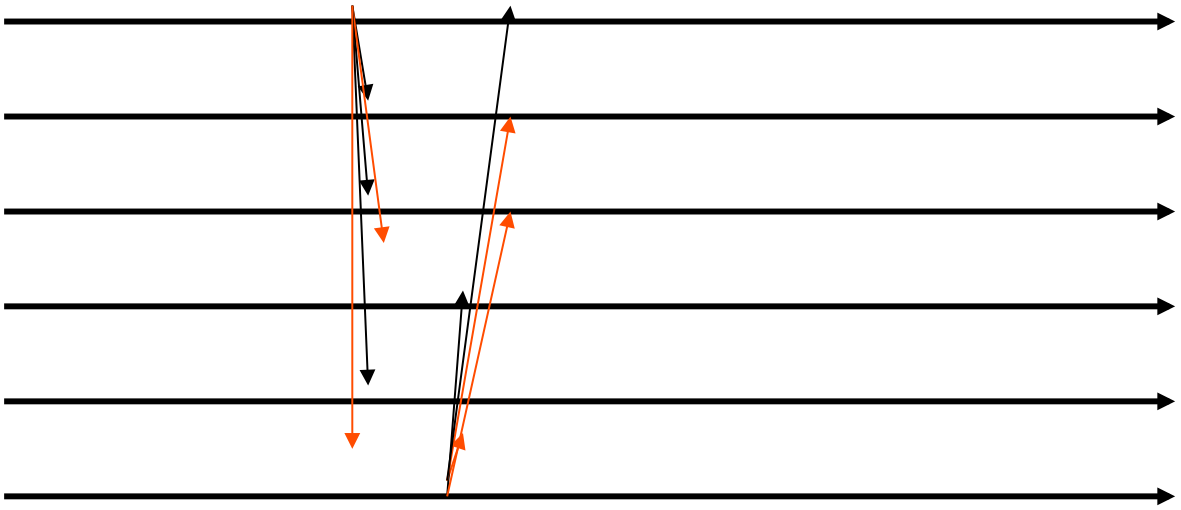
Problem

- SRM
  - Lacking knowledge of membership, SRM's NACK and retransmission is multicast
  - As the system grows large the "probabilistic suppression" fails (absolute likelihood of mistakes rises, causing the background overhead to rise)

Design

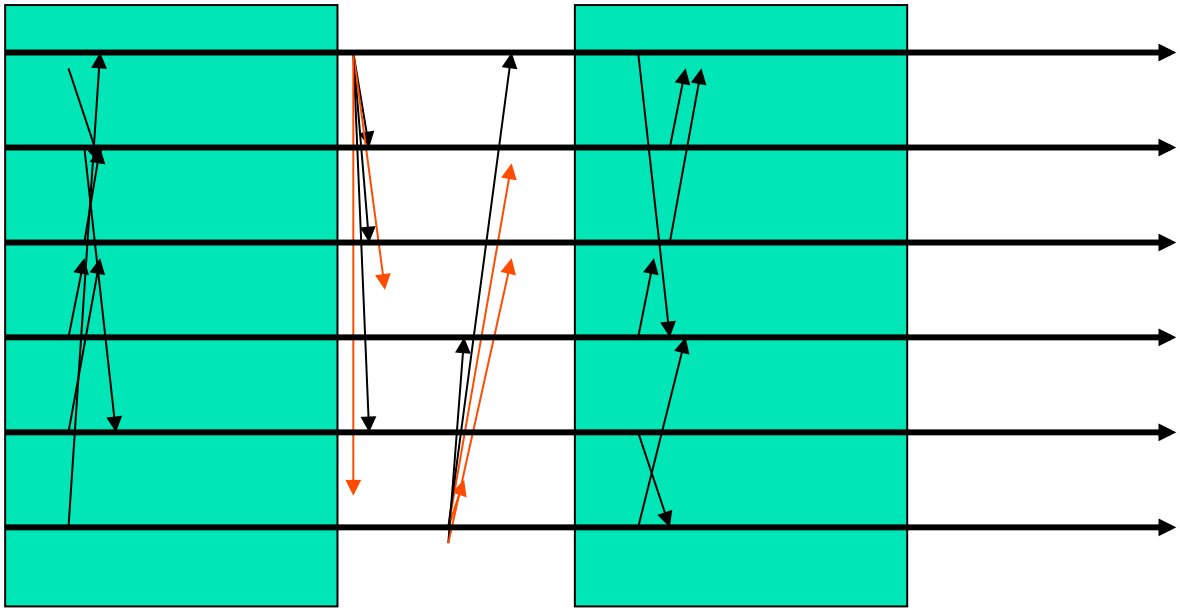
- Two step multicast
  - Optimistic multicast to disseminate message unreliably
  - Use two-phase anti-entropy gossip to repair
- Benefits
  - Knowlege of membership achieves better repair control
  - Gossip provides:
    - Ligher way of detecting loss and repair
    - Epidemic model to predict performance

Design



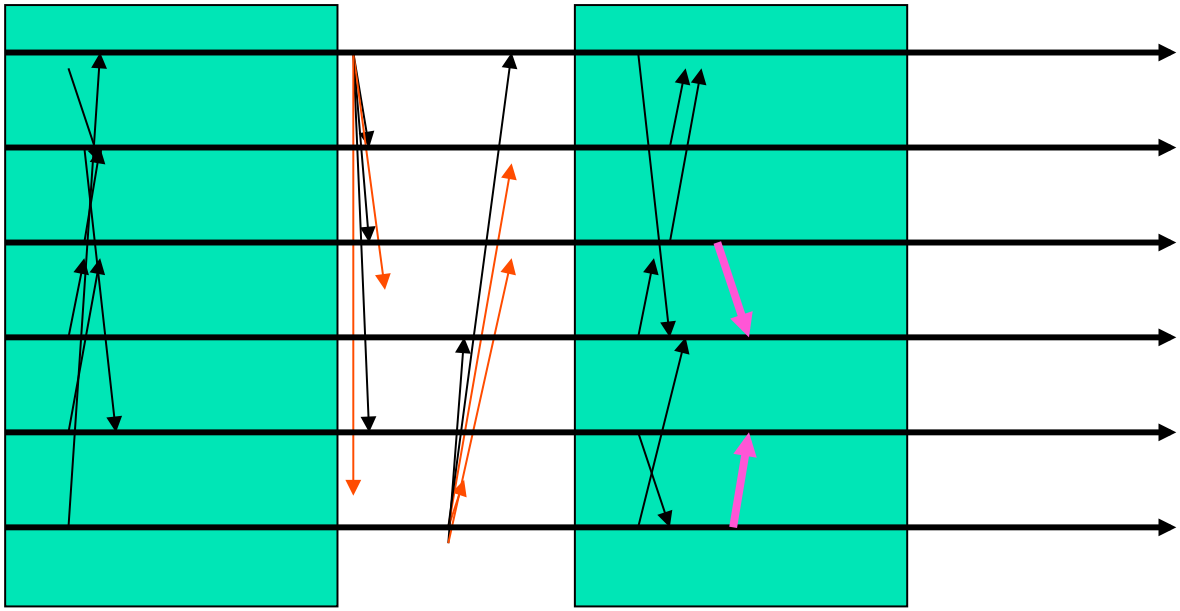
Start by using *unreliable* multicast to rapidly distribute the message. But some messages may not get through, and some processes may be faulty. So initial state involves partial distribution of multicast(s)

Design



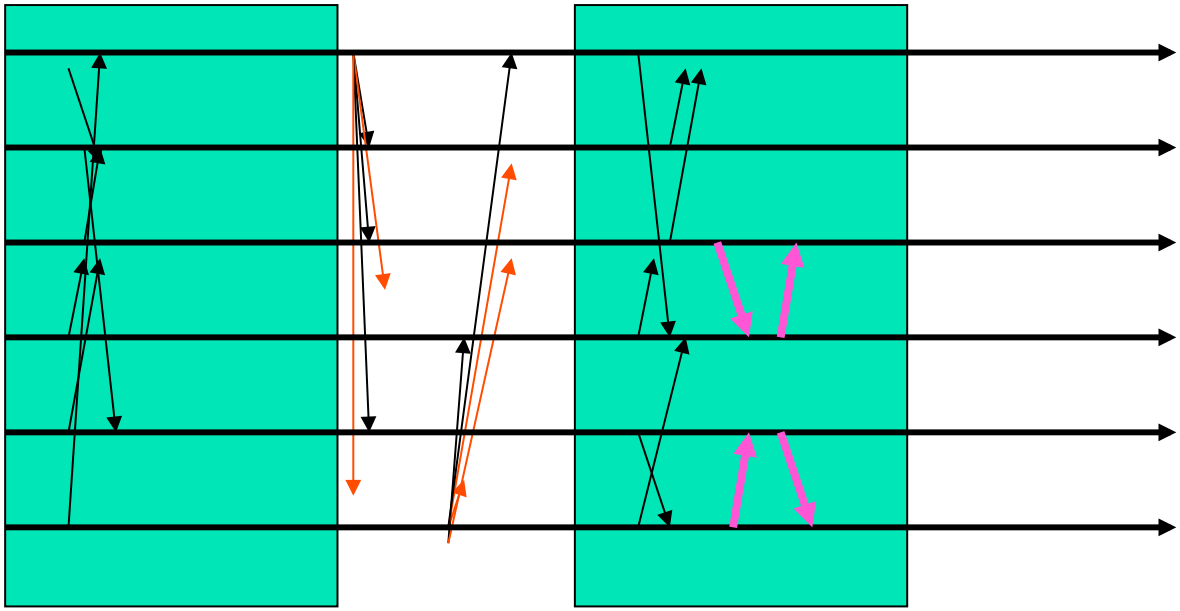
Periodically (e.g. every 100ms) each process sends a *digest* describing its state to some randomly selected group member. The digest identifies messages. It doesn't include them.

Design



Recipient checks the gossip digest against its own history and *solicits* a copy of any missing message from the process that sent the gossip

Design



Processes respond to solicitations received during a round of gossip by retransmitting the requested message. The round lasts much longer than a typical RPC time.

Design

- Deliver a message when it is in FIFO order
- Garbage collect a message when you believe that no “healthy” process could still need a copy (we used to wait 10 rounds, but now are using gossip to detect this condition)
- Match parameters to intended environment

Design

- Worries
  - Someone could fall behind and never catch up, endlessly loading everyone else
  - What if some process has lots of stuff others want and they bombard him with requests?
  - What about scalability in buffering and in list of members of the system, or costs of updating that list?

Design

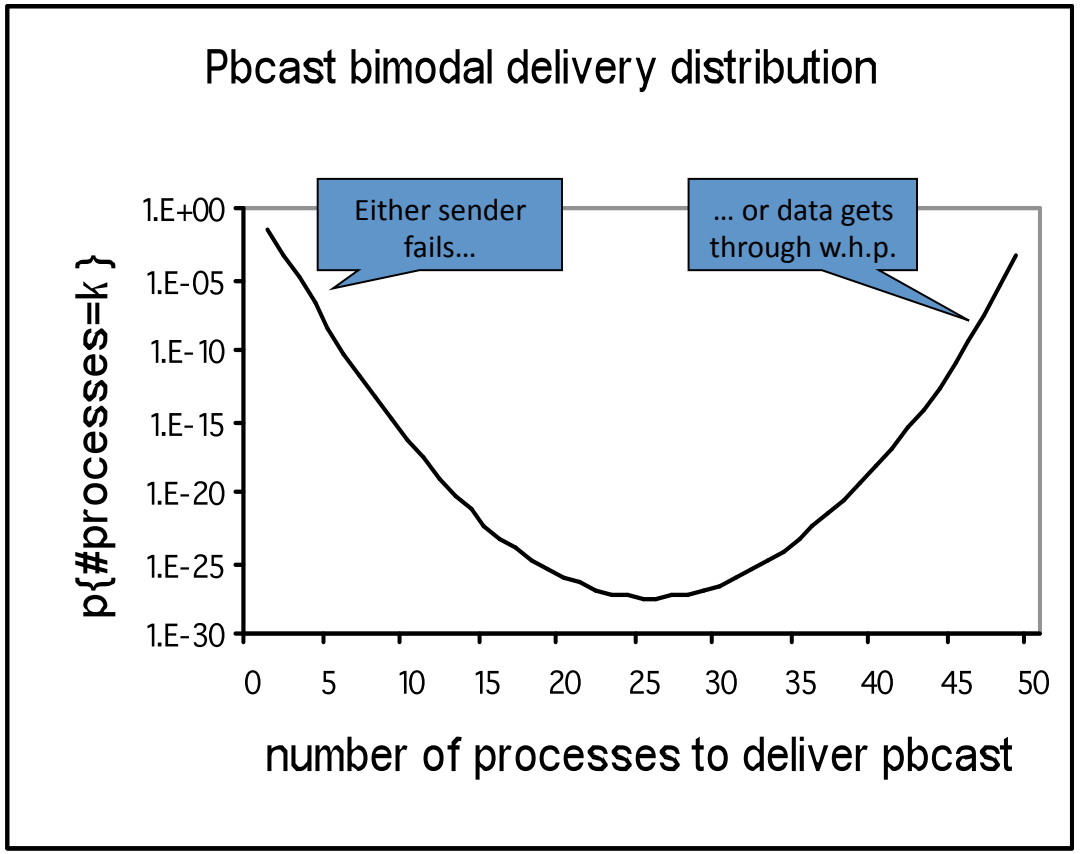
- Optimization
  - Request retransmissions most recent multicast first
  - Bound the amount of data they will retransmit during any given round of gossip.
  - Ignore solicitations that have expired round number, reasoning that they are from faulty nodes

Design

- Optimization
  - Don't retransmit duplicate message
  - Use IP multicast when retransmitting a message if several processes lack a copy
    - For example, if solicited twice
    - Also, if a retransmission is received from "far away"
    - Tradeoff: excess messages versus low latency
  - Use regional TTL to restrict multicast scope

Analysis

- Use epidemic theory to predict the performance



Analysis

- Failure analysis
  - Suppose someone tells me what they hope to "avoid"
  - Model as a predicate on final system state
  - Can compute the probability that pbcast would terminate in that state, again from the model

Analysis

- Two predicates
  - Predicate I: More than 10% but less than 90% of the processes get the multicast
  - Predicate II: Roughly half get the multicast but crash failures might "conceal" outcome
  - Easy to add your own predicate. Our methodology supports any predicate over final system state

Analysis

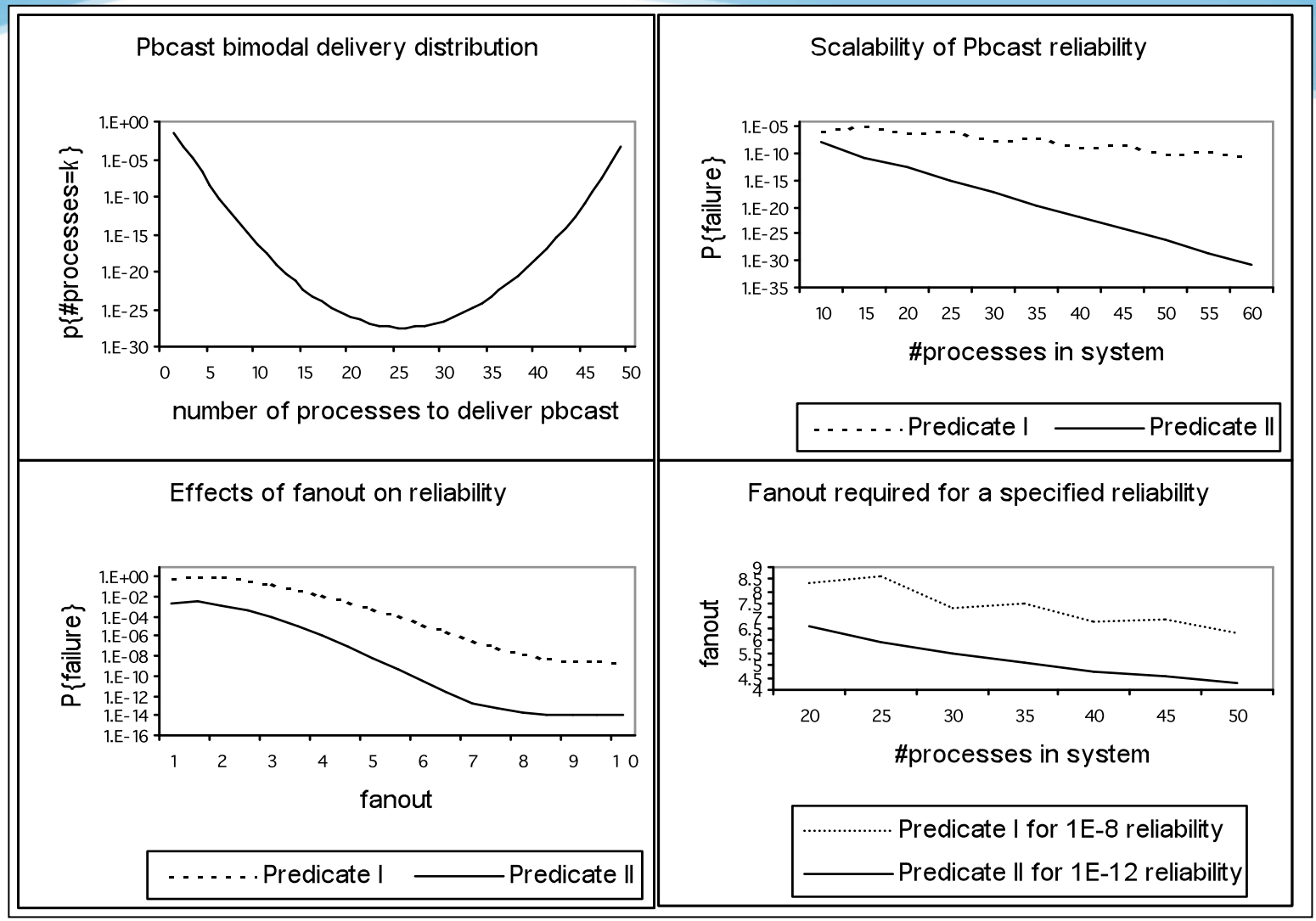
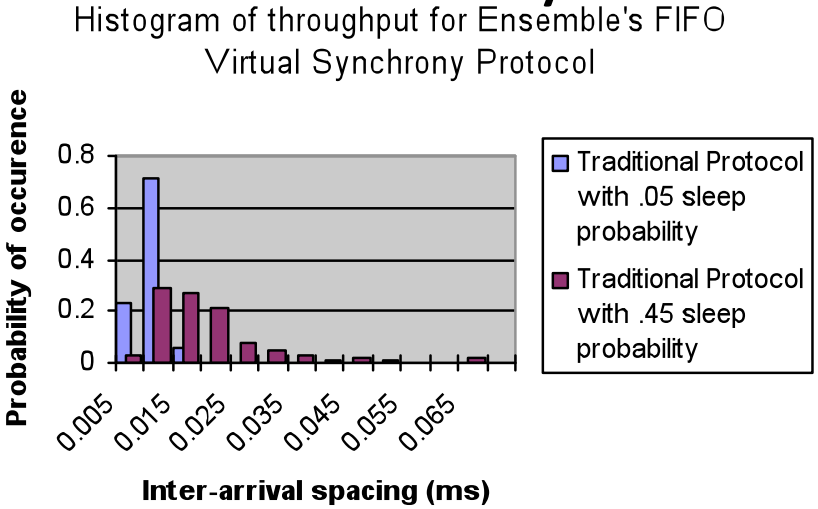
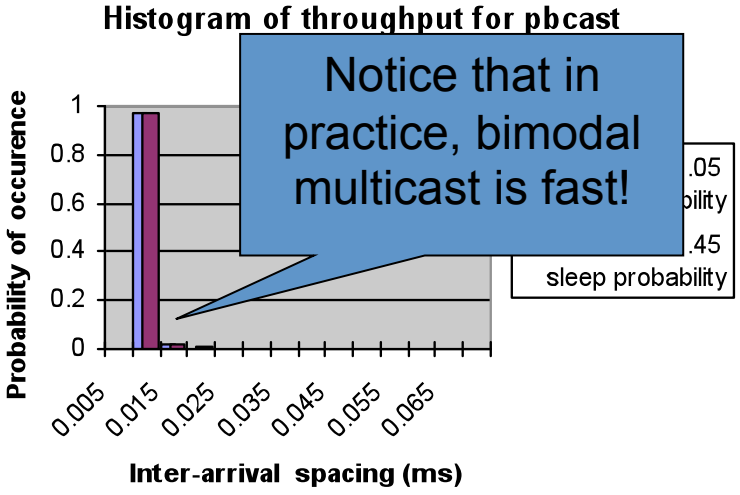


Figure 5: Graphs of analytical results

Experiment

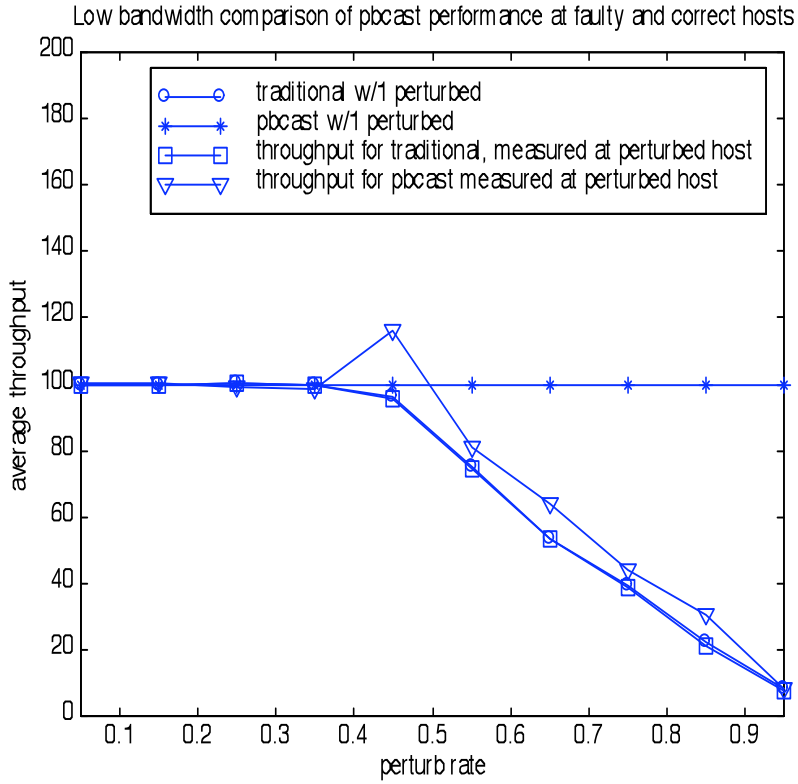
- Src-dst latency distributions

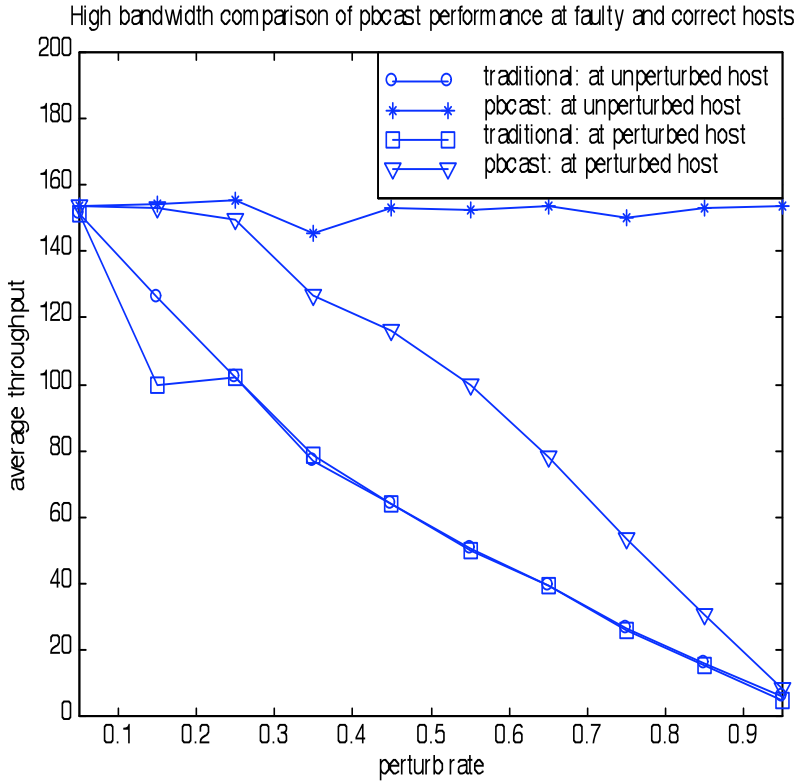




Experiment

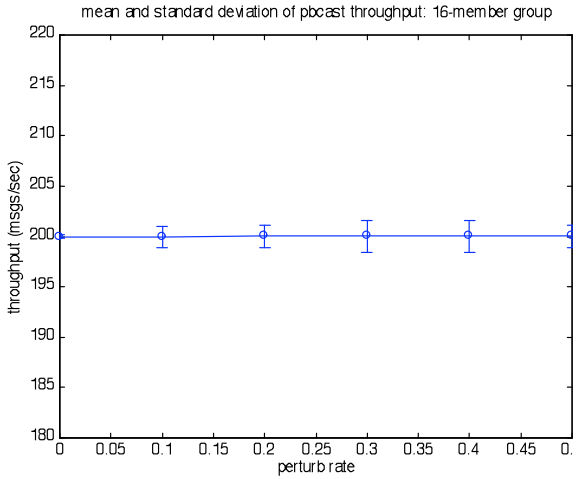
Revisit the problem figure, 32 processes

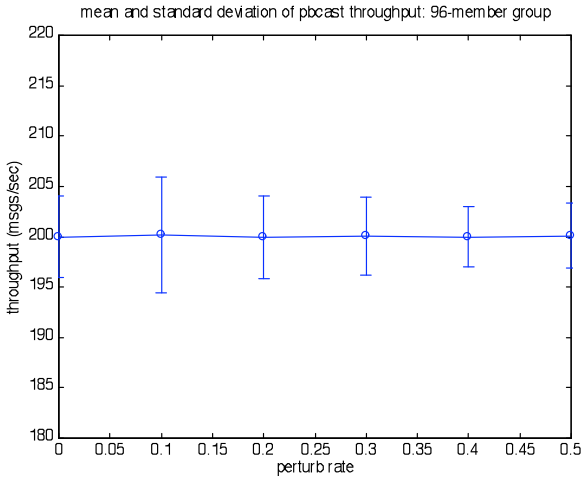


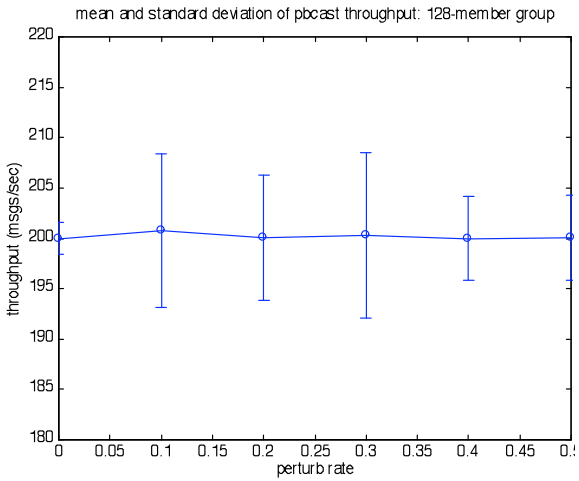


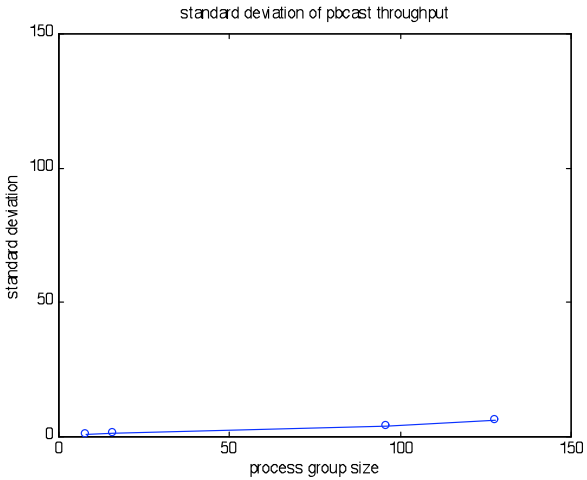
Experiment

- Throughput with various scale



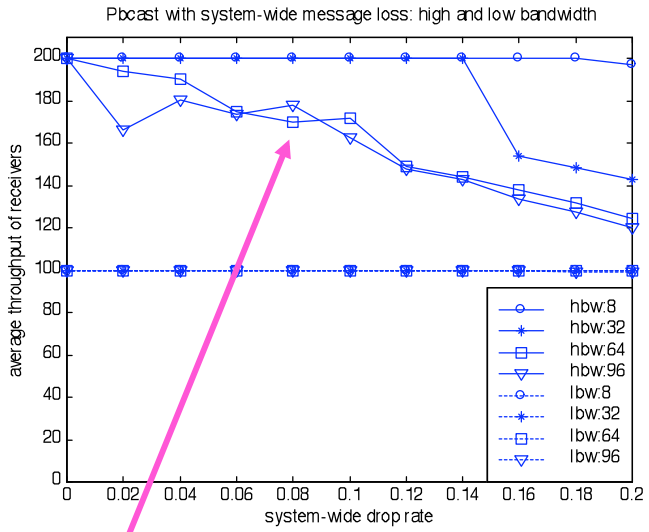


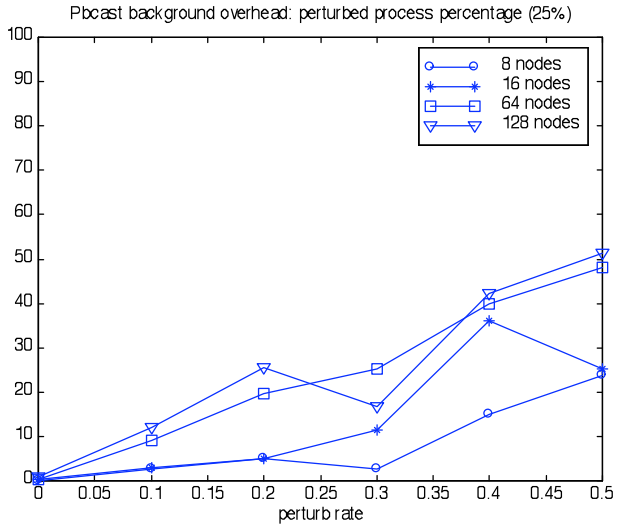




Experiment

Impact of packet loss on reliability and retransmission rate

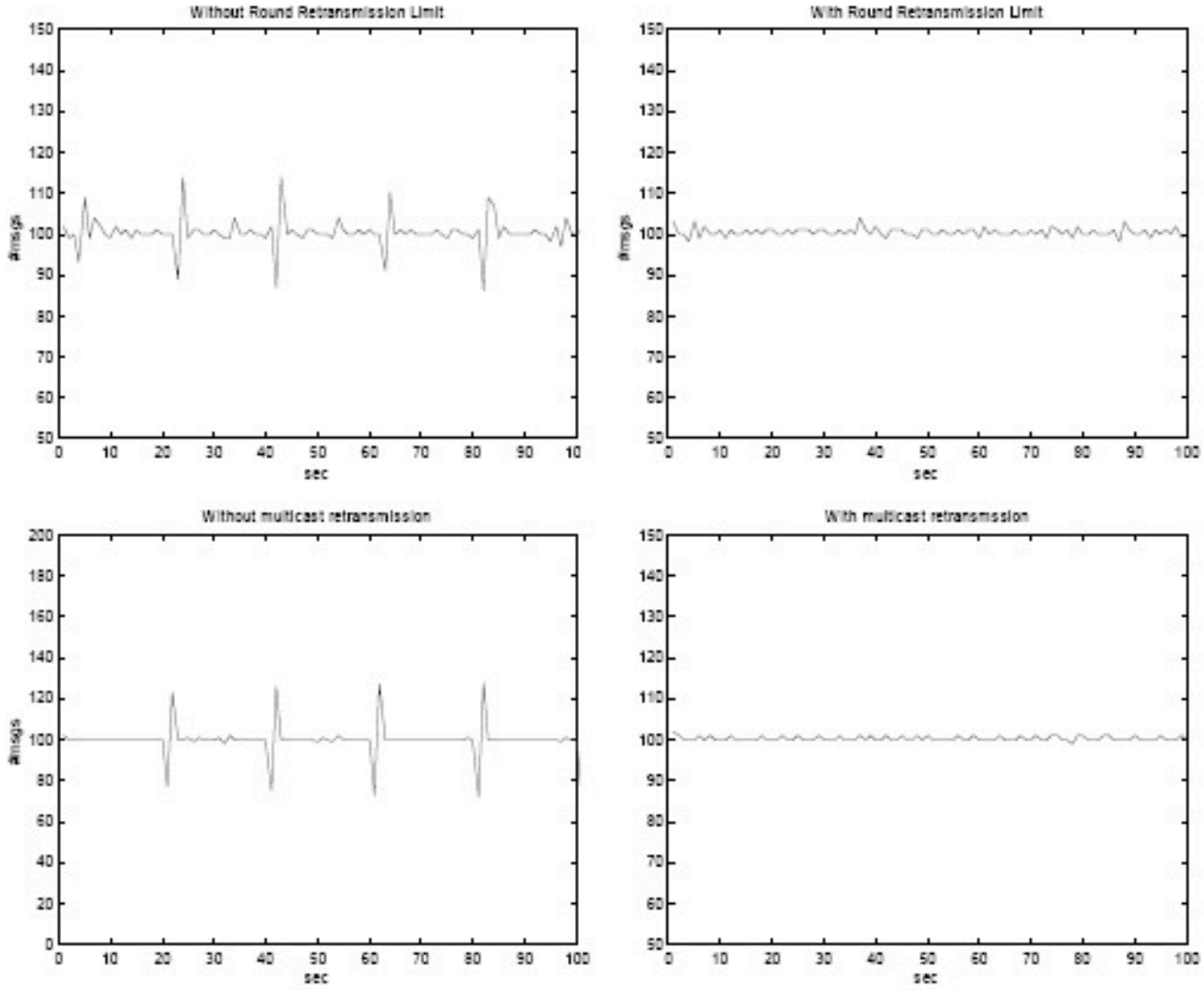




Notice that when network becomes overloaded, healthy processes experience packet loss!

Experiment

Optimization



Conclusion

- Bimodal multicast (pbcast) is reliable in a sense that can be formalized, at least for some networks
  - Generalization for larger class of networks should be possible but maybe not easy
- Protocol is also very stable under steady load even if 25% of processes are perturbed
- Scalable in much the same way as SRM

Splitstream

Miguel Castro
Microsoft Research

Peter Druschel
MPI-SWS

Anne-Marie Kermarrec
INRIA

Animesh Nandi
MPI-SWS

Antony Rowstron
Microsoft Research

Atul Singh
NEC Research Lab

Applicatoin scenario

- P2P video streaming, needs:
  - Capacity-aware bandwidth utilization
  - Latency-aware overlay
  - High tolerance of network churn
  - Load balance for all the end hosts

Outline

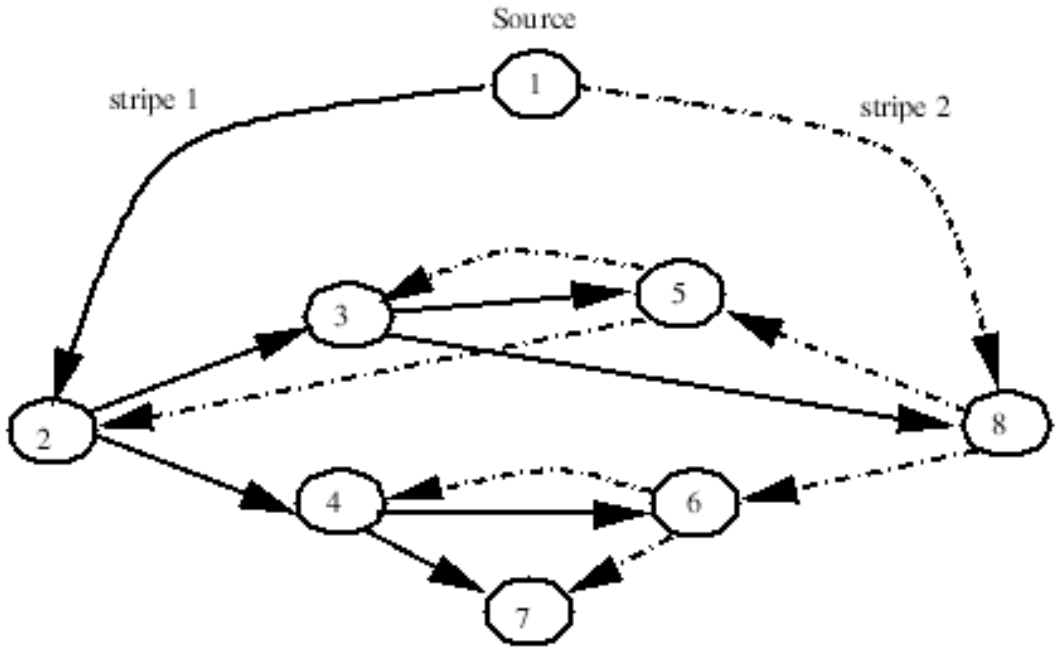
- Problem
- Design (with background)
- Experiment
- Conclusion

Problem

- Tree-based ALM
  - High demand on few internal nodes
- Cooperative environments
  - Peers contribute resources
  - We don't assume dedicated infrastructure
  - Different peers may have different limitations

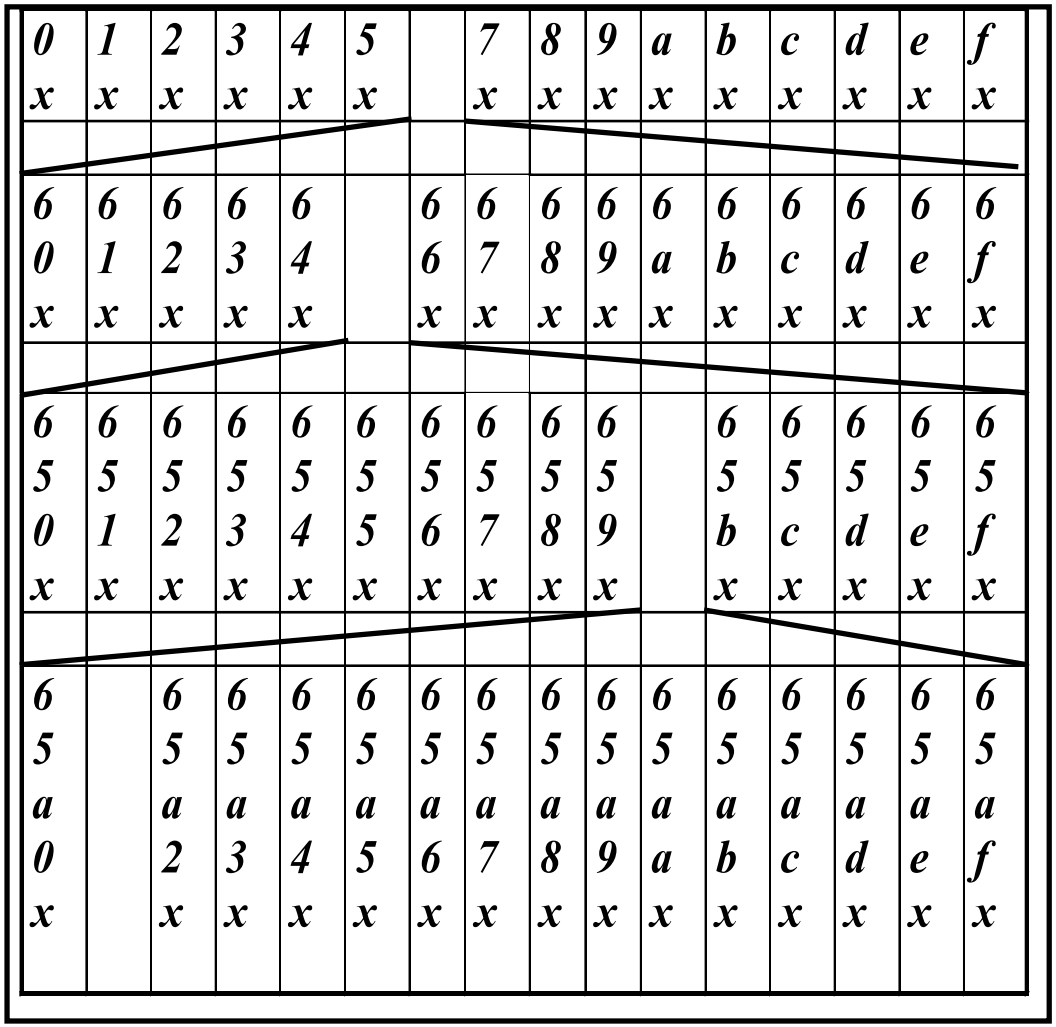
Design

- Split data into stripes, each over its own tree
- Each node is internal to only one tree
- Built on Pastry and Scribe



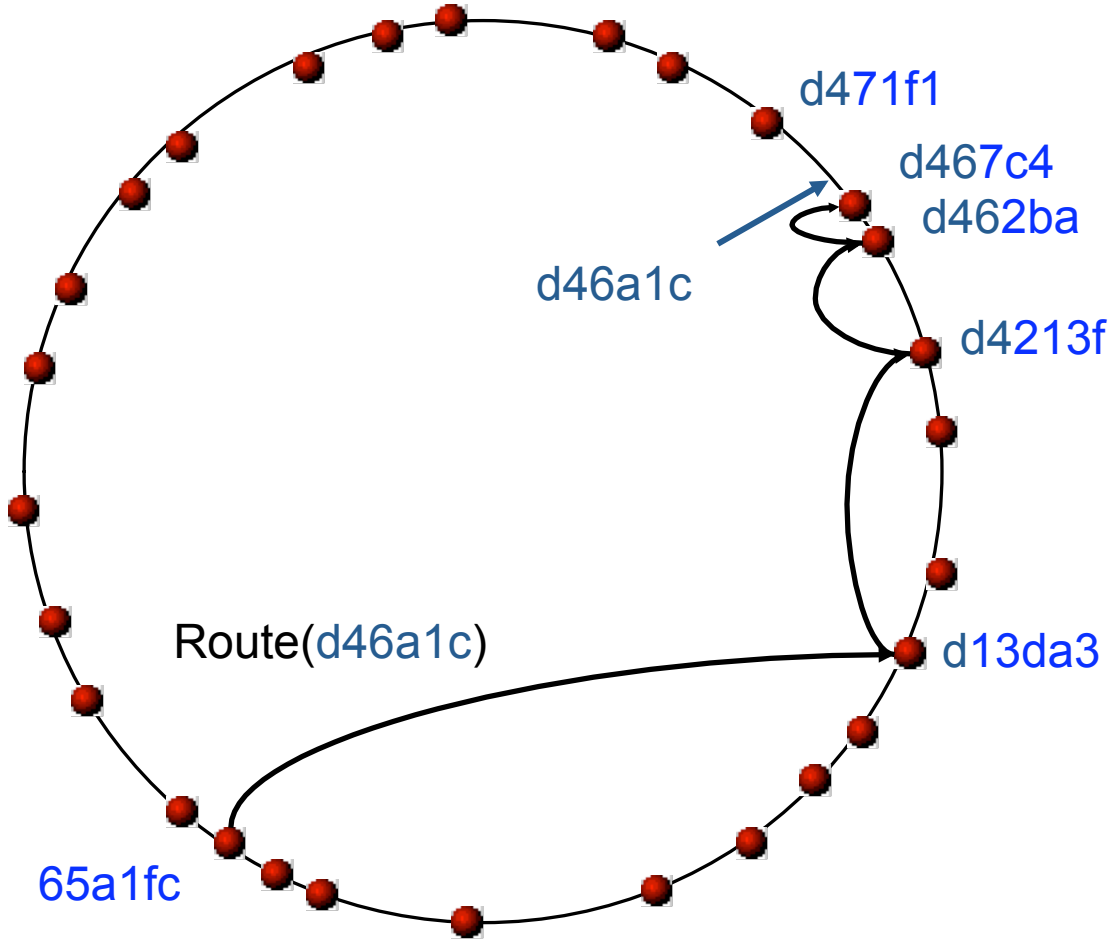
Background

- Pastry



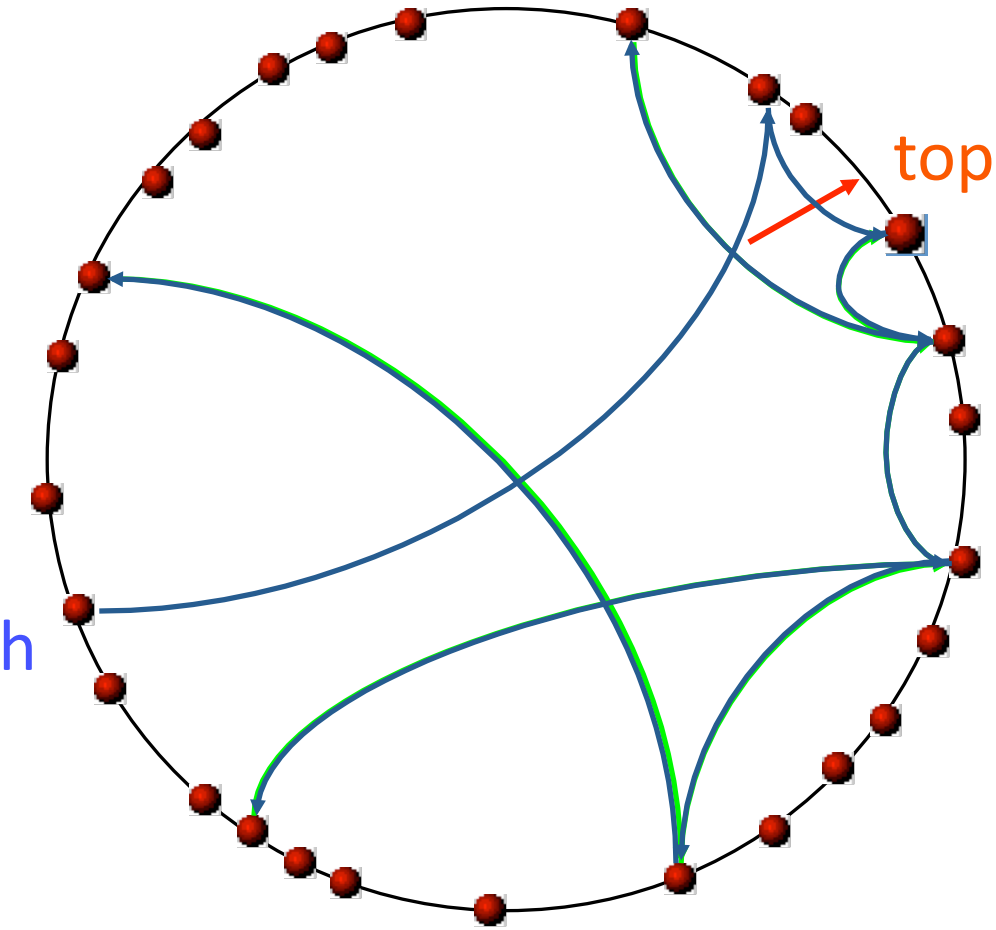
Background

- Pastry



Background

- Scribe

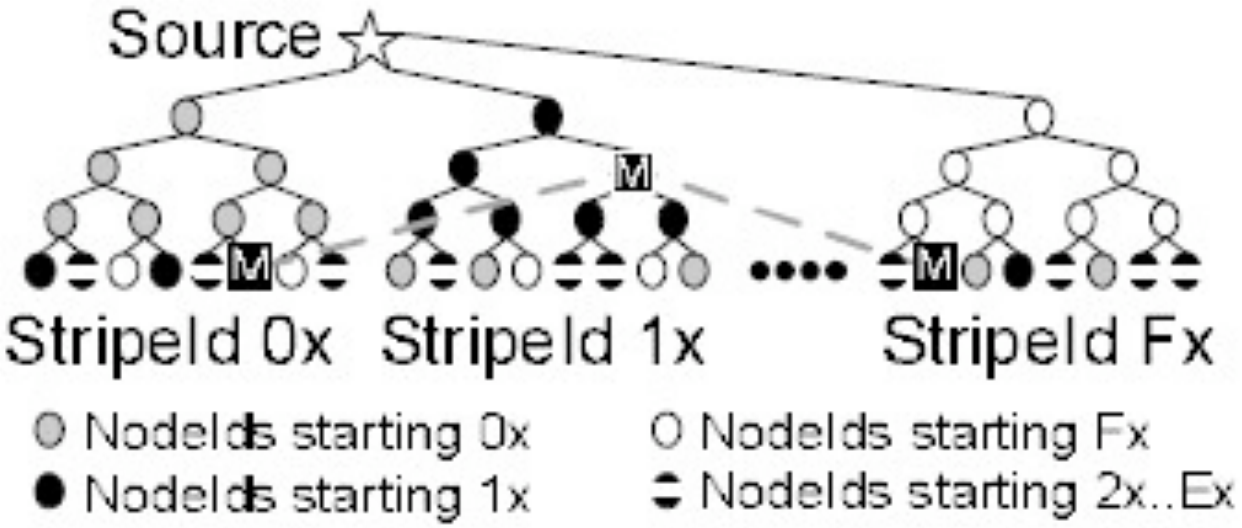


Design

- Stripe
  - SplitStream divides data into stripes
  - Each stripe uses one Scribe multicast tree
  - Prefix routing ensures property that each node is internal to only one tree
    - Inbound bandwidth: can achieve desired indegree while this property holds
    - Outbound bandwidth: this is harder—we'll have to look at the node join algorithm to see how this works

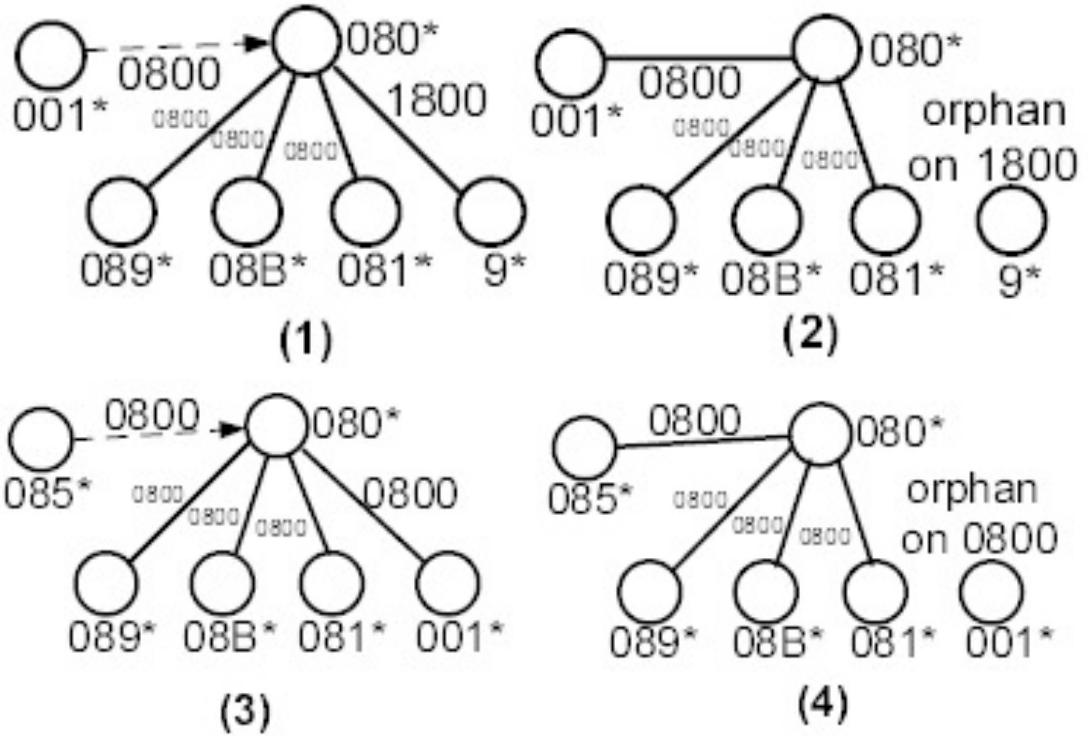
Design

- Stripe tree initialize
  - # of strips = # of Pastry routing table columns
  - Scribe's push-down fails in Splitstream



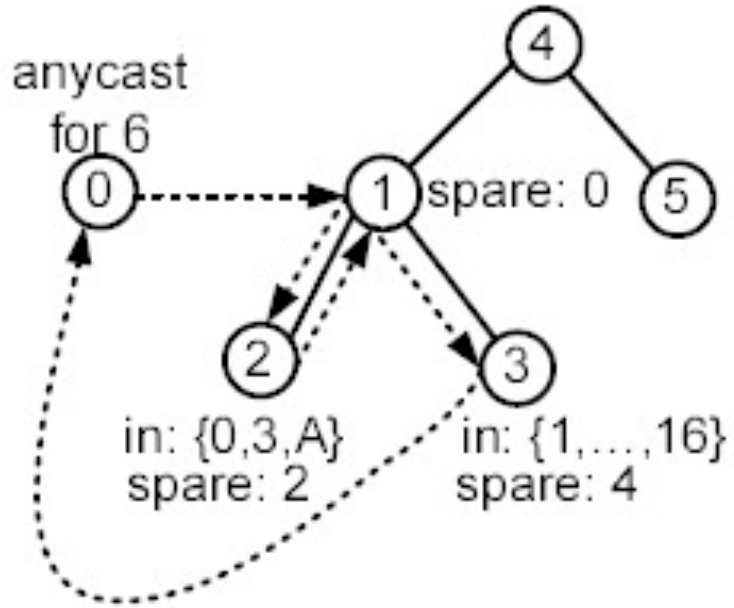
Design

- Splitstream push-down
  - The child having less prefix match between node ID and stripe ID will be push-down to its sibling



Design

- Spare capacity group
  - Organize peers with spare capacity as a Scribe tree
  - Orphan nodes anycast in the spare capacity group to join



Design

- Correctness and complexity
  - Node may fail to join a non prefix maching tree
  - But the failure of forest construction is unlikely

$$|\mathbf{N}| \times \mathbf{k} \times \left(\mathbf{1} - \frac{\mathbf{I_{min}}}{\mathbf{k}}\right)^{\frac{\mathbf{C}}{\mathbf{k-1}}}$$

Design

- Expected state maintained by each node is *O(log|N|)*
- *Expected number of messages to build forest is O(|N|log|N|) if trees are well balanced and O(|N|^2) in the worst case*
- *Trees should be well balanced if each node forwards its own stripe to two other nodes*

Experiment

- Forest construction overhead

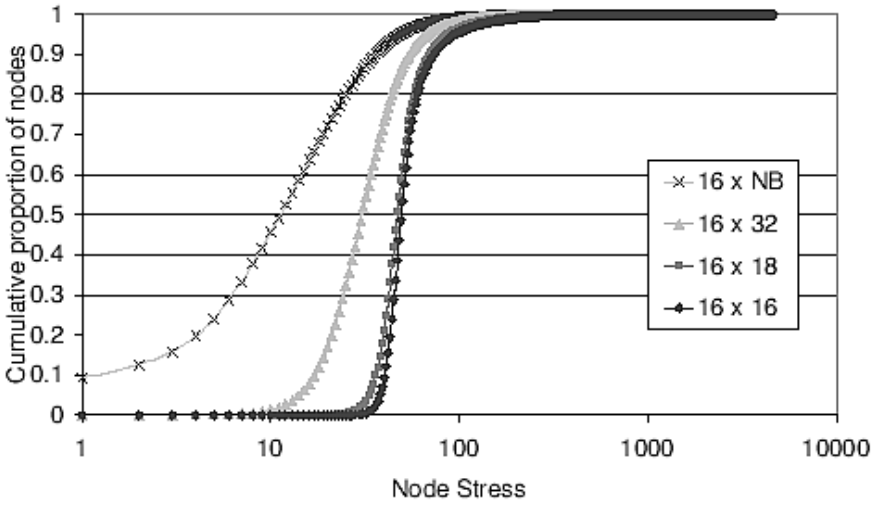


Figure 8: Cumulative distribution of node stress during forest construction with 40,000 nodes on GATech.

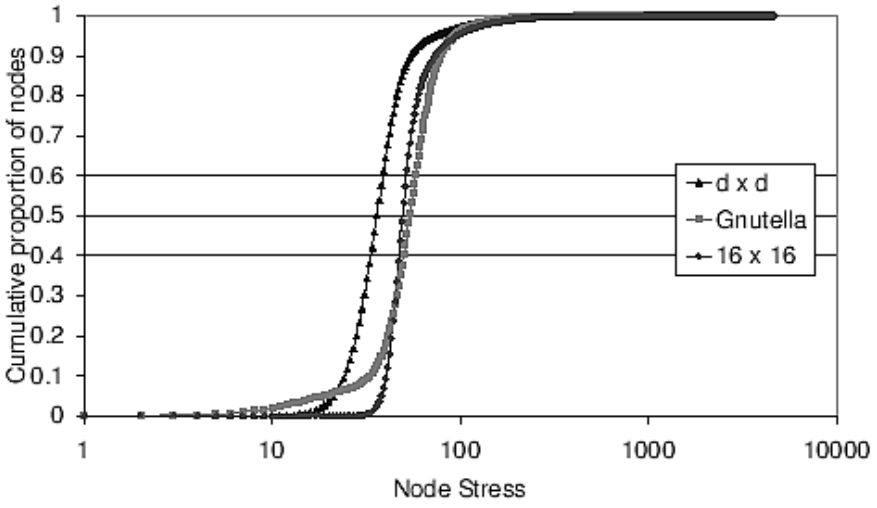
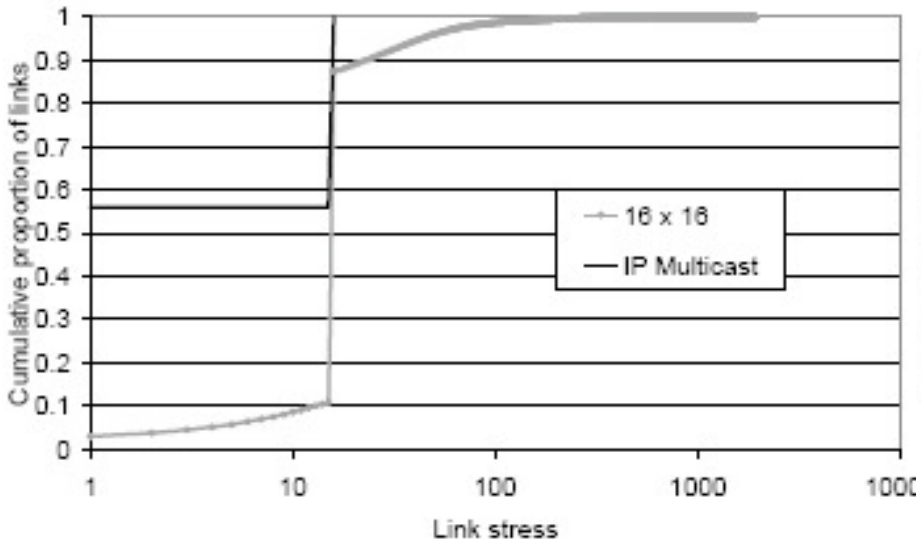


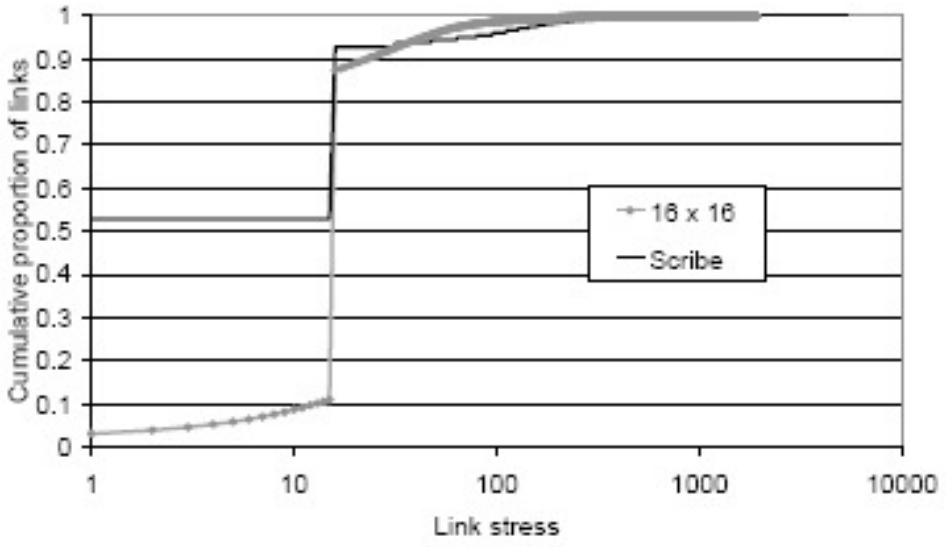
Figure 9: Cumulative distribution of node stress during forest construction with 40,000 nodes on GATech.

Experiment

- Multicast performance



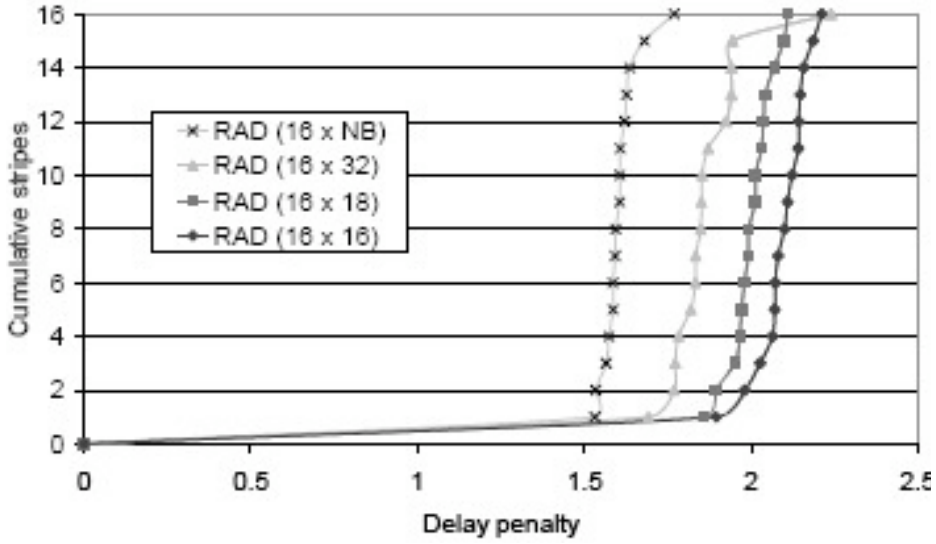
(a) SplitStream vs IP

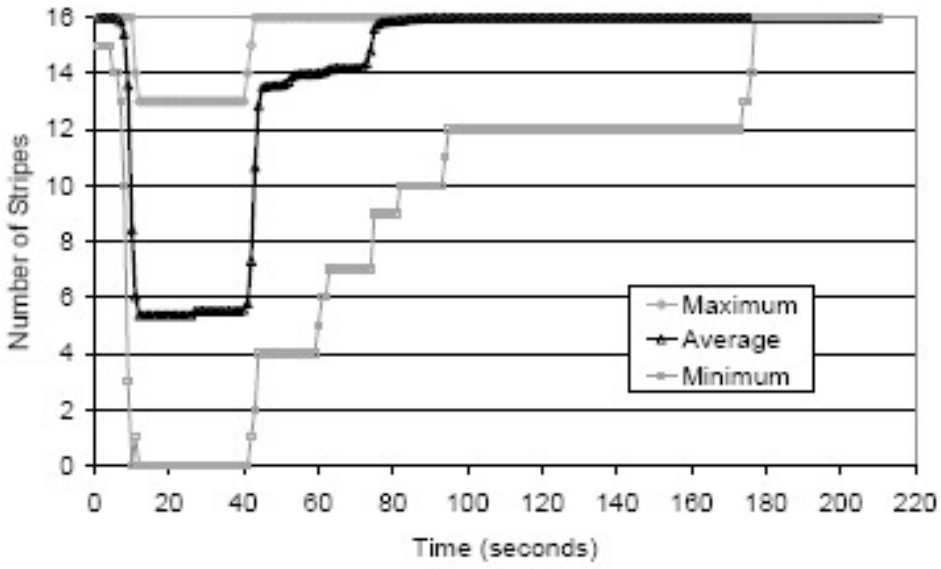


(b) SplitStream vs Scribe

Experiment

- Delay and failure resilience





Conclusion

- Striping a multicast into multiple trees provides:
  - Better load balance
  - High resilience of network churn
  - Respect of heteogeneous node bandwidth capacity
- Pastry support latency aware and cycle free overlay construction

Discussion

- Flaws of Bimodal multicast
  - Relying on IP multicast may fail
  - Tree multicast is load unbalanced and churn vulnarable
- Flaws of Splitstream
  - Network churn will introduce off-pastry link between parent and child, thus loose the benefit of pastry [from Chunkyspread]
  - ID-based constrains is stronger than load constrains