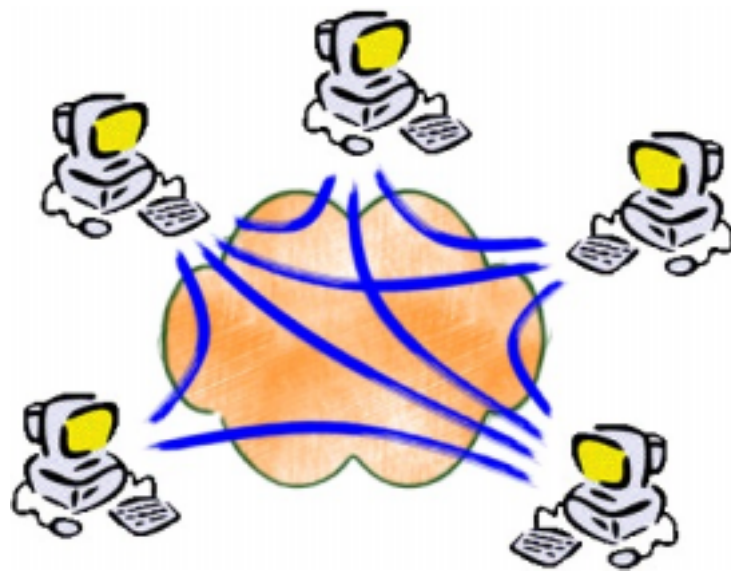


HyperCast 2.0

Design Document



HyperCast Team
Department of Computer Science
University of Virginia

Email: hypercast@cs.virginia.edu
Web: <http://www.cs.virginia.edu/~hypercast>

January 2002

Acknowledgements

Several undergraduate and graduate students, and Postdocs have contributed to the development of the HyperCast protocol. In alphabetical order, these are Jean Ablutz, Tyler Beam, Burton Filstrup, Konrad Lorincz, Huafeng Lu, Michael Nahas, Neelima Putrevu, Bhupinder Sethi, Weisheng Si, Scott Talbert, Dongwen Wang, Haiyong Wang, and Guimin Zhang.

Principal investigator of the project is Jörg Liebeherr.

The first version of HyperCast was designed by Tyler Beam, Jorg Liebeherr and Bhupinder Sethi. HyperCast 1.0 was implemented by Tyler Beam.

The design of the HyperCast 2.0 overlay socket was done by Jorg Liebeherr, Konrad Lorincz, Mike Nahas, and Dongwen Wang. Most of the implementation and testing of HyperCast 2.0 was done by Dongwen Wang, Mike Nahas, and Guimin Zhang.

The demo applications provided in the HyperCast distribution were written by Dongwen Wang (media streamer), Guimin Zhang (mftp), and Huafeng Lu (whiteboard).

The design document was written by Jorg Liebeherr, Dongwen Wang, and Guimin Zhang, with material supplied by all members.

The HyperCast project has been supported in part by the National Science Foundation under grants NCR-9624106, ANI-9870336, and ANI-0085955. The HyperCast project is part of the Denali project (<http://denali.berkeley.edu>) on scalable services for the Global Internet.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or any other sponsor.

CHAPTER 1

HyperCast 2.0 Overview

1.1 OVERLAY TOPOLOGIES IN HYPERCAST

The HyperCast software builds and maintains logical overlay networks between applications on the Internet. HyperCast provides support for data transmission between applications in the overlay. Applications self-organize into a logical overlay network, and transfer data along the edges of the overlay network using unicast transport services. Each application communicates only with its neighbors in the overlay network. Using the overlay, services for one-to-many transmissions ("multicast") and many-to-one transmissions ("incast") are relatively simple to provide.

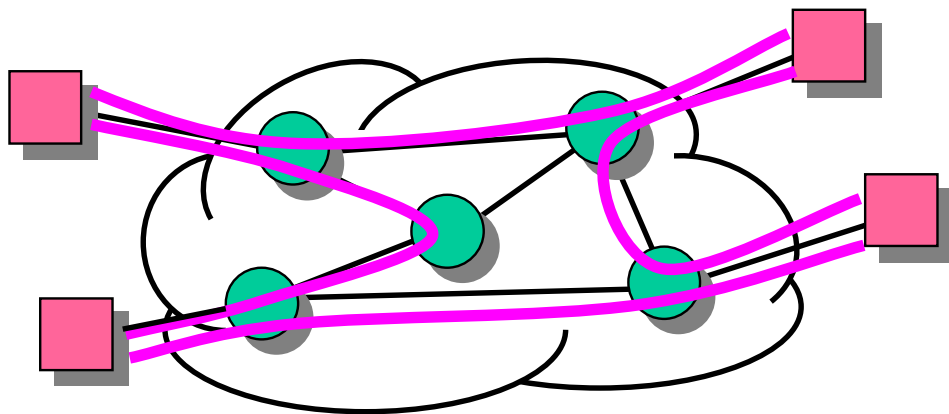


Figure 1.1. An overlay network in the Internet is a virtual network that is implemented on top of a network of routers and links. Each logical link of the virtual network consists of a complete end-to-end unicast route. Nodes in the overlay network can be hosts, routers, servers, or applications.

HyperCast builds topologies that are graphs with well-known properties. Therefore, it is feasible to make statements about worst-case properties of the overlay networks built by HyperCast. Currently, HyperCast can build two types of overlay network topologies, logical hypercubes and logical Delaunay triangulations. An important property of HyperCast overlay topologies is that, once the overlay networks are established, packet forwarding in the overlay networks can be performed without the need for a routing protocol.

1.1.1 Hypercubes

The logical hypercube overlay network topology organizes the applications into a logical n-dimensional hypercube. Each node is identified by a label (e.g., "010"), which indicates the position of the node in the logical hypercube.

One advantage of using a hypercube is that each node has at most $\log(N)$ neighbors, where N is the total number of nodes. Also, the longest route in the hypercube is $\log(N)$. One disadvantage of hypercubes is that they completely ignore the physical network infrastructure. Another disadvantage is that the hypercube construction must be done sequentially, i.e., one node at a time. Therefore, for large groups, it can take a long time before the overlay network is built. Also, the departure of a single node may require substantial changes to the overlay topology.

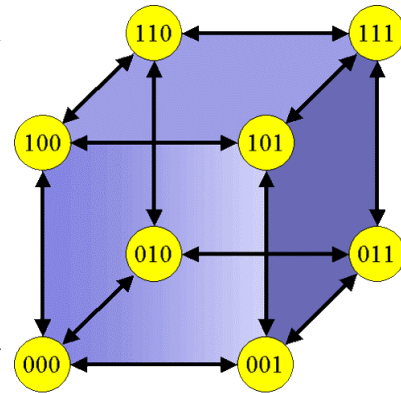


Figure 1.2. A 3-dimensional hypercube.

We refer to [HCOverlay] for a detailed description of a protocol that builds hypercube topologies. The protocol has been tested with up to 10,000 nodes running on up to 100 hosts (see [BEAM99] and [LORIN01]).

1.1.2 Delaunay Triangulation

A Delaunay triangulation is a special type of triangulation. Its main characteristic is that for each circumscribing circle of a triangle formed by three nodes, no other node of the graph is in the interior of the circle. Each node in a Delaunay triangulation has (x,y) coordinates which depict a point in the plane. In Figure 1.3, we show a Delaunay triangulation of five nodes and the circumscribing circles of some of its triangles.

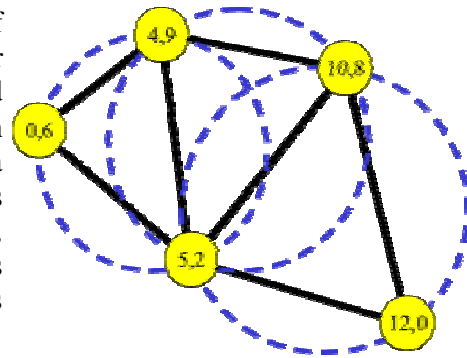


Figure 1.3. Delaunay triangulation.

An advantage of the Delaunay triangulation overlay network topology is that it can be constructed in a distributed fashion. Therefore, Delaunay triangulations can be built very quickly. In a Delaunay triangulation, each node has, on the average, six neighbors. In the worst-case, the number of neighbors of a node is $N-1$, where N is the total number of nodes.

If the coordinates of a node in the Delaunay triangulation reflect the node's geographical location, then nodes in the overlay network are likely to be neighbors if their geographical location is close. However, Delaunay triangulations are not aware of the layer-3 network infrastructure.

A protocol that builds a Delaunay triangulation overlay network is described in [HCOverlay]. The Delaunay triangulation topology has been tested with up to 10,000 running on up to 100 computers (see [LIEBE01b]).

1.2 EVALUATION CRITERIA FOR OVERLAY TOPOLOGIES

There are several criteria to evaluate an overlay network topology. We discuss how the hypercubes and logical Delaunay triangulations perform for some important criteria.

- ***Need for global information:*** The scalability of an overlay network, in terms of the achievable size of the overlay network, is limited if the state information at a node grows fast with the number of nodes in the overlay network. In both HyperCast topologies, the state information at each node is limited (with the exception of worst-case scenarios in the Delaunay triangulation).
- ***Effort to build and maintain the overlay topology:*** Ideally, the number of computations performed after a node joins or leaves the overlay network should be constant. The hypercube satisfies this for adding nodes to the overlay network, but not for removing them. The Delaunay triangulation satisfies this for both the average case of adding nodes and the average case of removing them.
- ***Need for a routing protocol in the overlay:*** For data forwarding in an overlay network, nodes generally maintain a routing table which determines the next-hop for the data towards its destination or destinations. Most existing overlay network approaches require the nodes to run a shortest path algorithm in order to establish the routing tables. In contrast, hypercubes and Delaunay triangulations can use the logical addresses of nodes (binary strings or coordinates) to determine next hop routing information. This eliminates the need for routing protocols.
- ***Match of the overlay network to the Internet topology:*** Data forwarding in overlay networks is done at the application level. Therefore, data may traverse the IP network several times before it reaches its destination or destinations. This may result in inefficient use of network capacity and increased delays compared to transmission at the IP layer. These disadvantages, which are shared to some degree by all overlay network topologies, are least pronounced if the overlay network is constructed with respect to the of the underlying Internet topology. Since the overlay topologies of HyperCast are constructed without awareness of the Internet topology, the overlay networks may not be good matches for the underlying Internet topology.

1.3 DATA TRANSPORT

Data is exchanged in the overlay network as formatted messages. Data transmission between two members of an overlay network can be done with either UDP or with TCP. An application selects a specific overlay topology and a specific protocol for data transport when it instantiates an overlay socket.

In HyperCast, all data is transmitted along trees that are embedded in the overlay network topology. For each member of an overlay network, there is an embedded spanning tree in the overlay network with that member as the root of the tree. Given the root of an embedded tree, each member of an overlay network can locally determine

its children and parent with respect to that tree. Each member forwards data to its children or parent in an embedded tree with respect to a specific node.

Hence, an important requirement for HyperCast overlay topologies is that, given the root of an embedded tree, each member of an overlay network can locally determine its children and parent with respect to that tree. The node forwards data either to its children or to its parent.

Multicast transmission in an overlay network is transmitted in the embedded tree that has the multicast sender as root (see Figure 1.4(a)). If an overlay member receives a multicast message, it passes the message to the application and forwards the message to all of its child nodes in the embedded tree.

Unicast transmission is performed by sending data upstream in an embedded tree. A unicast message is forwarded to the parent in the embedded tree that has the destination as the root (see Figure 1.4(b)).

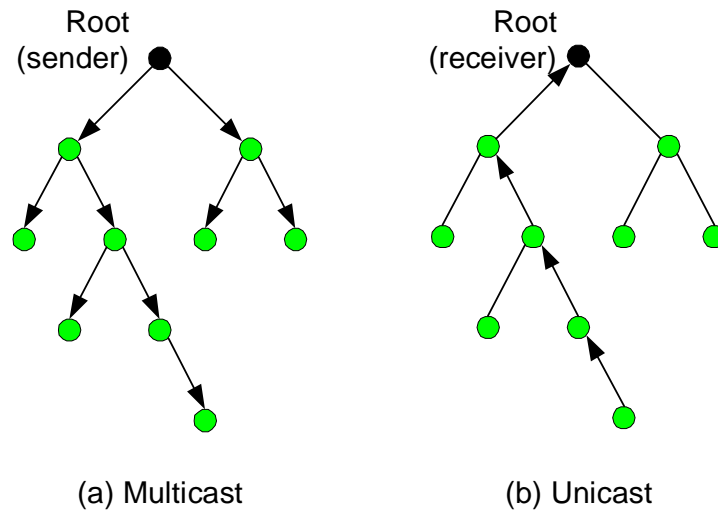


Figure 1.4. Data forwarding in overlay networks. Data is forwarded along trees that are embedded in overlay networks. For each member of the overlay, there is an embedded tree with this member at the root.

1.4 OVERLAY SOCKETS

The HyperCast software is built around the notion of an overlay socket (OL Socket). An overlay socket is an endpoint for communication in an overlay network. An overlay socket provides application programs an interface for communications over a logical overlay topology. The application programming interface (API) of an OL Socket offers applications the ability to

- join and leave existing overlays;
- send data to all or a subset of the members of the overlay network; and
- receive data from the overlay.

The HyperCast software is structured around components which are either part of an overlay socket or that interact with an overlay socket. The main components of an OL_Socket, as illustrated in Figure 1.5, are:

- **Overlay Node (OL Node):** The overlay node implements the protocol that establishes the overlay network topology. Currently, two overlay network topologies are supported: a logical hypercube, and a logical Delaunay triangulation. The overlay node is the only component of an overlay socket that is aware of the overlay topology.

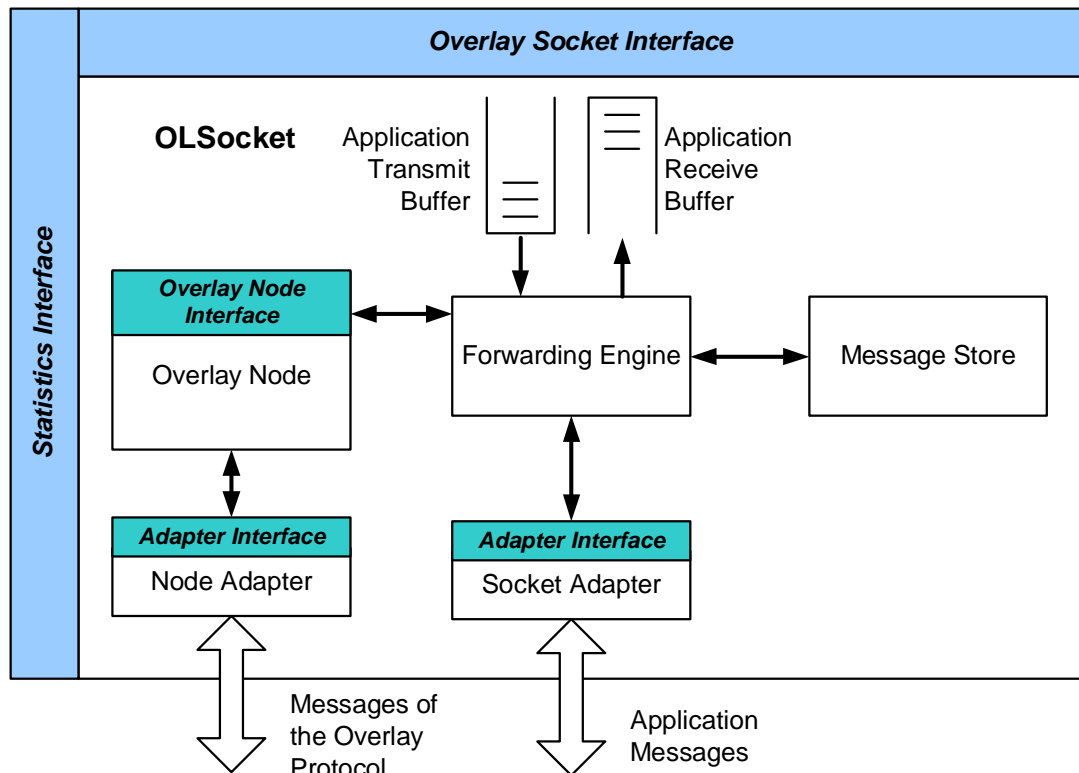


Figure 1.5. Components and interfaces of an overlay socket.

- **Forwarding Engine:** The forwarding engine is responsible for sending, receiving, and forwarding messages in the overlay network. The forwarding engine communicates with the overlay node to obtain next hop routing information for messages.
- **Application Transmit Buffer and Application Receive Buffer:** The application transmit buffer contains untransmitted messages which are buffered due to rate control or congestion control constraints. The application receive buffer contains messages which are received but have not yet been read by the application. If the receive operation uses upcalls, there is no need for application receive buffers.
- **Message Store:** The message store is a repository of transmitted messages. It buffers messages and maintains state information about them. The message store aids in providing services with various reliability semantics, synchronization, and other features.

- **Statistics:** Each component of an overlay socket supports an interface for monitoring and managing its properties. The interface for accessing statistics is similar for all components.
- **Adapters:** Messages in the overlay can be transmitted over different transport protocols, UDP or TCP. The adapters provide the interfaces for a specific transport protocol. Each overlay socket has two adapters. The node adapter is the network interface for the overlay node; the socket adapter is the network interface for application data.

The following components are external to an overlay socket, but interact closely with it.

- **Overlay Manager:** The overlay manager is responsible for creating and managing identifiers and properties for an overlay network. Each overlay network has a unique identifier, the overlay identifier (overlay ID).
- **Remote Control and Monitoring:** The statistics interfaces of the overlay socket provide access to the value and status of the components in the sockets. The remote control and monitoring components provide utilities for accessing this data through the statistics interfaces.

1.5 WRITING PROGRAMS WITH HYPERCAST

Programming with overlay socket is similar to network programming with a socket-style API. Below is a simple example which uses some of the HyperCast overlay socket application programming interface. Details of the application programming interface can be found [HCProgram].

```
//Generate the configuration object
OverlayManager om = new OverlayManager(propertyfilename);
OverlaySocketConfig config = null;
String overlayID = om.getDefaultProperty("OverlayID");
if (overlayID == null || overlayID.equals("") ||
    !gm.doesOverlayExist(overlayID))
    config = om.createOverlay(overlayID);
if (config== null)
    config = om.getOverlaySocketConfig(overlayID);

//create an overlay socket
OL_Socket socket = config.createOverlaySocket(callback);

//Join an overlay
socket.joinGroup();

//Create a message
OL_Message msg = socket.createMessage(byte[] data, int length);

//Send the message to all members in overlay network
socket.sendToAll(msg);

//Receive a message from the socket
OL_Message msg = socket.receive();

//Extract the payload
byte[] data = msg.getPayload();
```

The important calls are highlighted in bold face. Before creating an overlay socket, the application program creates an object for the management and configuration of the overlay network. The configuration object reads configuration parameters from a file.

Then the overlay socket is created and initialized with the configuration object. Once the overlay socket is created it can join the overlay network. A multicast message to all members of the overlay network is accomplished with the "sendToAll" method. The "receive" method is used for receiving data.

1.6 REFERENCES

[BEAM99] T. K. Beam. HyperCast: A Protocol for Maintaining a Logical Hypercube-Based Network Topolog. M.S. Thesis, University of Virginia, May 1999.

[HCOverlay] "Overlay Protocols", HyperCast 2.0, Design Documents, <http://www.cs.virginia.edu/hypercast/documentation.html#ProtocolDesign>, January 2002.

[HCProgram] "Programming", HyperCast 2.0, Design Documents, <http://www.cs.virginia.edu/hypercast/documentation.html#Programming>, January 2002.

[LIEBE99] J. Liebeherr and T. K. Beam. HyperCast: A protocol for maintaining multicast group members in a logical hypercube topology. In *Proceedings First International Workshop on Networked Group Communication (NGC '99)*, Lecture Notes in Computer Science, Volume 1736, pages 72-89, 1999.

[LIEBE01a] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. *Proceedings of IEEE Globecom 2001*, Global Internet Symposium.

[LIEBE01b] J. Liebeherr, M. Nahas, and W. Si. Large-scale Application-Layer Multicast with Delaunay Triangulations. Manuscript, September 2001.

[LORIN01] K. Lorintz, HyperCast: A Super-Scalable Many-to-Many Multicast Protocol for Distributed Internet Applications, Undergraduate Thesis, School of Engineering and Applied Science, University of Virginia. May 2001.