HyperCast -
Support of Many-To-Many Multicast Communications

Jörg Liebeherr
University of Virginia

Acknowledgements

- Collaborators:
  - Bhupinder Sethi
  - Tyler Beam
  - Burton Filstrup
  - Mike Nahas
  - Dongwen Wang
  - Konrad Lorincz
  - Neelima Putreeva

- This work is supported in part by the National Science Foundation
Many-to-Many Multicast Applications

<table>
<thead>
<tr>
<th>Group Size</th>
<th>Number of Senders</th>
<th>Collaboration Tools</th>
<th>Games</th>
<th>Distributed Systems</th>
<th>Streaming Content Distribution</th>
<th>Peer-to-Peer Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need for Multicasting?

- Maintaining unicast connections is not feasible
- Infrastructure or services needs to support a "send to group"
Problem with Multicasting

- **Feedback Implosion:** A node is overwhelmed with traffic or state
  - One-to-many multicast with feedback (e.g., reliable multicast)
  - Many-to-one multicast (Incast)

Multicast support in the network infrastructure (IP Multicast)

- **Reality Check** (after 10 years of IP Multicast):
  - Deployment has encountered severe scalability limitations in both the size and number of groups that can be supported
  - IP Multicast is still plagued with concerns pertaining to scalability, network management, deployment and support for error, flow and congestion control
Host-based Multicasting

- Logical overlay resides on top of the Layer-3 network
- Data is transmitted between neighbors in the overlay
- No IP Multicast support needed
- Overlay topology should match the Layer-3 infrastructure

Host-based multicast approaches
(all after 1998)

- Build an overlay mesh network and embed trees into the mesh:
  - Narada (CMU)
  - RMX/Gossamer (UCB)
- Build a shared tree:
  - Yalcast/Yoid (NTT)
  - Banana Tree Protocol (UMich)
  - AMRoute (Telcordia, UMD – College Park)
  - Overcast (MIT)
- Other: Gnutella
Our Approach

- **Build virtual overlay is built as a graph with known properties**
  - N-dimensional (incomplete) hypercube
  - Delaunay triangulation

- **Advantages:**
  - Routing in the overlay is implicit
  - Achieve good load-balancing
  - Exploit symmetry

- **Claim:** Can improve scalability of multicast applications by orders of magnitude over existing solutions

Multicasting with Overlay Network

**Approach:**
- Organize group members into a virtual overlay network.
- Transmit data using trees embedded in the virtual network.
Introducing the Hypercube

- An n-dimensional hypercube has $N=2^n$ nodes, where
  - Each node is labeled $k_n \ k_{n-1} \ ... \ k_1$ ($k_n = 0, 1$)
  - Two nodes are connected by an edge only if their labels differ in one position.

We Use a Gray Code for Ordering Nodes

- A Gray code, denoted by $G(.)$, 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.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>“normal”</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>Gray ordering</td>
<td>000</td>
<td>001</td>
<td>011</td>
<td>010</td>
<td>110</td>
<td>111</td>
<td>101</td>
<td>100</td>
</tr>
</tbody>
</table>
Nodes are added in a Gray order

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 \( (G^{-1}(I) < G^{-1}(R)) \)

\begin{align*}
\text{Parent} &:= \text{In} \ I_{n-1} \ldots I_{k+1} (1 - I_k) I_{k-1} \ldots I_2 I_1 \\
& \quad \text{with } k = \min_i (I_i \neq R_i).
\end{align*}

Else

\begin{align*}
\text{Parent} &:= \text{In} \ I_{n-1} \ldots I_{(k+1)} (1 - I_k) I_{k-1} \ldots I_2 I_1 \\
& \quad \text{with } k = \max_i (I_i \neq R_i).
\end{align*}

Endif
Tree Embedding

- Node 000 is root:

Another Tree Embedding

- Node 111 is root:
Performance Comparison (Part 1)

- Compare tree embeddings of:
  - Shared Tree
  - Hypercube

Comparison of Hypercubes with Shared Trees
(Infocom 98)

- $T_h$ is a control tree with root $l$
  - $w_k(T_i)$: The number of children at a node $k$ in tree $T_i$.
  - $v_k(T_i)$: The number of descendants in the sub-tree below a node $k$ in tree $T_i$.
  - $p_k(T_i)$: The path length from a node $k$ in tree $T_i$ to the root of the tree

\[
\begin{align*}
    w_k &:= \frac{1}{N} \sum_{i=1}^{N} w_k(T_i) \\
    v_k &:= \frac{1}{N} \sum_{i=1}^{N} v_k(T_i) \\
    w_{avg} &:= \frac{1}{N} \sum_{k=1}^{N} w_k \\
    v_{avg} &:= \frac{1}{N} \sum_{k=1}^{N} v_k \\
    w_{max} &:= \max_k w_k \\
    v_{max} &:= \max_k v_k
\end{align*}
\]

$P_{avg} := \frac{1}{N} \sum_{k=1}^{N} p_k$

$P_{max} := \max_k p_k$
Average number of descendants in a tree

HyperCast Protocol

- **Goal:** The goal of the protocol is to organize members of a multicast group in a logical hypercube.

- **Design criteria** for scalability:
  - Soft-State (state is not permanent)
  - Decentralized (every node is aware only of its neighbors in the cube)
  - Must handle dynamic group membership efficiently
HyperCast Protocol

- The HyperCast protocol maintains a stable hypercube:
  - **Consistent**: No two nodes have the same logical address
  - **Compact**: The dimension of the hypercube is as small as possible
  - **Connected**: Each node knows the physical address of each of its neighbors in the hypercube

Hypercast Protocol

1. A new node joins
2. A node fails
The node with the highest logical address (HRoot) sends out beacons.
The beacon message is received by all nodes.
Other nodes use the beacon to determine if they are missing neighbors.

Each node sends ping messages to its neighbors periodically.
A node that wants to join the hypercube will send a beacon message announcing its presence.

The HRoot receives the beacon and responds with a ping, containing the new logical address.
The joining node takes on the logical address given to it and adds “110” as its neighbor.

The new node responds with a ping.
The new node is now the new HRoot, and so it will beacon.

Upon receiving the beacon, some nodes realize that they have a new neighbor and ping in response.
The new node responds to a ping from each new neighbor.

The join operation is now complete, and the hypercube is once again in a stable state.
Hypercast Protocol

1. A new node joins

2. A node fails
Holes in the hypercube fabric are discovered via lack of response to pings.

The HRoot and nodes which are missing neighbors send beacons.
Nodes which are missing neighbors can move the HRoot to a new (lower) logical address with a ping message.

The HRoot sends leave messages as it leaves the 111 logical address.
The HRoot takes on the new logical address and pings back.

The "old" 111 is now node 001, and takes its place in the cube.
The new HRoot and the nodes which are missing neighbors will beacon.
Upon receiving the beacons, the neighboring nodes ping each other to finish the repair operation.

The repair operation is now complete, and the hypercube is once again stable.
State Diagram of a Node

Experiments

- **Experimental Platform:**
  Centurion cluster at UVA (cluster of Linux PCs)
  - 2 to 1024 hypercube nodes (on 32 machines)
  - current record: 10,080 nodes on 106 machines

- **Experiment:**
  - Add $N$ new nodes to a hypercube with $M$ nodes

- **Performance measures:**
  - Time until hypercube reaches a stable state
  - Rate of unicast transmissions
  - Rate of multicast transmissions
Experiment 1:

**Time for Join Operation vs. Size of Hypercube and Number of Joining Nodes**

Experiment 1:

**Unicast Traffic for Join Operation vs. Size of Hypercube and Number of Joining Nodes**
Experiment 1: Multicast Traffic for Join Operation vs. Size of Hypercube and Number of Joining Nodes

- Hypercube topology does not consider geographical proximity
- Triangulation can achieve that logical neighbors are “close by”
The Voronoi region of a node is the region of the plane that is closer to this node than to any other node.

The Delaunay triangulation has edges between nodes in neighboring Voronoi regions.
A Leader is a node with a Y-coordinate higher than any of its neighbors. The Leader periodically broadcasts a beacon.

Each node sends ping messages to its neighbors periodically.
A node that wants to join the triangulation will send a beacon message announcing its presence.

The new node is located in one node’s Voronoi region. This node (5,2) updates its Voronoi region, and the triangulation.
Jörg Liebeherr, 2001

(5,2) sends a ping which contains info for contacting its clockwise and counterclockwise neighbors.

(8,4) contacts these neighbors...

Jörg Liebeherr, 2001
... which update their respective Voronoi regions.

(4,9) and (12,0) send pings and provide info for contacting their respective clockwise and counterclockwise neighbors.
(8,4) contacts the new neighbor (10,8) ...

...which updates its Voronoi region...
...and responds with a ping

This completes the update of the Voronoi regions and the Delaunay Triangulation
Problem with Delaunay Triangulations

- Delaunay triangulation considers location of nodes, but not the network topology
- 2 heuristics to achieve a better mapping

Hierarchical Delaunay Triangulation

- 2-level hierarchy of Delaunay triangulations
- The node with the lowest x-coordinate in a domain DT is a member in 2 triangulations
Multipoint Delaunay Triangulation

- Different (“implicit”) hierarchical organization
- “Virtual nodes” are positioned to form a “bounding box” around a cluster of nodes. All traffic to nodes in a cluster goes through one of the virtual nodes

Work in Progress: Evaluation of Overlays

- **Simulation:**
  - Network with 1024 routers (“Transit-Stub” topology)
  - 2 - 512 hosts

- **Performance measures for trees embedded in an overlay network:**
  - Degree of a node in an embedded tree
  - “Relative Delay Penalty”: Ratio of delay in overlay to shortest path delay
  - “Stress”: Number of overlay links over a physical link
Transit-Stub Network

Transit-Stub
- GA Tech graph generator
- 4 transit domains
- 4×16 stub domains
- 1024 total routers
- 128 hosts on stub domain

Overlay Topologies

Delaunay Triangulation and variants
- Hierarchical DT
- Multipoint DT

Degree-6 Graph
- Similar to graphs generated in Narada

Degree-3 Tree
- Similar to graphs generated in Yoid

Logical MST
- Minimum Spanning Tree

Hypercube
**Maximum Relative Delay Penalty**

Transit–stub (hop count): Max RDP vs. Number of active nodes (Avg. over 5 graphs)

**Maximum Node Degree**

Transit–stub (hop count): Max Out–degree vs. Number of active nodes (Avg. over 5 graphs)
Maximum Average Node Degree (over all trees)

Max Stress (Single node sending)
Maximum Average Stress (all nodes sending)

Transit-stub(hopcount): Max Avg Stress (all active sending) vs. Number of active nodes (Avg. over 5 graphs)

Time To Stabilize

Transit-stub(hopcount): Time to Stabilize vs. Number of active nodes (Avg. over 5 graphs)
90th Percentile Relative Delay Penalty

Work in progress: Provide an API

- Goal: Provide a Socket like interface to applications
  - Unconfirmed Datagram Transmission
  - Confirmed Datagram Transmission
  - TCP Transmission

Overlay Protocol

OLCast

Group Manager

OLSocket Interface

OLSocket

Application Receive Buffer

Forwarding Engine

ARQ_Engine

Jörg Liebeherr, 2001
Summary

• Overlay network for many-to-many multicast applications using Hypercubes and Delaunay Triangulations

• Performance evaluation of trade-offs via analysis, simulation, and experimentation

• “Socket like” API is close to completion
• Proof-of-concept applications:
  – MPEG-1 streaming
  – file transfer
  – interactive games

• Scalability to groups with > 100,000 members appears feasible