MULTICAST

Presented by Tom Ternquist

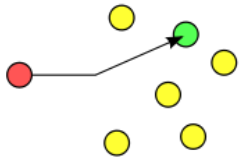
CS 6410

10/28/10

Network Communication

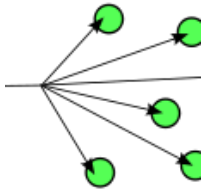
- Unicast
  - One to one

e.g. downloading file from webserver



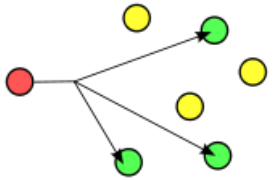
- Broadcast
  - One to many (everyone)

e.g. radio transmission



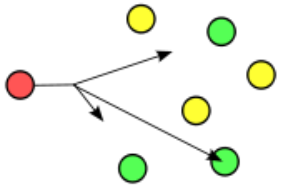
- Multicast
  - One to many (groups)

e.g. streaming video, IRC



- Anycast
  - One to one of many

e.g. server selection

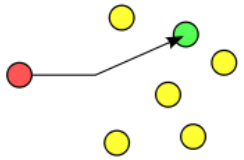


Images courtesy of Wikipedia

Network Communication

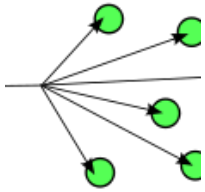
- Unicast
  - One to one

e.g. downloading file from webserver



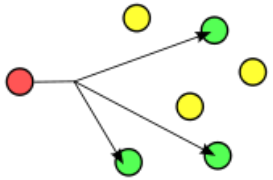
- Broadcast
  - One to many (everyone)

e.g. radio transmission



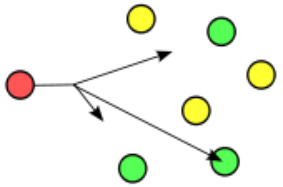
- Multicast
  - One to many (groups)

e.g. streaming video, IRC



- Anycast
  - One to one of many

e.g. server selection



Images courtesy of Wikipedia

Multicast

- Two Challenges
  - Identifying the receivers
  - How to address packets that are sent to receivers
- Can’t specify individual IP addresses
  - Obvious scaling problem
- Solution - Address Indirection
  - Multicast Groups
- Remaining Challenges
  - Establishing multicast groups and managing membership
  - How to route messages to recipients within multicast groups

Multicast

- IP Multicast
  - Pros: Efficient
  - Cons: Unstable, can send more than capacity; not widely deployed
- Network Overlay Multicast
  - Pros: Can only send what you're capable of; can use it anywhere
  - Cons: Not as efficient as IP Multicast

Multicast

- IP Multicast
  - Pros: Efficient
  - Cons: Unstable, can send more than capacity; not widely deployed
- Network Overlay Multicast
  - Pros: Can only send what you're capable of; can use it anywhere
  - Cons: Not as efficient as IP Multicast

Overlay Network

The Good

- Incrementally Deployable
- Adaptable
- Robust
- Customizable
- Standard

The Bad

- Management Complexity
- The Real World
- Inefficiency
- Information Loss

Types of Application-Level Multicast

Single-Source Multicast

- Trusted servers provide infrastructure for distribution
- Manual infrastructure setup

Peer-to-Peer Multicast

- Depends on cooperative clients
- Flexible, distributed infrastructure

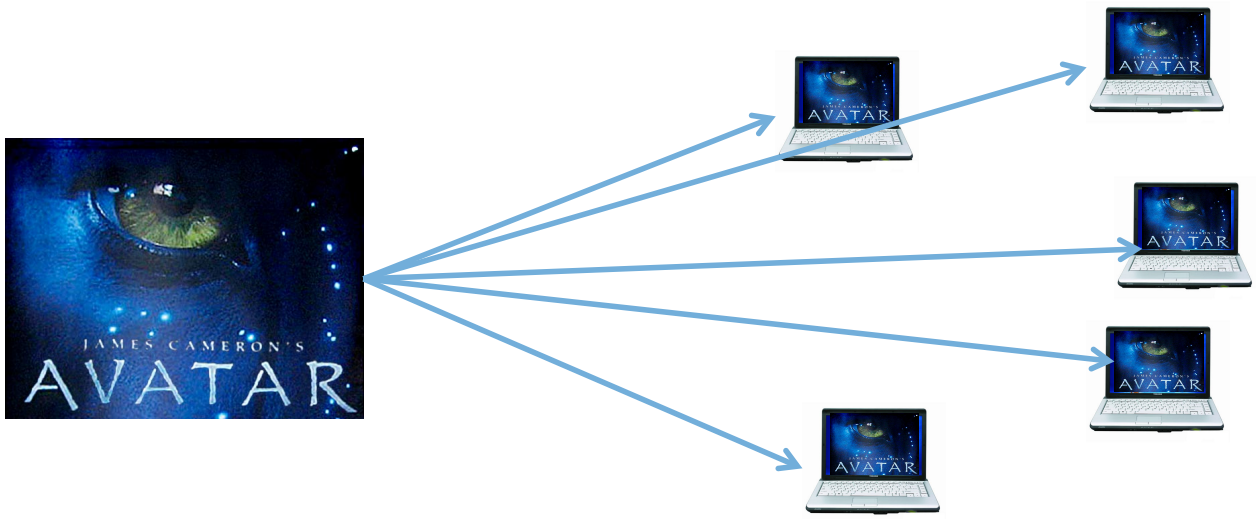
Overcast: Reliable Multicasting with an Overlay Network

Authors

- John Jannotti
  - Assistant Professor at Brown
- David Gifford
  - Computational Genomics at MIT
- Kirk Johnson
- Frans Kaashoek
  - Professor at MIT
  - Founded SightPath Inc, acquired by Cisco
- James O'Toole
  - Student of Kaashoek
  - MIT PhD
  - High school dropout

Motivation and Goals

- Deliver bandwidth-intensive content on demand
- How do we scale to support large numbers of clients for long-running content?
- Applications
  - On demand high-quality video streams
  - Full-fidelity content



Approach

- Application-level multicast using an overlay network
- Distribution trees that maximize each node's bandwidth and efficiently utilize network substrate topology
- Global status at root that allows quick client joining while still scalable
- Scalability and efficiency approaching IP Multicast

Design – Overview

- Tree Building
- Maintaining Node Status
- Reliability
- Dissemination

Bandwidth Optimization – Tree Building Protocol

- New node contacts root of Overcast group, root set as *current*
- Tries to get as far away from root as possible without sacrificing bandwidth

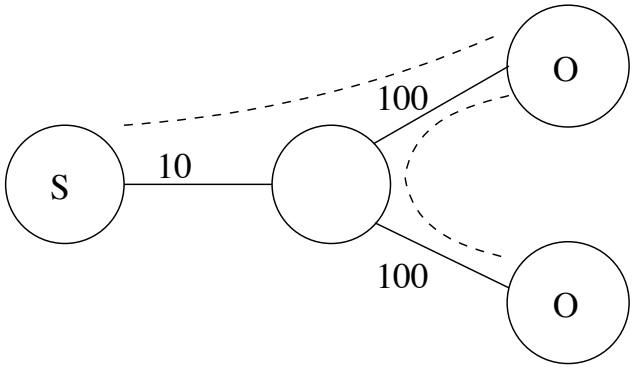
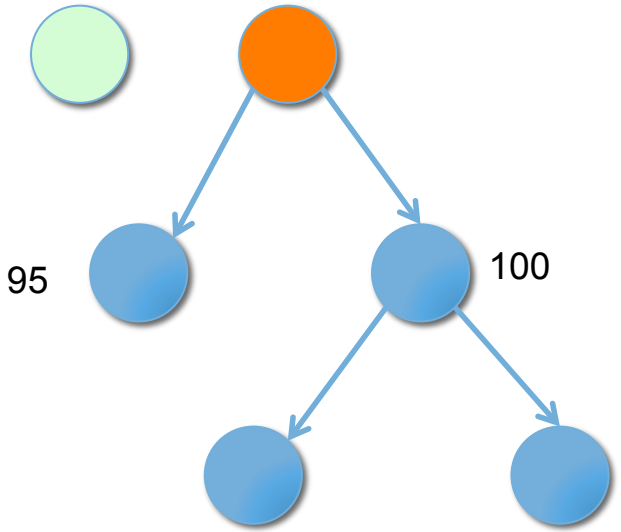


Figure 1: An example network and Overcast topology. The straight lines are the links in the substrate network. These links are labeled with bandwidth in Mbit/s. The curved lines represent connections in the Overlay network. **S** represents the source, **O** represents two Overcast nodes.

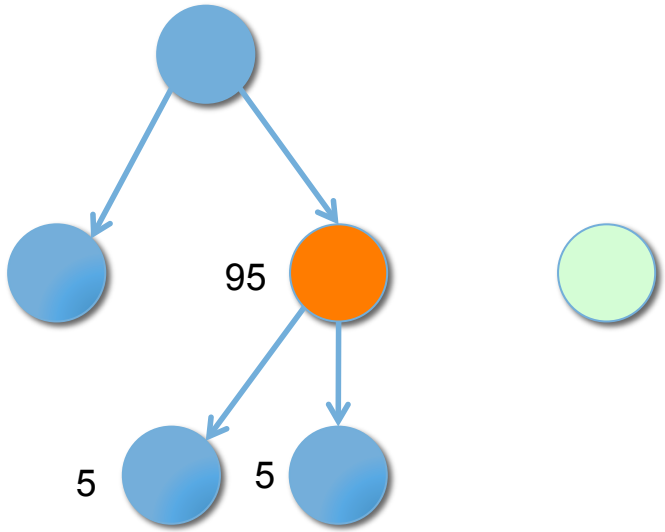
Bandwidth Optimization – Tree Building Protocol

- New node contacts root of Overcast group, root set as *current*
- Tries to get as far away from root as possible without sacrificing bandwidth



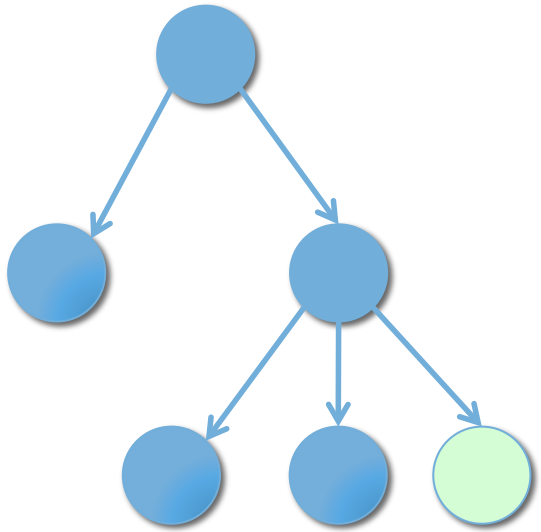
Bandwidth Optimization – Tree Building Protocol

- New node contacts root of Overcast group, root set as *current*
- Tries to get as far away from root as possible without sacrificing bandwidth



Bandwidth Optimization – Tree Building Protocol

- New node contacts root of Overcast group, root set as *current*
- Tries to get as far away from root as possible without sacrificing bandwidth

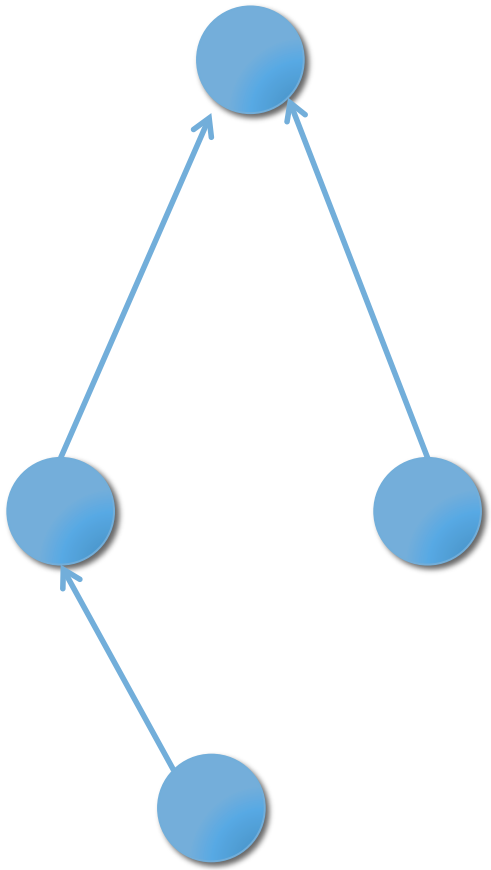


Maintaining Node Status – Up/Down Protocol

- Keeps track of which nodes are currently up and which are down
- Relies on periodic check ins with node's parent.
- Check in notifications are propagated up the tree to the root.
  - "Death certificates"
  - "Birth certificates"
  - Changes to reporting node's "extra information"
  - Certificates/changes from node's children

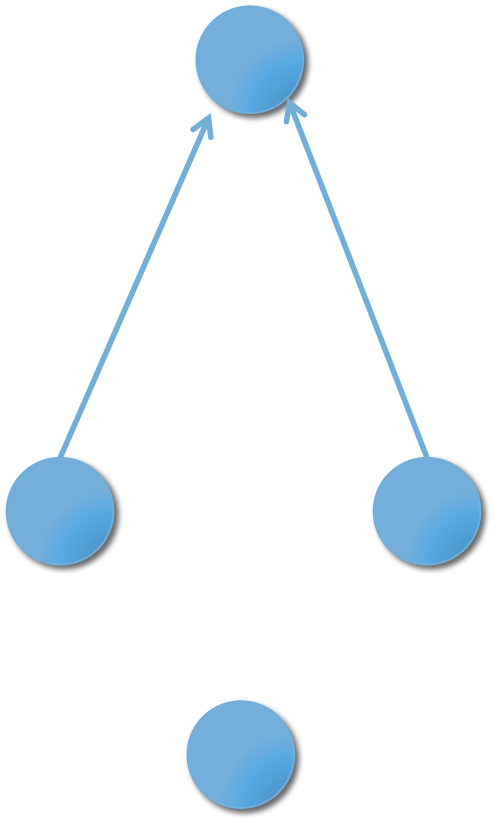
Maintaining Node Status – Up/Down Protocol

- Potential race condition when node chooses a new parent
  - Death certificate from former parent
  - Birth certificate from new parent
  - Birth certificate arrives just prior to death certificate.



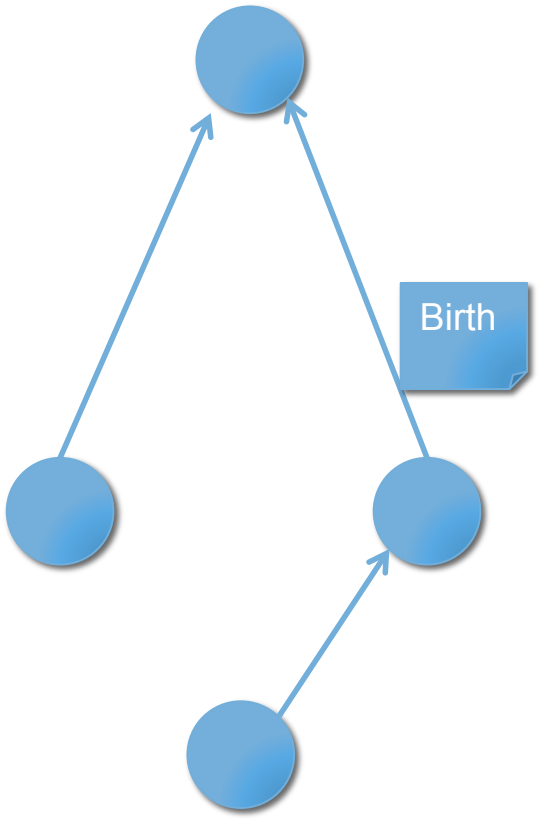
Maintaining Node Status – Up/Down Protocol

- Potential race condition when node chooses a new parent
  - Death certificate from former parent
  - Birth certificate from new parent
  - Birth certificate arrives just prior to death certificate.



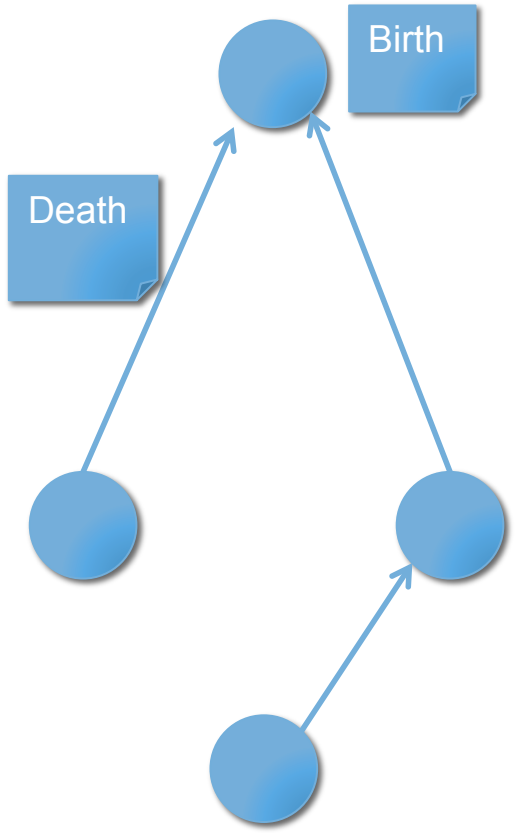
Maintaining Node Status – Up/Down Protocol

- Potential race condition when node chooses a new parent
  - Death certificate from former parent
  - Birth certificate from new parent
  - Birth certificate arrives just prior to death certificate.



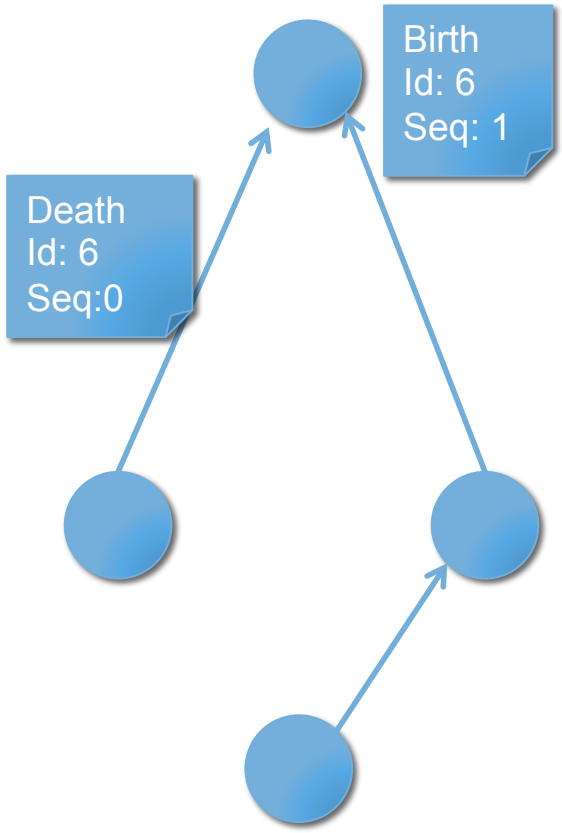
Maintaining Node Status – Up/Down Protocol

- Potential race condition when node chooses a new parent
  - Death certificate from former parent
  - Birth certificate from new parent
  - Birth certificate arrives just prior to death certificate.



Maintaining Node Status – Up/Down Protocol

- **Solution**: Nodes maintain sequence number for parent changes.



Reliability – Root Replication

- Potential for overload and reliability problems
- Replicated roots used in round-robin fashion
- Fault tolerance thorough linear configuration at top of hierarchy

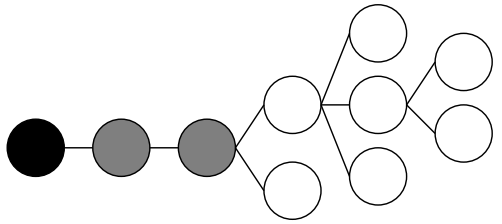


Figure 2: A specially configured distribution topology that allows either of the grey nodes to quickly stand in as the root (black) node. All filled nodes have complete status information about the unfilled nodes.

Dissemination – “Overcasting”

- Support for HTTP Clients
- Data moves along distribution tree using TCP streams
- Failures during overcasting result in the distribution tree being rebuilt
  - Distribution paused while rebuilding is in process
  - Designed around assumption that application level buffers will mask most failures

Evaluation – Goals

- How well does Overcast utilize bandwidth?
- How does Overcast perform relative to IP Multicast?
- How resilient is Overcast to failures?

Evaluation – Methodology

- Evaluated using simulation because real-world deployments too small
- Overlay network simulated with constant number of network nodes with increasing numbers of Overcast nodes
- Network topologies generated from Georgia Tech Internetwork Models
- Five different 600 node graphs

Evaluation – Bandwidth Utilization

- Testing the efficiency of Overcast
- Simulation artifact
  - Backbone routers turned on 1st in 600 node case
  - Put at top of tree

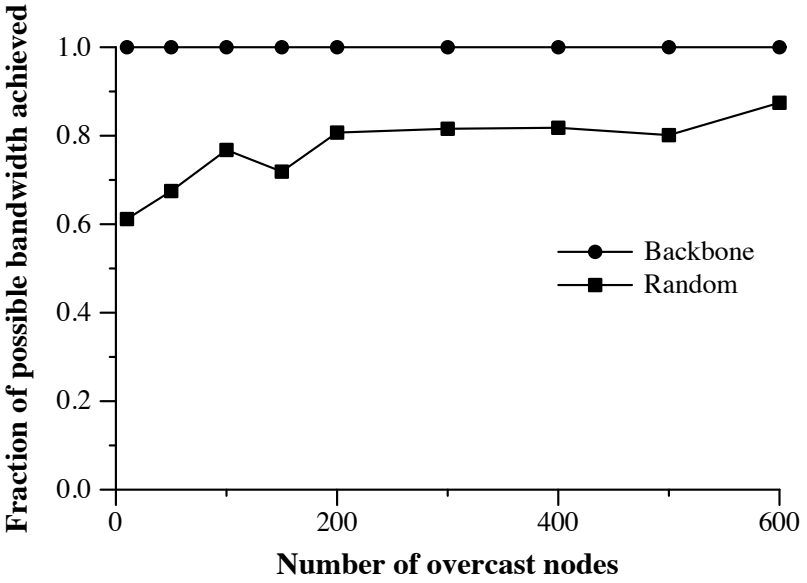


Figure 3: Fraction of potential bandwidth provided by Overcast.

Evaluation – Efficiency

- Network Load
  - Number of times a particular piece of data must traverse a network link to reach all Overcast nodes
  - IP Multicast takes one less than number of nodes
- Ratio of Overcast network load to IP Multicast load

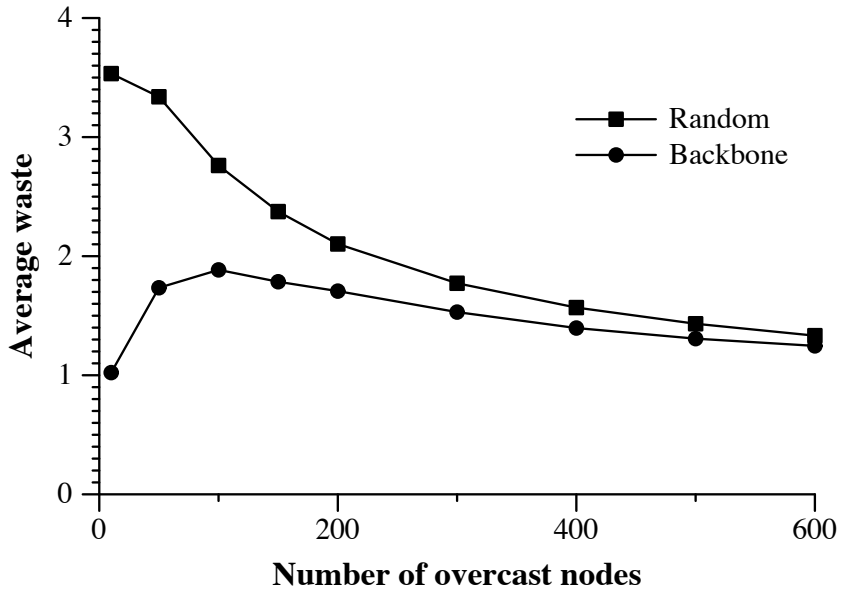


Figure 4: Ratio of the number of times a packet must "hit the wire" to be propagated through an Overcast network to a lower bound estimate of the same measure for IP Multicast.

Evaluation – Resiliency

- Measure convergence time using “round time” as fundamental unit
- Authors expect round time to be approximately 1-2 seconds
- Uses backbone approach

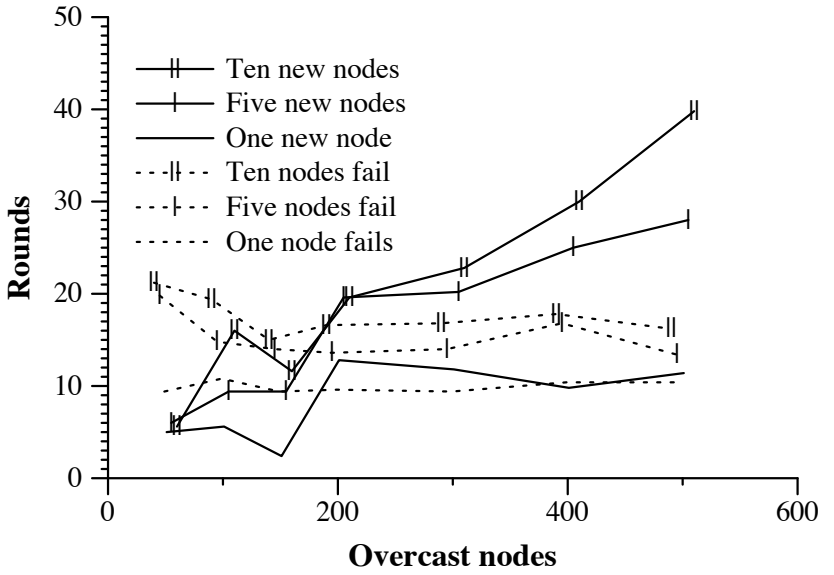


Figure 6: Number of rounds to recover a stable distribution tree as a function of the number of nodes that change state and the number of nodes in the network.

Discussion and Thoughts

- Deployable
- Infrastructure requirements
- Acceptable fault tolerance?
  - Rebuild time
  - Delays due to polling acceptable?

SplitStream: High-Bandwidth Multicast in Cooperative Environments

Authors

- Miguel Castro
  - Microsoft Research
  - Pastry, Scribe
- Peter Druschel
  - Rice University
  - Pastry, Scribe
- Anne-Marie Kermarrec
  - Pastry, Scribe
- Atul Singh
  - Student of Druschel
  - NEC Research

Motivation

- Conventional tree-based multicast not well matched to a cooperative environment

- Only a relatively small number of interior nodes carry the burden of most multicast messages
- Applications
  - Peer exchanging resources in cooperative in environment
  - Decentralized, generic infrastructure

Approach

- Striping content across a forest of interior-node-disjoint multicast trees
  - peers contribute only as much forwarding bandwidth as they receive
- Accommodate different bandwidth capacities
- Focus on low overhead for maintenance

Routing using Pastry

- Each member is given a unique *nodeId* and each multicast group is given a unique *key*
- Nodes maintain a routing table and a leaf set
  - Prefix routing
  - Proximity-aware
  - Leaf-sets are the set of neighboring nodes

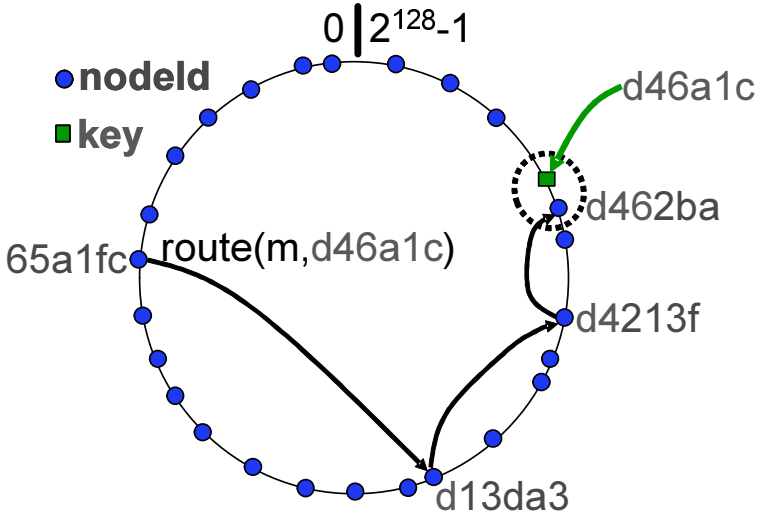


Figure 3: Routing a message from the node with nodeId $65a1fc$ to key $d46a1c$. The dots depict the nodeIds of live nodes in Pastry's circular namespace.

Multicast and Group Membership with Scribe

- Scribe is an application-level multicast infrastructure
- Built upon Pastry's peer-to-peer routing substrate
- Group management
- Multicast tree creation
  - Joins Pastry routes of each group member to common rendezvous point (groupID)

Design – Stripes and Forest Construction

- Content is divided into *k* stripes
- Each stripe is given its own Scribe multicast tree
- Pastry provides prefix routing to ensure that each node is an interior node in one tree.
- Inbound constraints met, need addition mechanism for outbound

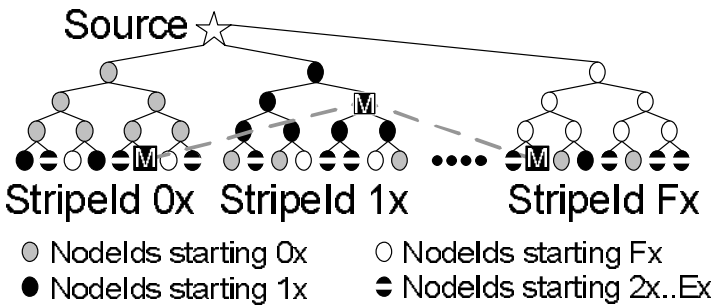
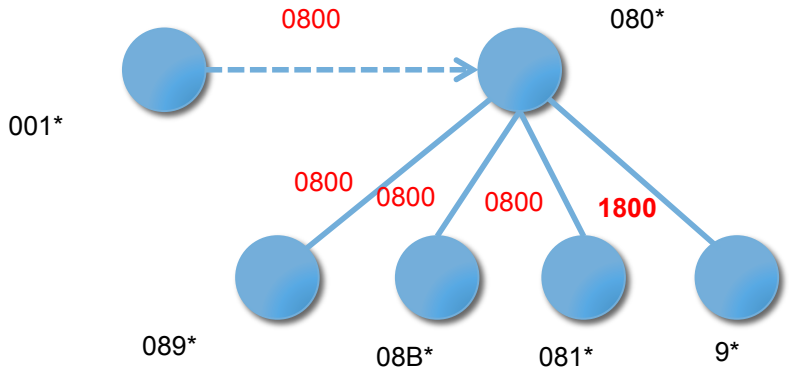


Figure 4: SplitStream's forest construction. The source splits the content and multicasts each stripe in its designated tree. Each stripe's *stripeId* starts with a different digit. The nodeIds of interior nodes share a prefix with the stripeId, thus they must be leaves in the other trees, e.g., node M with a nodeId starting with 1 is an interior node in the tree for the stripeId starting with 1 and a leaf node in other trees.

Design – “Push-down”

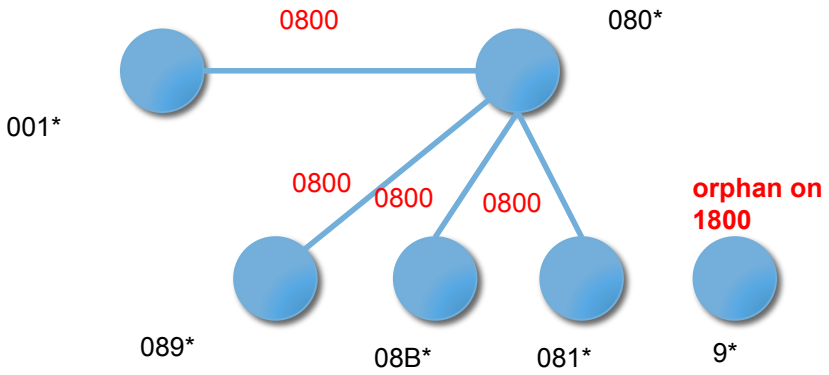
- Scribe’s “push-down” process involves recursively pushing the prospective child down to children.
- Modified “push-down”
  - Prospective child always admitted,
  - Looks to reject a child whose stripeId do not share prefix (or shortest match) with local node’s nodeId



Node 001* requests to join stripe 0800

Design – “Push-down”

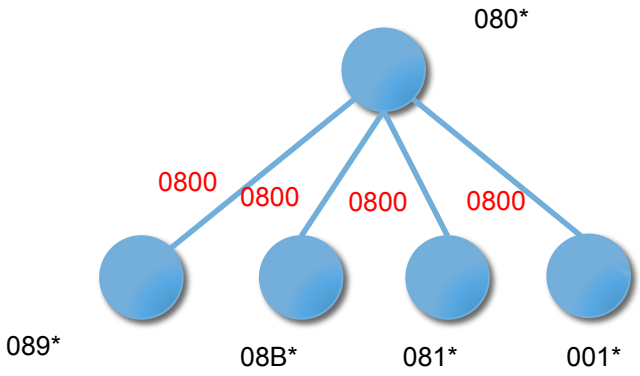
- Scribe’s “push-down” process involves recursively pushing the prospective child down to children.
- Modified “push-down”
  - Prospective child always admitted,
  - Looks to reject a child whose stripeId do not share prefix (or shortest match) with local node’s nodeId



Node 080* takes 001* as a child and drop 9*

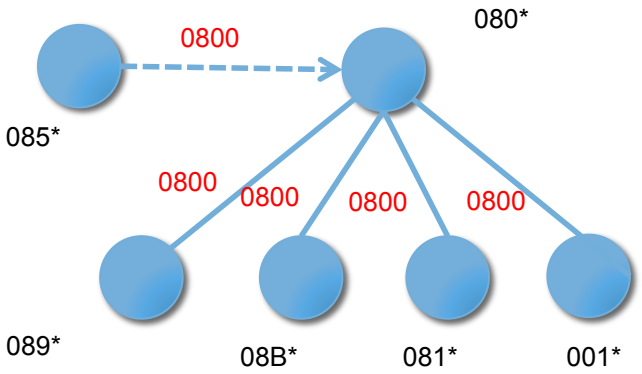
Design – “Push-down”

- Scribe’s “push-down” process involves recursively pushing the prospective child down to children.
- Modified “push-down”
  - Prospective child always admitted,
  - Looks to reject a child whose stripeId do not share prefix (or shortest match) with local node’s nodeId



Design – “Push-down”

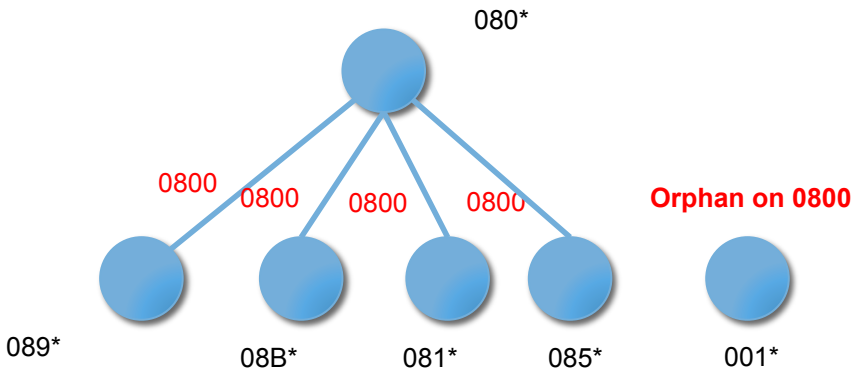
- Scribe’s “push-down” process involves recursively pushing the prospective child down to children.
- Modified “push-down”
  - Prospective child always admitted
  - Looks to reject a child whose stripeId do not share prefix (or shortest match) with local node’s nodeId



Node 085* requests to join stripe 0800

Design – “Push-down”

- Scribe’s “push-down” process involves recursively pushing the prospective child down to children.
- Modified “push-down”
  - Prospective child always admitted,
  - Looks to reject a child whose stripeId do not share prefix (or shortest match) with local node’s nodeId



Node 080* takes 085* as a child and drops 001*, since it has shorter prefix match

Design – Spare Capacity Group

- Used when an orphaned node can't locate a parent
- Nodes have spare forwarding capacity
- Checks for cycles

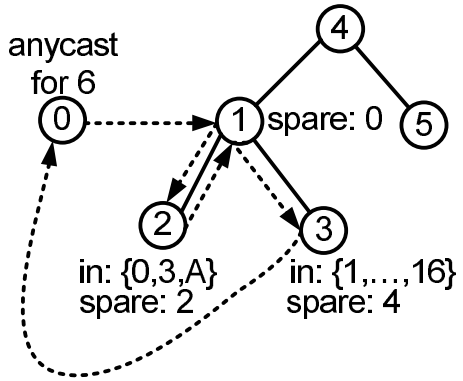


Figure 6: Anycast to the spare capacity group. Node 0 anycasts to the spare capacity group to find a parent for the stripe whose identifier starts with 6. The request is routed by Pastry towards the root of the spare capacity group until it reaches node 1, which is already in the tree. Node 1 forwards the request to node 2, one of its children. Since 2 has no children, it checks if it can satisfy node 0's request. In this case, it cannot because it does not receive stripe 6. Therefore, it sends the request back to node 1 and node 1 sends it down to its other child. Node 3 can satisfy 0's request and replies to 0.

Design – Correctness

- Argue that forests can be constructed with high probability given the sufficient condition for feasibility (on right)

CONDITION 2. *A sufficient condition for the feasibility of forest construction is for Condition 1 to hold and for all nodes whose forwarding capacity exceeds their desired indegree to receive or originate all k stripes, i.e.,*

$$\forall i : C_i > I_i \Rightarrow I_i + T_i = k. \quad (2)$$

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

Evaluation – Goals

- What is the overhead of maintaining the forest?
- How well does its multicast perform compared to IP and Scribe?
- How resilient is SplitStream to failures?

Evaluation – Experimental Setup

- Network Simulation
  - Packet level, discrete event simulator
- Network Topologies
  - GATech
    - Transit-stub topology
    - 5050 hierarchically arranged routers
  - Mercator
    - Topology model based on measurements of Internet using Mercator System
    - 102,639 routers
  - CorpNet
    - Generated using measurements from Microsoft corporate network
    - 298 routers

Evaluation – Forest Construction Overhead

- Efficiency metrics
  - Node stress
  - Link stress
- Point (x,y) indicates that a fraction y of all nodes have stress <= x

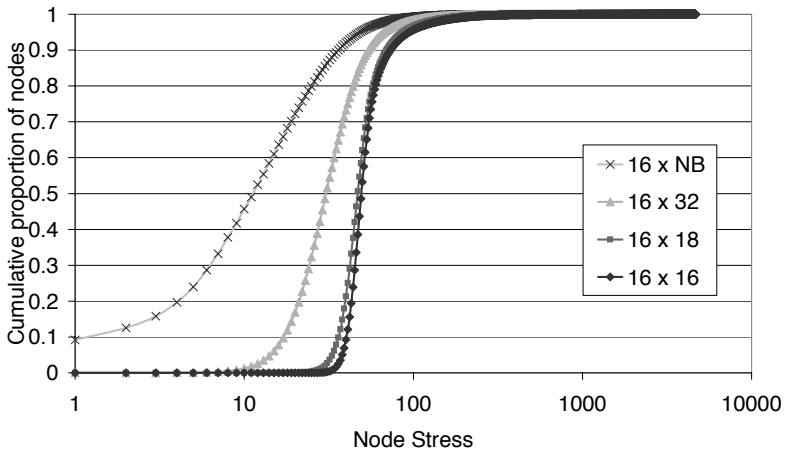


Figure 8: Cumulative distribution of node stress during forest construction with 40,000 nodes on GATech.

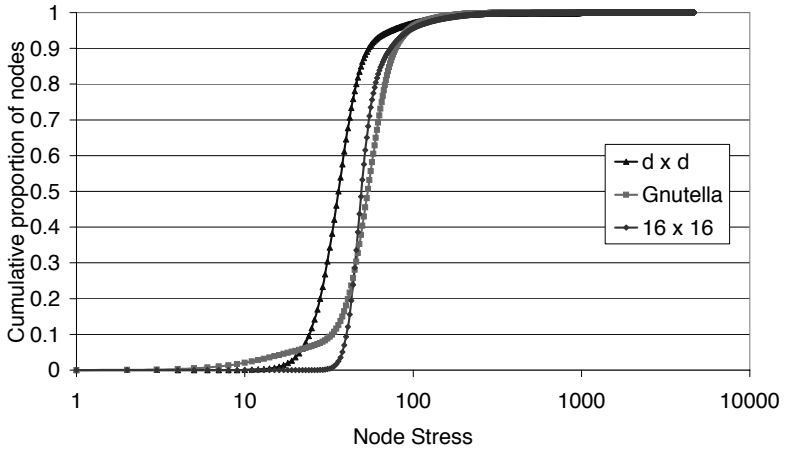
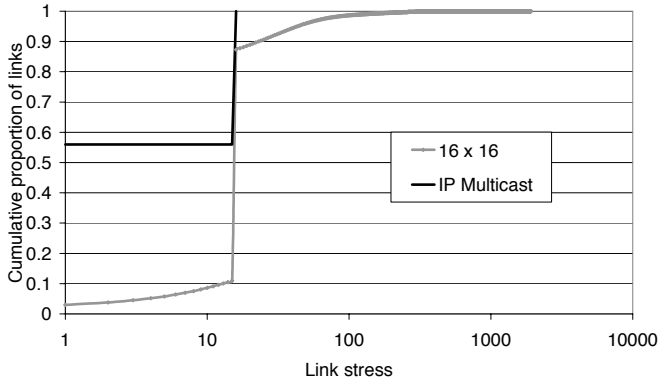


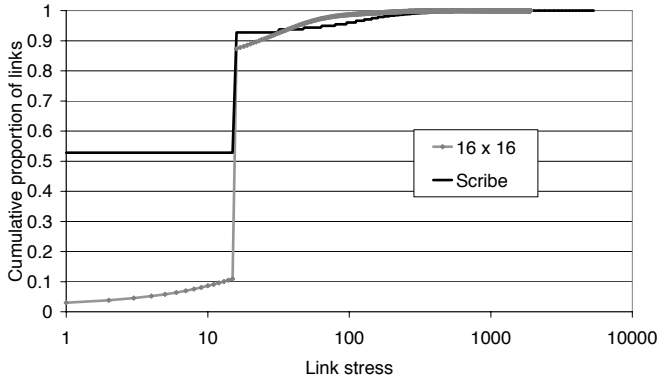
Figure 9: Cumulative distribution of node stress during forest construction with 40,000 nodes on GATech.

Evaluation – Multicast Performance

- Compared performance to Scribe and IP Multicast



(a) SplitStream vs IP



(b) SplitStream vs Scribe

Figure 12: Cumulative distribution of link stress during multicast with 40,000 nodes on GATech.

Evaluation – Resilience to Node Failures

- Tested performance under catastrophic failure
- Failed 2,500 of 10,000 nodes 10 seconds into simulation

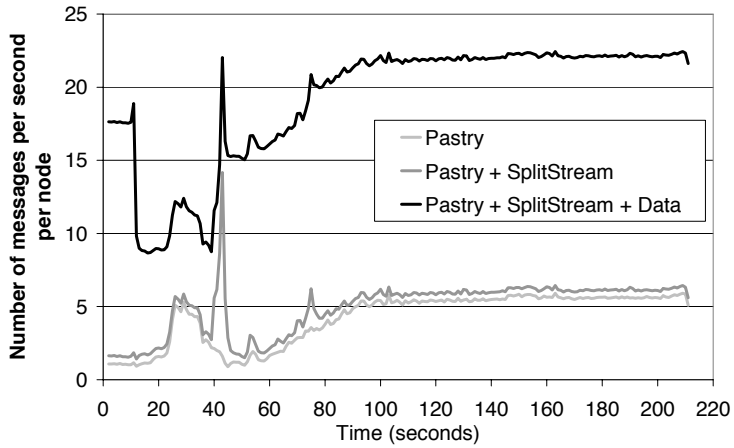


Figure 16: Number of messages per second per node when 25% out of 10,000 nodes fail on GATech.

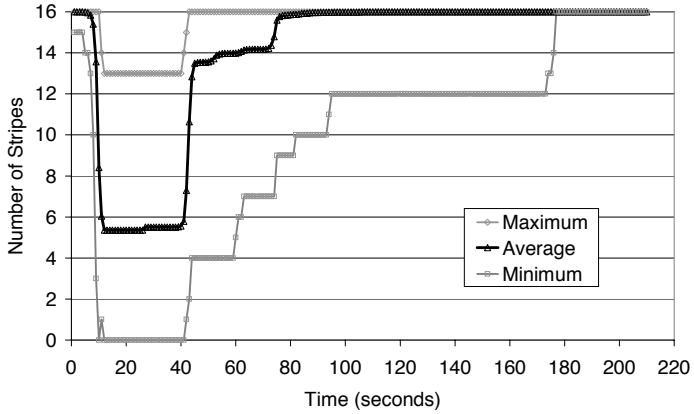


Figure 15: Maximum, average, and minimum number of stripes received when 25% out of 10,000 nodes fail on GATech.

Discussion

- Practical/Deployable?
  - Compare to Overcast
  - Complexity of mechanism
- Is it safe to assume there is enough bandwidth to carry each stripe?
- Will all clients be cooperative and trusting?
  - What happens with misbehaving clients?
- Does SplitStream actually deliver on its promises?