Scribe: A large-scale and decentralized application-level multicast infrastructure

Paper by: Miguel Castro, Peter Druschel,
Anne-Marie Kermarrec,
Antony Rowstron,
IEEE JSAC,2002.
Presented by: Sankardas Roy

Acknowledgement :

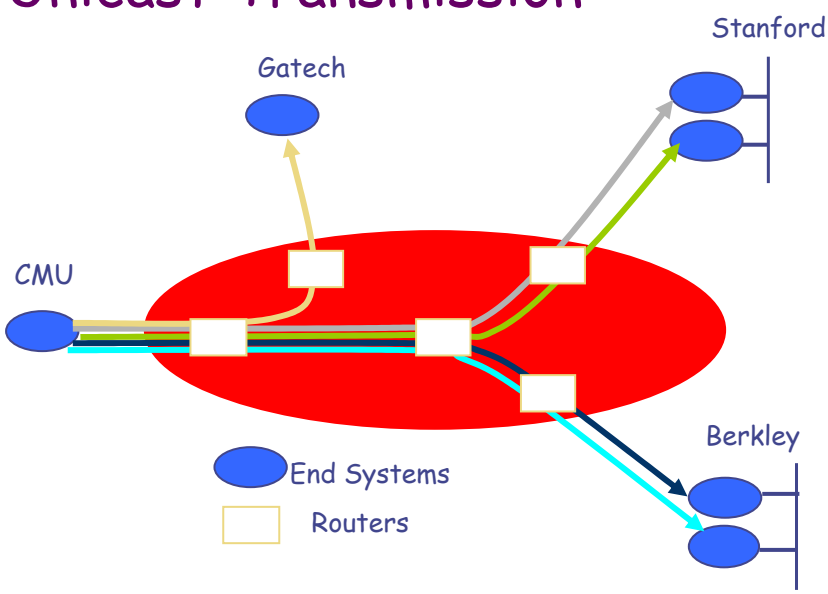
Wang Ting and Wei Ran, unsw.edu.au

Yang-hua Chu, Sanjay Rao and Hui Zhang, CMU

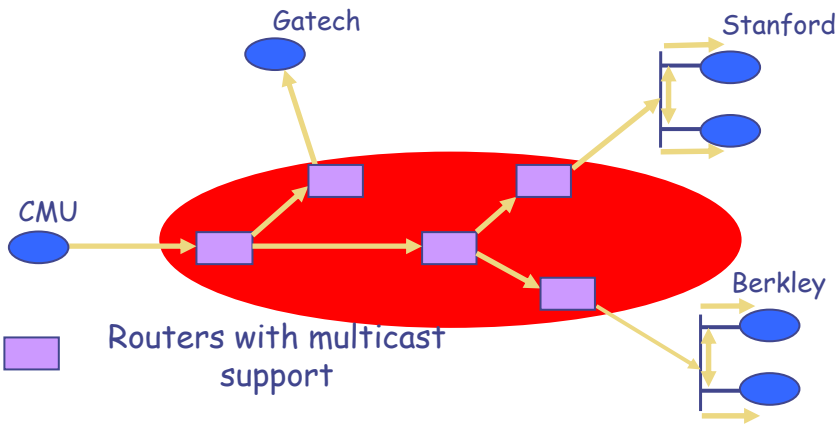
Outline

- Introduction
- Pastry
- The design of Scribe
- Experimental evaluation
- Conclusion

Unicast Transmission



IP Multicast



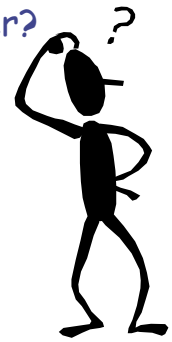
- No duplicate packets
- Highly efficient bandwidth usage

Key Concerns with IP Multicast

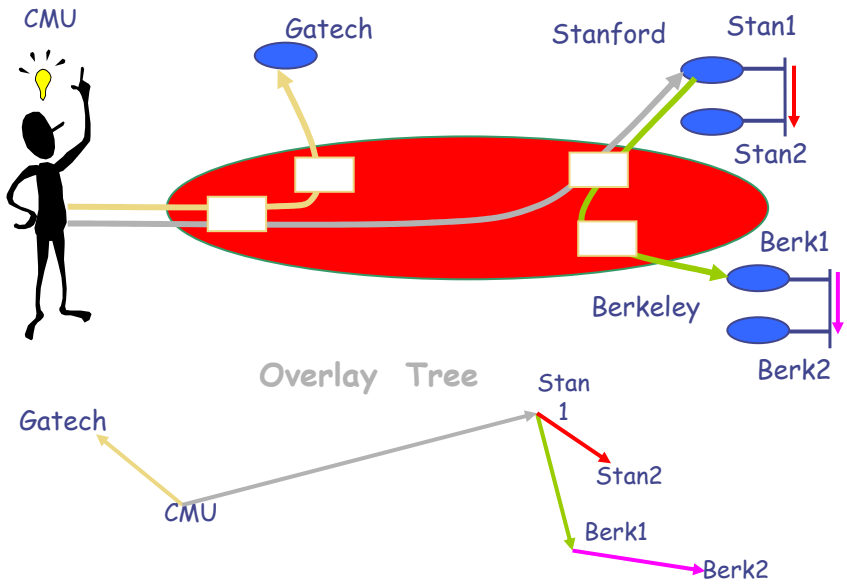
- Scalability with number of groups
  - Routers maintain per-group state
  - Analogous to per-flow state for QoS guarantees
- Supporting higher level functionality is difficult
  - IP Multicast: best-effort multi-point delivery service
  - End systems responsible for handling higher level functionality
  - Reliability and congestion control for IP Multicast complicated
- Deployment is difficult and slow
  - ISP's reluctant to turn on IP Multicast

An important question…

- Can we achieve
  efficient multi-point delivery,
  without support from the IP layer?

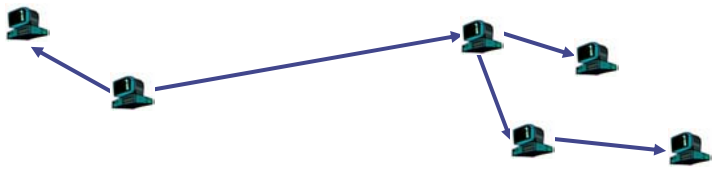


Overlay Multicast

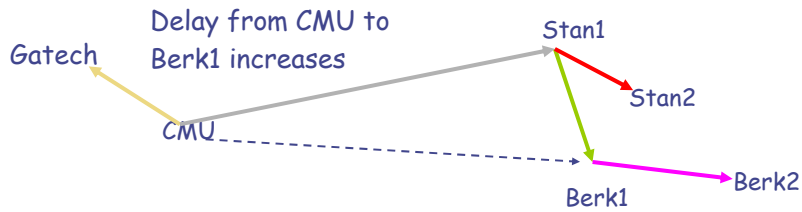


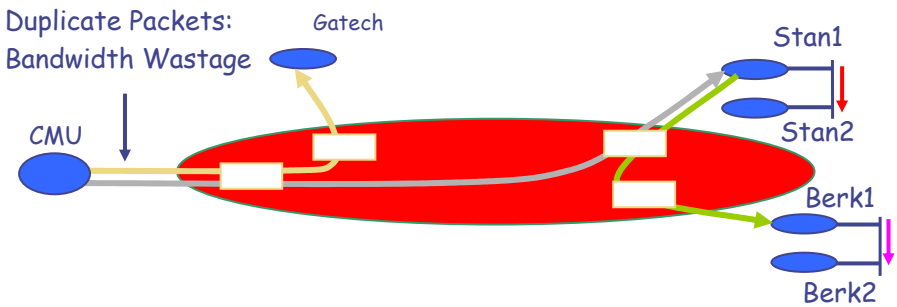
Potential Benefits

- Scalability
  - Routers do not maintain per-group state
  - End systems do, but they participate in very few groups
- Easier to deploy
- Potentially simplifies support for higher level functionality
  - Leverage computation and storage of end systems
  - For example, for buffering packets, ACK aggregation
  - Leverage solutions for unicast congestion control and reliability



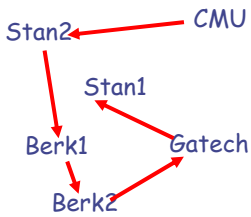
Performance Concerns



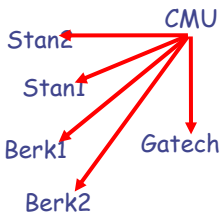


What is an efficient overlay tree?

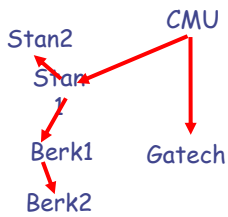
- The delay between the source and receivers is small
- Ideally,
  - The number of redundant packets on any physical link is low



High latency



High degree (unicast)



"Efficient" overlay

Scribe

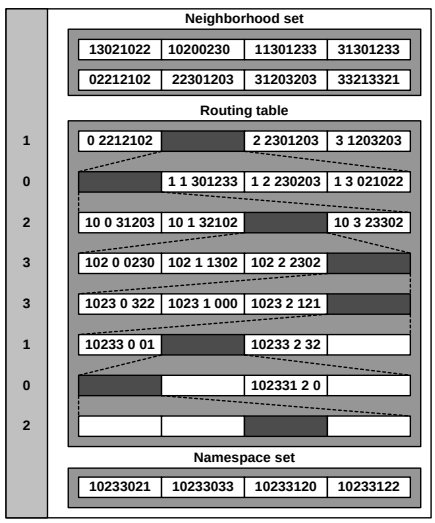
- **Scribe**
  - A large-scale, decentralized application-level multicast infrastructure built on top of Pastry
  - Scaling across a wide range of groups and group sizes
- **Pastry**

  A scalable, self-organizing, robust object location and routing substrate

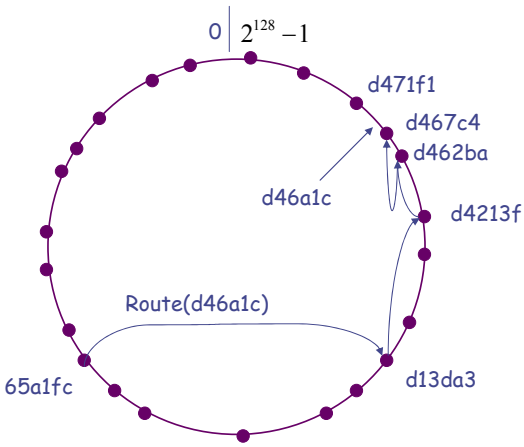
Pastry

- Given a message and a key, Pastry routes the message to the node with the nodeId that is numerically closest to the key in less than $\log_2{}^b N$ steps on average.
- Pastry routing scheme:
  - routing table
  - each entry refers to one of potentially many nodes whose nodeId have the appropriate prefix
  - Forward to the node whose nodeId shares with the key a prefix at least one digit longer than current nodeId do.

Pastry – routing table



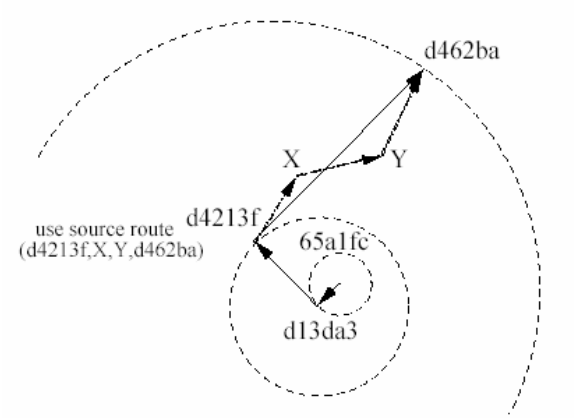
Pastry – message routing



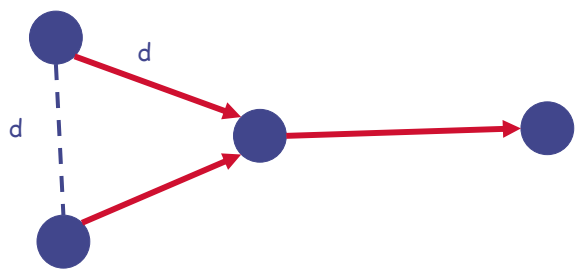
Pastry – locality

- Two of Pastry's locality properties related to Scribe:
  - Short routes property:
    Concerns the total distance that the messages travel along Pastry route, in each step a message is routed to the nearest node with a longer prefix match.
  - Route convergence property:
    Concerns the distance traveled by two messages sent to the same key before converging.

Short route property



Route convergence property



Pastry API

- Pastry exports:
  - nodeId = *pastryInit(Credentials)*
  - *route(msg,key)*
  - *send(msg,IP-addr)*
- Applications based upon Pastry exports:
  - *deliver(msg,key)*
  - *forward(msg,key,nextId)*
  - *newLeafs(leafSet)*

Scribe

- API exported:
  - *create(credentials, groupId)*
  - *join (credentials, groupId, messageHandler)*
  - *leave(credentials, groupId)*
  - *multicast(credentials, groupId, message)*

Scribe - implementation

- Scribe software provides the *forward* and *deliver* methods to be invoked by Pastry
- *forward*
  - called whenever a Scribe message is routed through a node
- *deliver*
  - called when a message arrives at the final destination or when a message was addressed to a node by IP address.

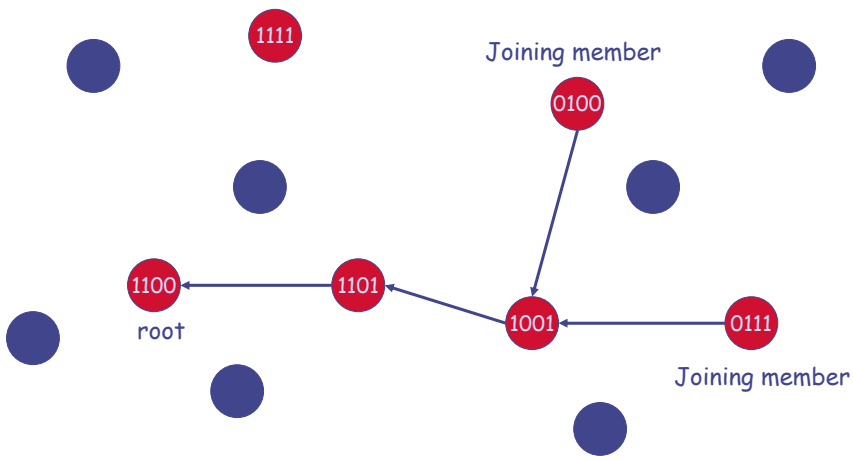
Scribe – group creation

- Each group has a groupId
- Rendezvous point
  - the Scribe node with a nodeId numerically closest to the groupId
  - is the root of the multicast tree
- Methods to create a group
  - route a create message
  - deliver the message to the node with appropriate nodeId
  - add the group to the group list

Scribe – group joining

- *Forwarders*:
  - Scribe nodes that are part of multicast tree
  - maintain a children table
- Joining group:
  - sends JOIN message
  - routed toward the rendezvous point
  - accepts the node as a child

Model of joining mechanism



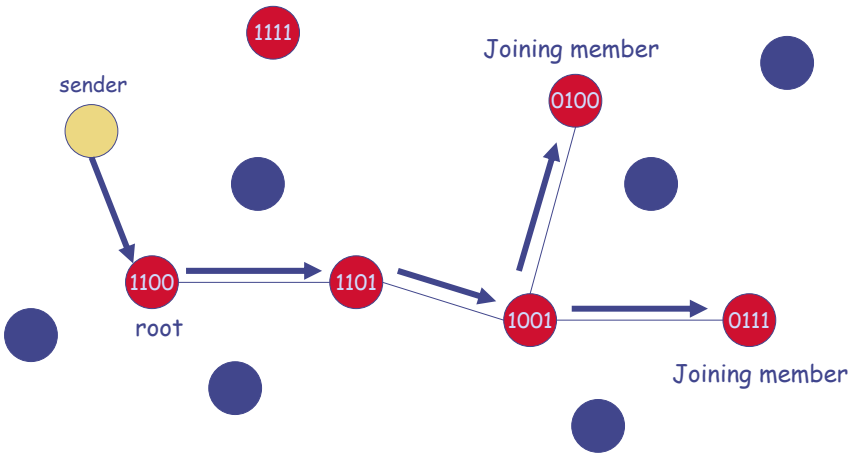
Scribe – leaving group

- Record locally that it left the group
- If no other entries, sends leave message to its parent
- Repeats until a node is reached that still has entries in the children table after removing the leaving node

Scribe – multicast message

- Use Pastry to locate the rendezvous point
- Multicast messages are disseminated from the rendezvous point along the multicast tree to the group
- The locality properties of Pastry ensure that the multicast tree can be used to disseminate messages efficiently (short routes and route convergence)

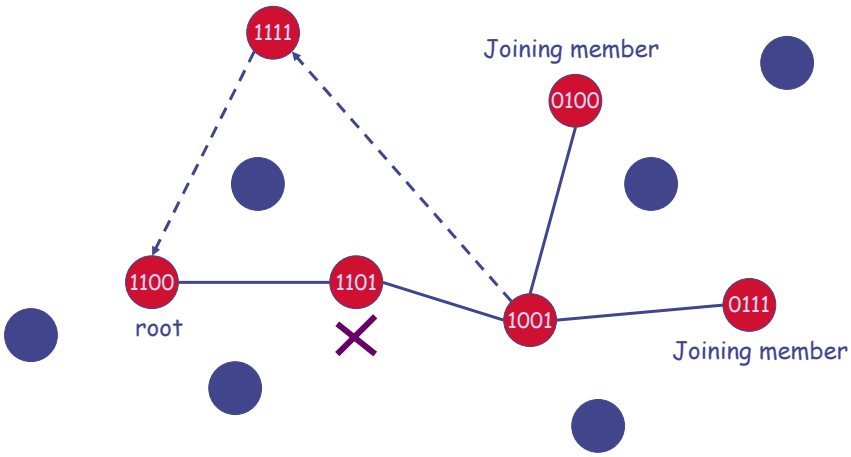
Model of multicast message



Scribe – reliability

- Uses TCP to disseminate message and perform flow control
- Uses Pastry to repair the multicast tree when a forwarder fails
  - for forwarders: child calls Pastry to route a JOIN message to a new parent when it fails to receive heartbeat messages
  - for roots: the state associated with the rendezvous point is replicated across k closest nodes to the root node
  - children join a new root by Pastry routing the JOIN message

Model of repairing



Experimental Setup I

- Network: transit stub model
  - 5050 router
  - Tool: Georgia Tech random graph generator

Experimental Setup II

- Group number : 1500
- Scribe nodes : 100,000
- Minimum group size: 11
- Maximum group size: 100,000
- Group size varies according to Zipf's law
- Compare with IP multicast
  - *Delay* to deliver messages to group member
  - *Stress* on each node
  - *Stress* on each physical link
  - Scalability with many small groups

Delay Penalty

- Compare the delay to multicast messages using Scribe and IP multicast

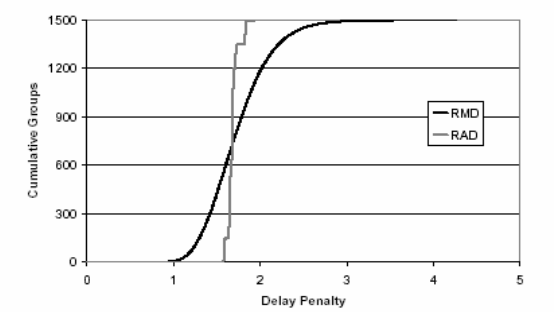


Fig. 7. Cumulative distribution of delay penalty relative to IP multicast per group (average standard deviation was 62 for RAD and 21 for RMD).

Node Stress I

- Measure the number of groups with non-empty children tables and the number of entries in children tables in each Scribe node

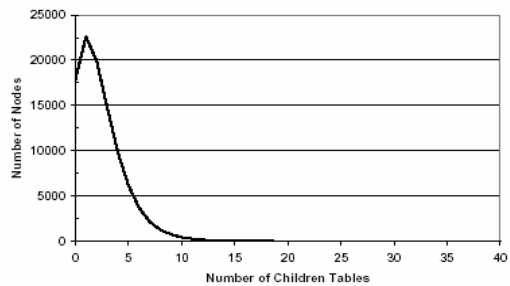


Fig. 8. Number of children tables per Scribe node (average standard deviation was 58).

Node Stress II

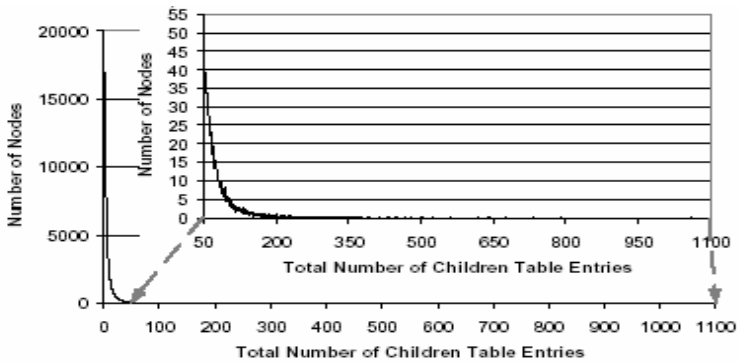


Fig. 9. Number of children table entries per Scribe node (average standard deviation was 3.2).

Link Stress

- Compare the stress imposed by Scribe and IP multicast on each directed link in the network topology

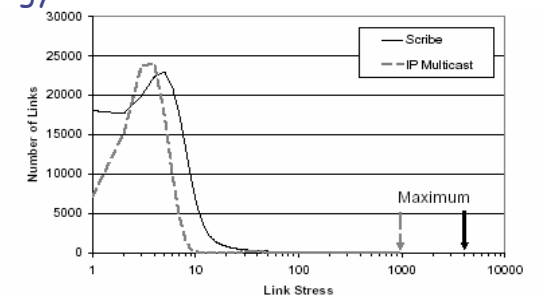


Fig. 10. Link stress for multicasting a message to each of 1,500 groups (average standard deviation was 1.4 for Scribe and for 1.9 for IP multicast).

Bottleneck Remover I

- Bottleneck due to inequality of node's capacity
- The bottleneck remover algorithm allows nodes to bound the amount of multicast forwarding they do by offloading children to other nodes
- When a node detect that it is overloaded
  - Select the group that consumes the most resources (most children)
  - Choose the child of the group that is farthest away
  - Tell the child to join his sibling that provides the smallest combined delay

Bottleneck Remover II

- The distribution of the number of children table entries per node

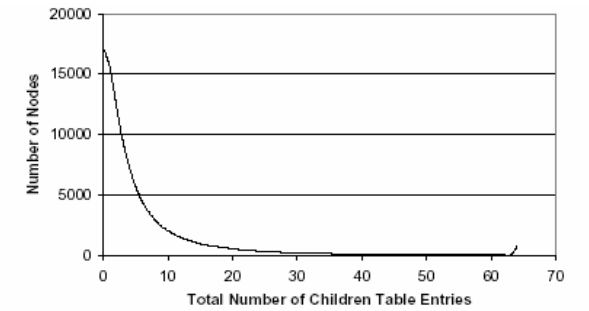


Fig. 11. Number of children table entries per Scribe node with the bottleneck remover (average standard deviation was 57).

Scalability with many small groups

- 50,000 Scribe nodes
- 30,000 groups with 11 members each
- Distribution of children tables and children table entries per node

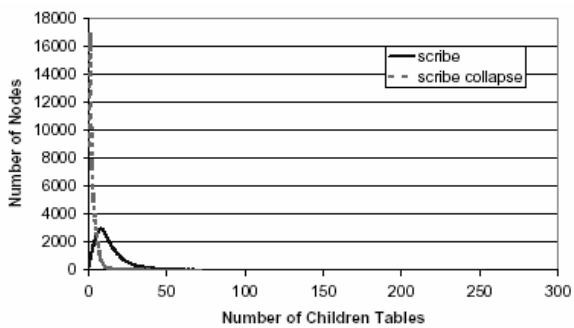


Fig. 12. Number of children tables per Scribe node.

Scalability with many small groups II

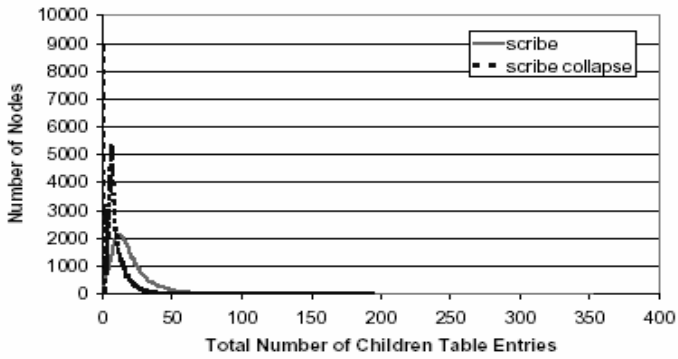


Fig. 13. Number of children table entries per Scribe node.

Scalability with many small groups III

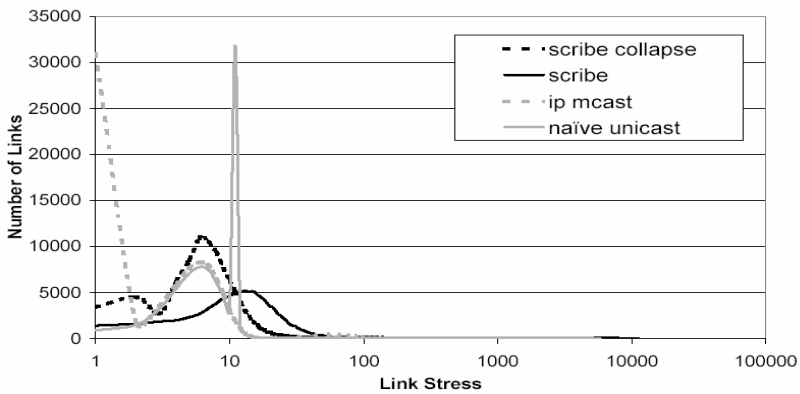


Fig. 14. Link stress for multicasting a message to each of 30,000 groups.

Conclusions

- Scribe is a large –scale and fully decentralized application-level multicast infrastructure built on top of Pastry
- Scribe scales well
- Scribe is able to efficiently support a large number of nodes, groups, and a wide range of group sizes

An Evaluation of scalable Application-level Multicast Built Using Peer-to-peer Overlays

- **Paper by: Miguel Castro et al, INFOCOM,03.**
- **Presented by: Sankardas Roy**

Main topic of the Paper

- This paper evaluates tree-based and flooding-based multicast using two different types of structured overlay:

  1) overlays which use a form of generalized hypercube routing, e.g., Chord, Pastry and Tapestry,

  2) overlays which use a numerical distance metric to route through a Cartesian hyper-space, e.g., CAN.

- Pastry and CAN are chosen as the representatives of each type of overlay.

Multicast Types

- Flooding
  - Each multicast session (or group) form a mini-CAN or mini-Pastry
  - Flood msgs to neighbors
- Tree based
  - Root will be the source node
  - When new member joins, JOIN msg is routed to the route.
  - Intermediate nodes set up routing table

Experimental setup

- Network: transit stub model, 5050 router, Tool: Georgia Tech random graph generator
- The 1st set of experiments run with a single multicast group and all the overlay nodes (total 80,000) were members of the group.
- The second set of experiments run with a large number of groups (1500) and with a wide range of membership sizes(according to Zipf's law).

CAN FLOODING

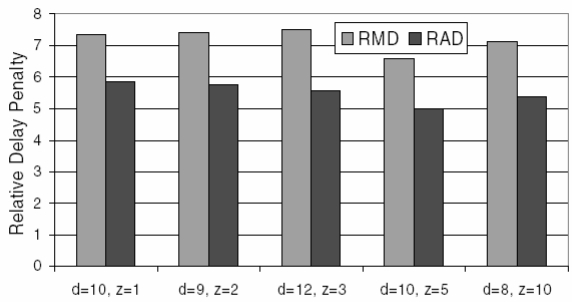


Fig. 1. Relative delay penalty for CAN flooding with different values of d and z.

CAN FLOODING

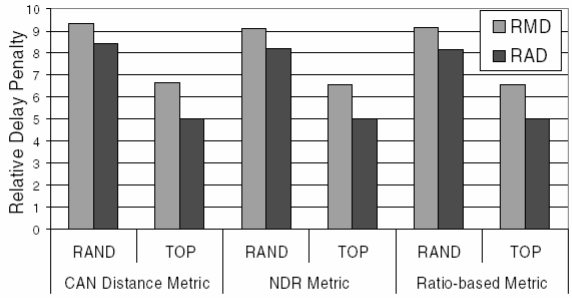


Fig. 2. Relative delay penalty for CAN flooding with and without topology-aware optimizations using a d=10,z=5 CAN configuration.

CAN FLOODING

Configuration	d=10 z=1	d=9 z=2	d=12 z=3	d=10 z=5	d=8 z=10
State size	18	29	38	59	111
Joining phase					
Max	91615	149341	197977	309212	416361
Average	154	183	219	281	431
Flooding phase					
Max	1958	1595	1333	985	631
Average	3.49	3.27	2.93	2.73	2.69

TABLE II

LINK STRESS FOR FLOODING IN CAN.

CAN TREE-BASED

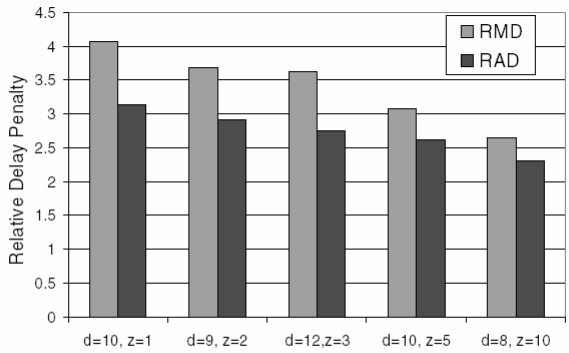


Fig. 3. Relative delay penalty for CAN tree-based multicast with different values of d and z.

CAN TREE-BASED

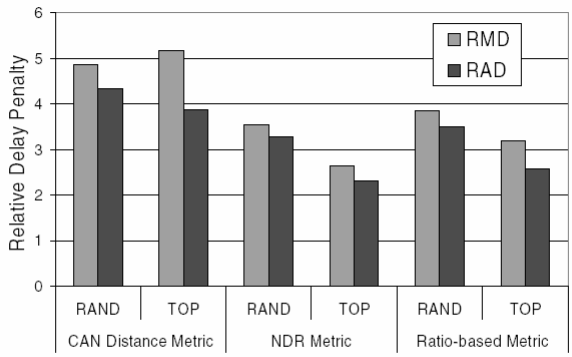


Fig. 4. Relative delay penalty for CAN tree-based multicast with and without topology-aware optimizations using a d=8,z=10 CAN configuration.

CAN TREE-BASED

Configuration	d=10 z=1	d=9 z=2	d=12 z=3	d=10 z=5	d=8 z=10
State size	18	29	38	59	111
Max	323	220	198	184	225
Average	1.69	1.49	1.42	1.37	1.36

TABLE III

LINK STRESS FOR CAN TREE-BASED MULTICAST.

PASTRY FLOODING

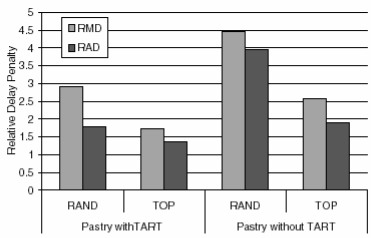


Fig. 6. Relative delay penalty for Pastry flooding with and without topology-aware optimizations for $b = 4$.

	with TART		without TART	
	RAND	TOP	RAND	TOP
Max	6801.4	65.4	2119.0	61.0
Average	4.3	1.4	4.6	1.4

TABLE V

LINK STRESS FOR PASTRY FLOODING WITH AND WITHOUT TOPOLOGY-AWARE OPTIMIZATIONS FOR $b = 4$.

PASTRY TREE-BASED

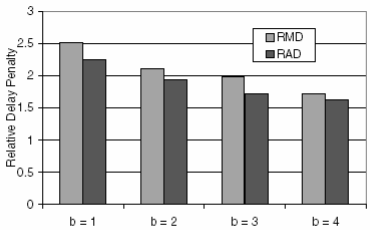


Fig. 7. Relative delay penalty for Pastry tree-based multicast for different values of b.

PASTRY TREE-BASED

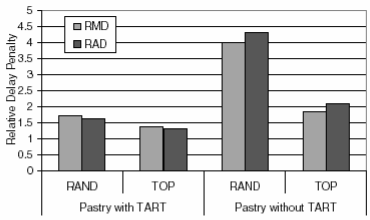


Fig. 8. Relative delay penalty for Pastry tree-based multicast with and without topology-aware optimizations for $b = 4$.

	with TART		without TART	
	RAND	TOP	RAND	TOP
Max	286.2	22,073.8	1,910.6	23,999.4
Average	1.17	3.34	3.87	3.90

TABLE VII
LINK STRESS FOR PASTRY TREE-BASED MULTICAST WITH AND WITHOUT TOPOLOGY-AWARE OPTIMIZATIONS FOR $b = 4$.

More than one groups (1500 groups)

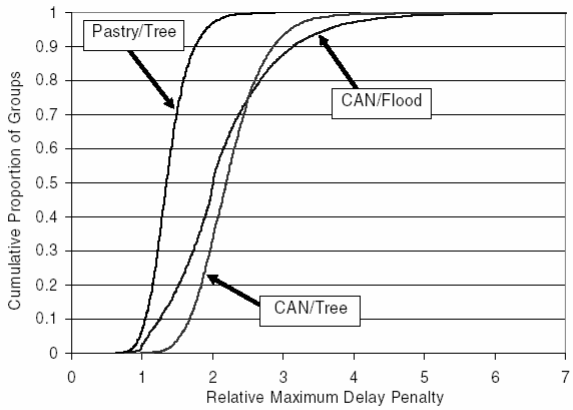


Fig. 9. CDF for RMD for 1500 concurrent multicast groups with localized group members.

Questions?