# HyperCast - Towards Super-Scalable Multicast Communication

**Jörg Liebeherr**

**Department of Electrical Engineering**
**Polytechnic University**
jorg@catt.poly.edu

## Outline

- Motivation and Background
- Hypercube Control Topology
- Evaluation
- HyperCast Protocol
- Conclusions

## Context

**"Super-Scalable" Multicast**:

- Multicast Applications with hundreds of thousand simultaneous users

**Symmetric Multicast**

- Every user can generate traffic

## Background I

- Internet multi-user applications use IP Multicast
- IP Multicast:
  - Connectionless, best-effort service
  - No ordering or reliability guarantees
- However, multi-user applications need
  - Error control
  - Rate control
  - Ordering mechanisms

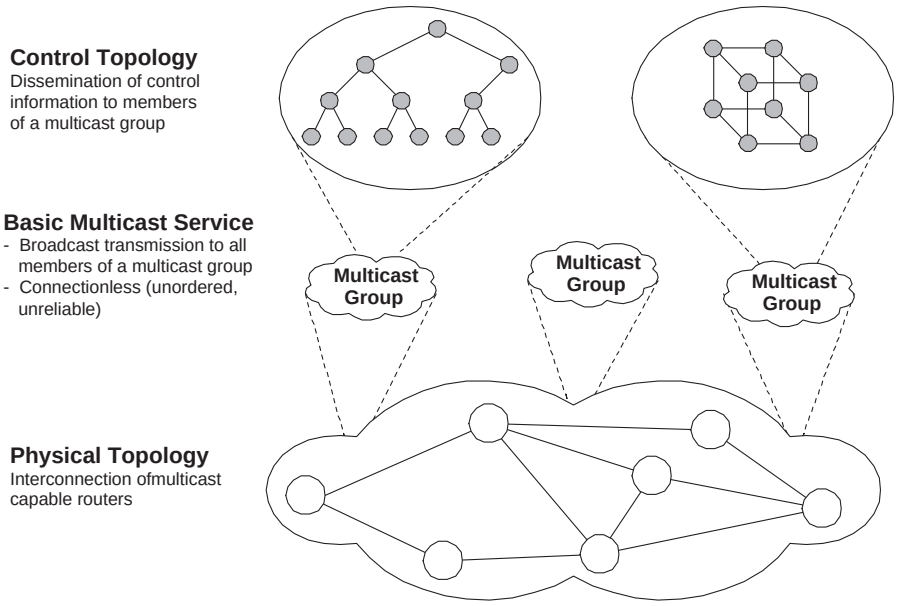
## Background II

- Add protocol mechanisms above IP Multicast that provide services for error, rate control, ordering
- Such mechanisms require feedback between senders and receivers

$\Longrightarrow$ **"Implosion Problem:"** Feedback packets from receivers overwhelm the sender

$\Longrightarrow$ *Solution:* Provide a **Control Topology** to manage flow of control information

## The Multicast Framework



## Control Topologies I

*No Control Topology*

- *Nack Suppression* (Ramakrishnan/Jain, XTP, SRM)
  → Suppressing feedback slows down application
- *Correlate control traffic with data traffic* (RTP)
  → Feedback decreases as traffic picks up

## Control Topologies II

*Ring Topologies* (Chang/Maxemchuck, RMP)

- *Requires token management*
- *Bottleneck at the token holding node*

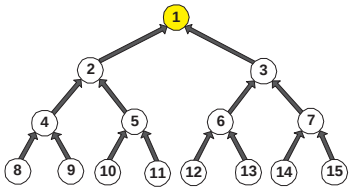
## Control Topologies III

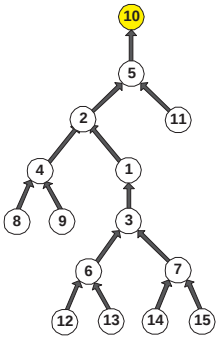
*Tree Topologies* (TMPT/Yavatkar, Holbrook/Cheriton, RMTP, Levine/Garcia-Luna-Aceves)

- *Problem with multiple senders:*
  - *One tree per sender is not practical*
  - *"Rehang" a single tree with different nodes as root* (Shared Tree).

## Shared K-ary Tree

- Use a single tree to transmit control information
- *Re-hang* tree on different roots depending on sender.

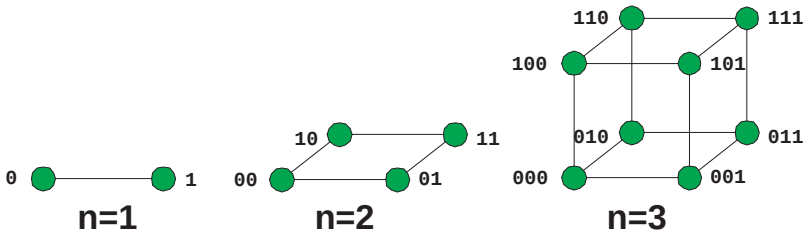


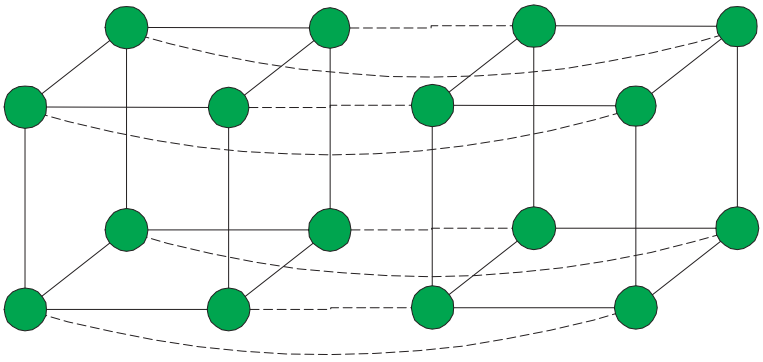
## Introducing the Hypercube

An *$n$-dimensional hypercube* has $N = 2^n$ nodes, where

- Each node labeled $k_n k_{n-1} ... k_1$ $(k_i = 0|1)$
- Two nodes are connected by an edge only if their labels differ in 1 position.




## Hypercube



n=4

## Hypercube as Control Topology

- Arrange the members of the group as nodes of a hypercube
- Embed trees into the hypercube
- Use trees to disseminate control information

*Goals:*

- Trees should have small height
- Hypercube should be as compact as possible

## Ordering Nodes

- Need to keep the dimension of the hypercube minimal.
- Can be done by the $Bin$ ordering.
  - Interpret the node labels as binary numbers.
  - $a = \sum_{i=1}^{n} a_i \cdot 2^{i-1}$ is associated with node $Bin(a) := a_n \ldots a_1$ $(a_i \in \{0, 1\})$.
- Hypercube is kept compact if we ensure that the lowest binary numbers are occupied.
- Not clear how to embed trees.

## Gray Ordering of Nodes

- A Gray code, denoted by $G(\cdot)$, is defined by
  - $G(i)$ and $G(i+1)$ differ in exactly one bit.
  - $G(2^{d-1})$ and $G(0)$ differ in only one bit.
- Gray Code $G(i) := Bin(i) \otimes Bin(i/2)$
- If we ensure that the lowest Gray codes are occupied, compactness is ensured.

## Gray Ordering

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $Bin(i)$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| $G(i)$ | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

## Tree Embedding Algorithm

**Input:** $G(i) := I = I_n \ldots I_2 I_1$, $G(r) := R = R_n \ldots R_2 R_1$

**Output:** Parent of node $I$ in the embedded tree rooted at $R$.

**Procedure Parent** $(I, R)$

**If** $(G^{-1}(I) < G^{-1}(R))$

**Parent** $:= I_n I_{n-1} \ldots I_{k+1}(1 - I_k) I_{k-1} \ldots I_2 I_1$

with $k = \min_i (I_i \neq R_i)$.

**Else**

**Parent** $:= I_n I_{n-1} \ldots I_{k+1}(1 - I_k) I_{k-1} \ldots I_2 I_1$

with $k = \max_i (I_i \neq R_i)$.

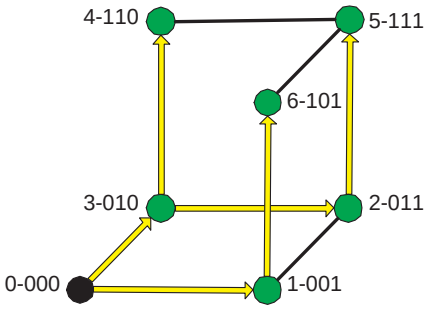
**Endif**

## Properties of Algorithm

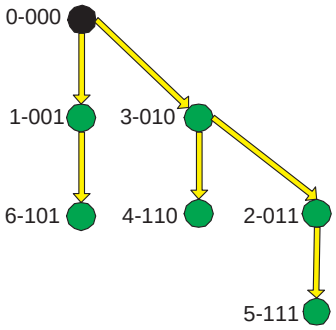
All trees generated by the algorithm satisfy:

- *Completely Contained Trees*
  A spanning tree can be created even in an incomplete hypercube
- *Local Computation of Trees*
  Given the label of the root, every node can compute its parent simply using its own label.

## Examples of Tree Embeddings

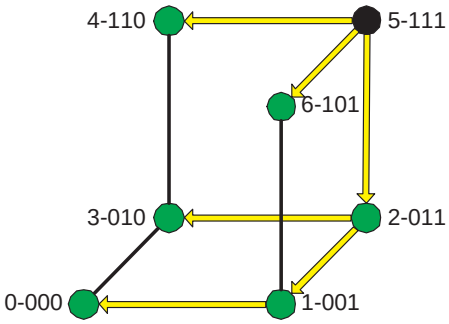



(a) Embedding

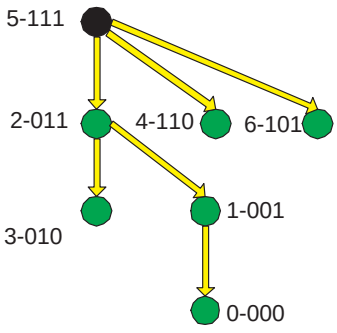



(b) ResultingTree

## Examples of Tree Embeddings




(a) Embedding




(b) ResultingTree

## Comparison of Hypercube with K-ary Tree

**Performance Metrics** ($T_l$ is a control tree with root $l$).

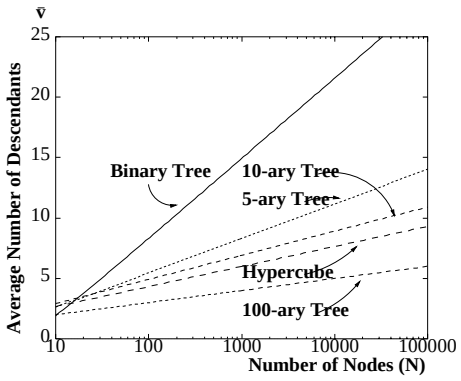
$w_k(T_l)$ := Number of children of node $k \in V$ in tree $T_l$

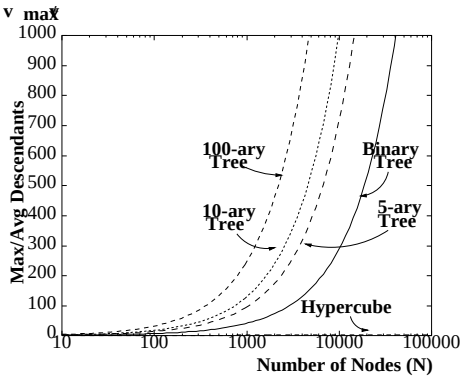
$v_k(T_l)$ := Number of descendants of node $k \in V$ in tree $T_l$ (including node $k$),

$p_k(T_l)$ := Length of the path from node $k$ to root node $l$ in $T_l$.

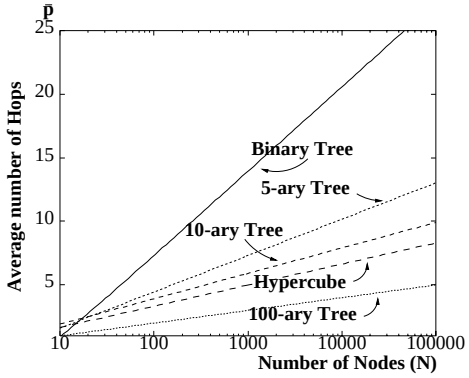
- $w_k, v_k, p_k$ are the averages averaged over all trees
- $\overline{w}$, $\overline{v}$, $\overline{p}$ are the averages of the averages
- $w_{max}, v_{max}, p_{max}$ are the maxima of the averages

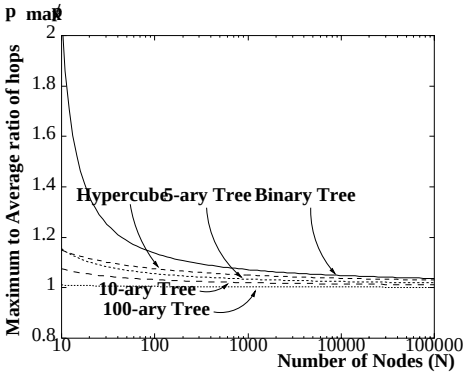
## Results for $\overline{v}$ and $v_{max}/\overline{v}$



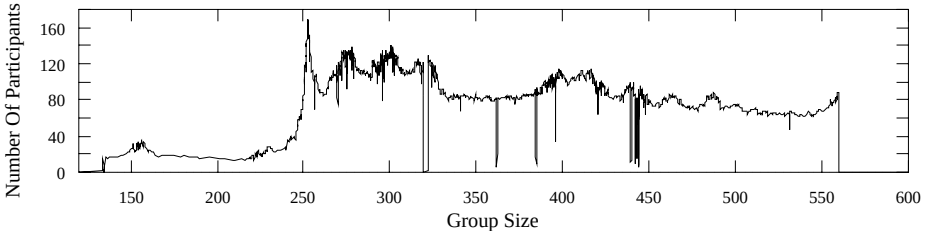
## Results for $\overline{p}$ and $p_{max}/\overline{p}$



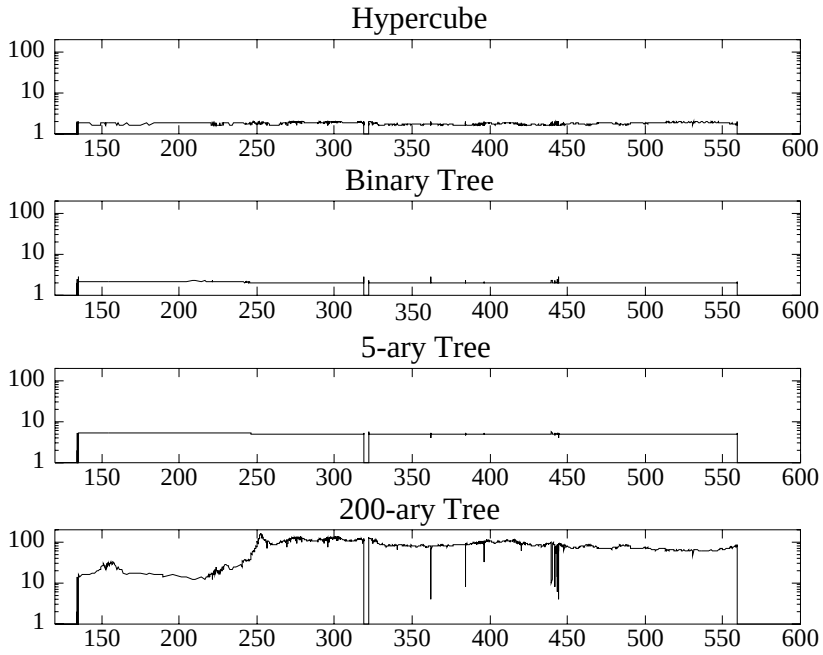




## Empirical Evaluation

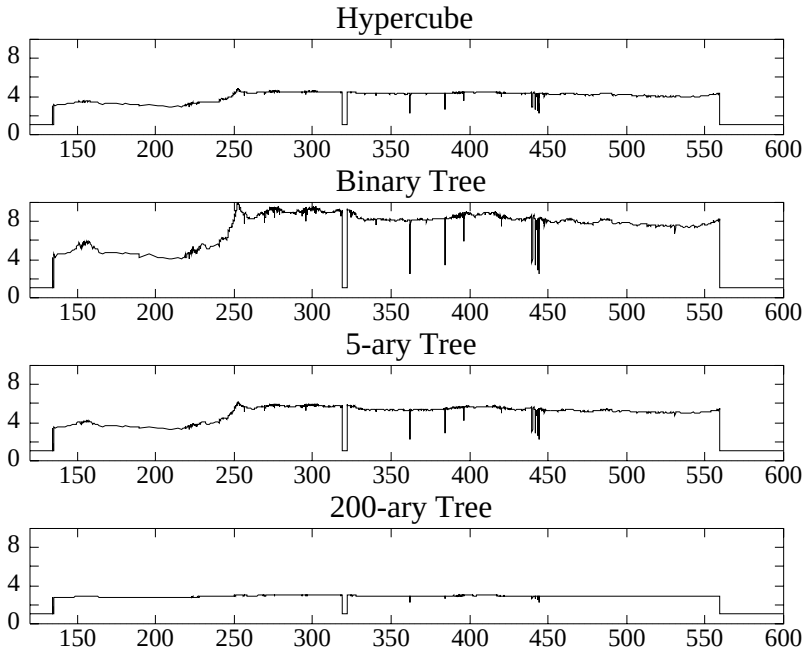
- Study the dynamics of performance metrics for actual multicast groups.
- Traces of an MBONE session obtained from GA Tech: *NASA's STS-80C space shuttle mission* from 11/8/96–12/4/96 .



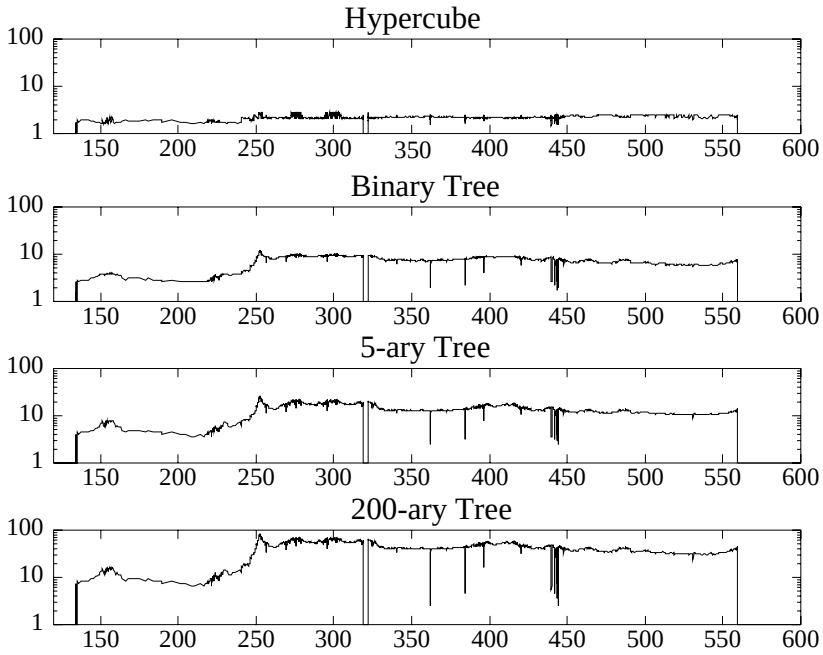
## Results for $\overline{w}$ and $w_{max}/\overline{w}$



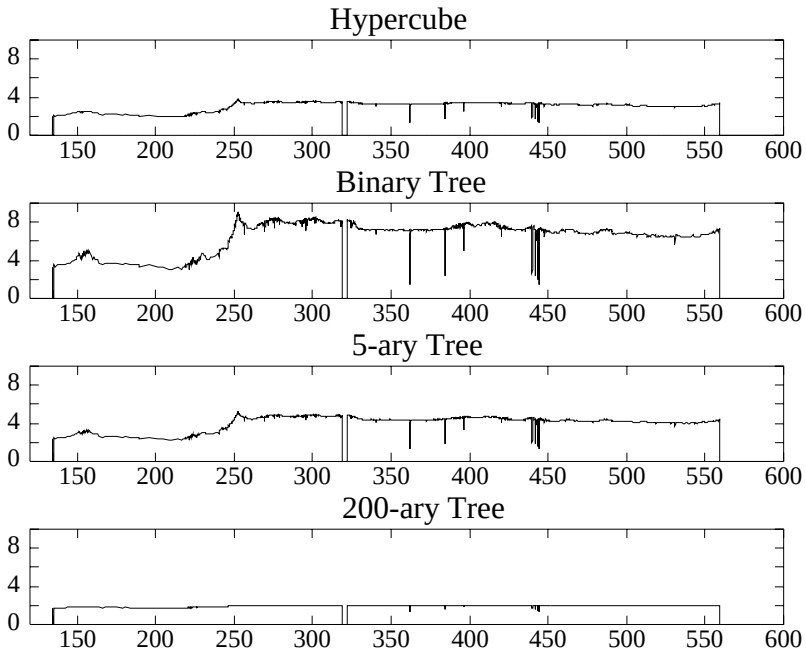
## Results for $\overline{v}$



## Results for $\overline{w}$ and $v_{max}/\overline{v}$



## Results for $\overline{p}$



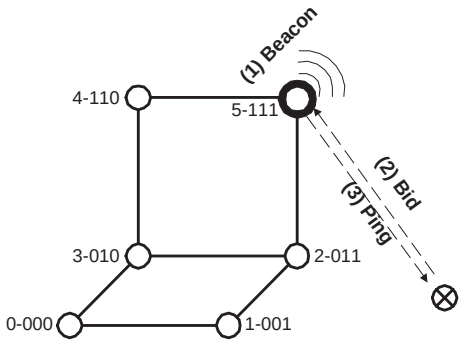
## HyperCast – The Hypercube Protocol

*HyperCast* is a protocol for maintaining the hypercube architecture.

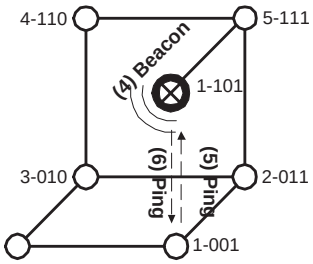
- Keeps hypercube compact
- Procedures for joining and leaving of nodes
- Recovery of hypercube from node failures
- Soft-state protocol
- Based on 5 simple message formats: `Ping`, `Beacon`, `Leave`, `Kill`, `Bid`

Java Implementation is in progress!

## HyperCast: Join Procedure




(a) Steps (1) to (3).




(b) Steps (4) to (6)..

## Conclusions

- Hypercube is new control topology for very large multicast groups
- Maintenance operations are based upon Gray ordering of nodes
- Excellent load-balancing properties
- The HyperCast protocol is a soft-state protocol that maintains the hypercube
- Implementation and scalability tests will decide if approach is viable