

CHAPTER 2

Overlay Socket

2.1 OVERVIEW

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 overlay 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.

When an application creates an overlay socket, the socket attempts to join the overlay network. The type and attributes of the overlay network are configuration parameters of an overlay socket.

Overlay sockets can exchange data other overlay sockets that are neighbors in the overlay network. Data is sent as formatted messages that are called application messages. If an overlay socket receives an application message from one of its neighbors in the overlay network, it determines if the message must be forwarded to other overlay sockets. In addition, the forwarding engine passes the message to the application.

Application data is sent either over UDP or over TCP connections. Messages of the protocol that build the overlay network are generally sent as UDP datagrams. The transmission of application data and overlay protocol messages is handled by a different transport layer socket.

The main components of an overlay socket are illustrated in

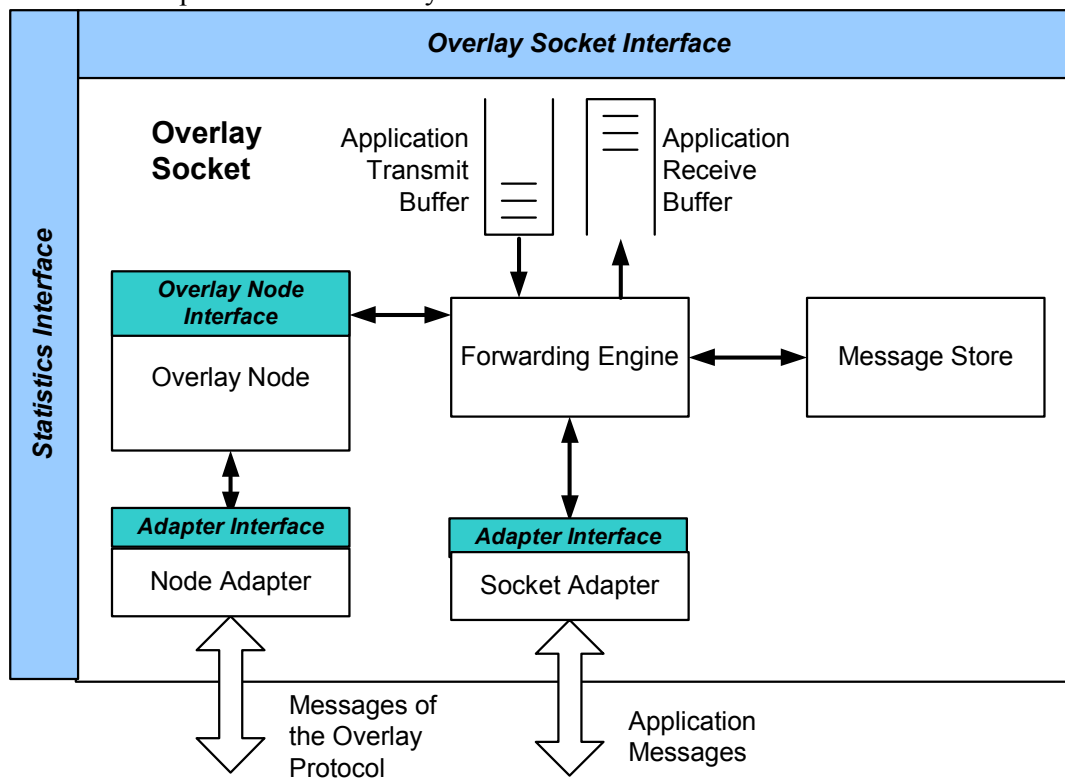


Figure 2.1:

- **Overlay Node:** The overlay node implements the protocol that establishes the overlay network topology. The overlay node is the only component, which is specific to an overlay topology. Currently, two overlay network topologies are supported: logical hypercube and logical Delaunay triangulations.
- **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 information for a message.

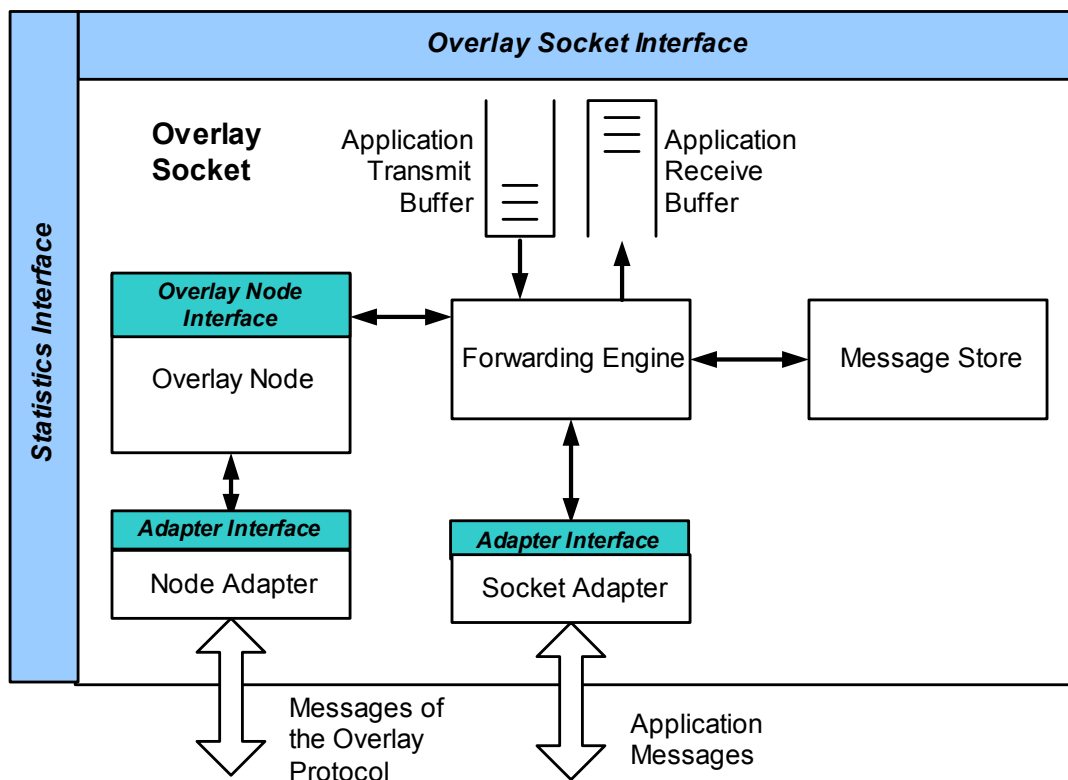


Figure 2.1. Components and interfaces of an Overlay Socket.

- **Application Transmit Buffer and Application Receive Buffer:** The application transmit buffer contains transmitted messages which are buffered due to rate control or congestion control constraints. The application receive buffer contains messages which have been received, but have not yet been read by the application.
- **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 others features.
- **Network Adapter:** Messages in the overlay can be transmitted over different transport protocols (i.e. either UDP or TCP). The transmission of data is handled by the network adapter. Each overlay socket has two network adapters. One adapter handles messages for the protocol that maintains the overlay network. The second adapter handles the sending and receiving of application data.

Management of overlay membership information is done outside of the overlay socket. The overlay socket has a set of internal and external interfaces.

- **Overlay Socket Interface:** The overlay socket interface is the application programming interface (API) of the overlay socket. The interface has some similarity to the API of BSD sockets.

- **Statistics Interface:** Each component of an overlay socket supports an interface for monitoring and managing its properties. The interface for accessing the statistics is similar for all components of the overlay socket.
- **Overlay Node Interface:** The overlay node interface provides access to the Overlay Node. Since all overlay nodes have the same interface, the overlay network topology can be changed without changing other components of the overlay socket.

The following components are external to an Overlay Socket, but interact closely with it.

- **Overlay Manager:** Overlay Manager is responsible for creating and managing identifiers and properties for an overlay network. Each overlay network has a unique identifier called Overlay ID, and a set of properties. The overlay manager may communicate with a remote server. When an Overlay Socket is created, the Overlay ID is provided as a parameter.
- **Remote Control and Monitor:** The statistics interfaces of the Overlay Sockets provide access to the value and status of the components in the sockets. The remote control and monitor components provide utilities for accessing this data through the statistics interfaces.

2.2 SOCKETS SERVICES AND NETWORK ADAPTORS

Overlay Sockets can use either TCP or UDP to transfer application data to other Overlay Sockets. Therefore, there are two different types of Overlay Sockets:

- **Connectionless (CL) Overlay Sockets:** Application messages are exchanged as UDP unicast datagrams between neighbors in the overlay network. Since UDP provides only a best effort service, a CL overlay socket provides an unassured message delivery service without guarantees of reliable or in-sequence delivery. Note that the delivery service of CL overlay sockets can be made reliable by using the services of the message store.
- **Connection-Oriented (CO) Overlay Sockets:** TCP connections are used to exchange application messages between neighbors in the overlay network. CO overlay sockets guarantee sequential delivery (with respect to the same sender) and reliable transfer of messages to neighbors in the overlay.

The type of socket is specified when the overlay socket is created. The only component of the overlay socket that is impacted by the choice of either a CL or a CO overlay socket is the node adapter shown in

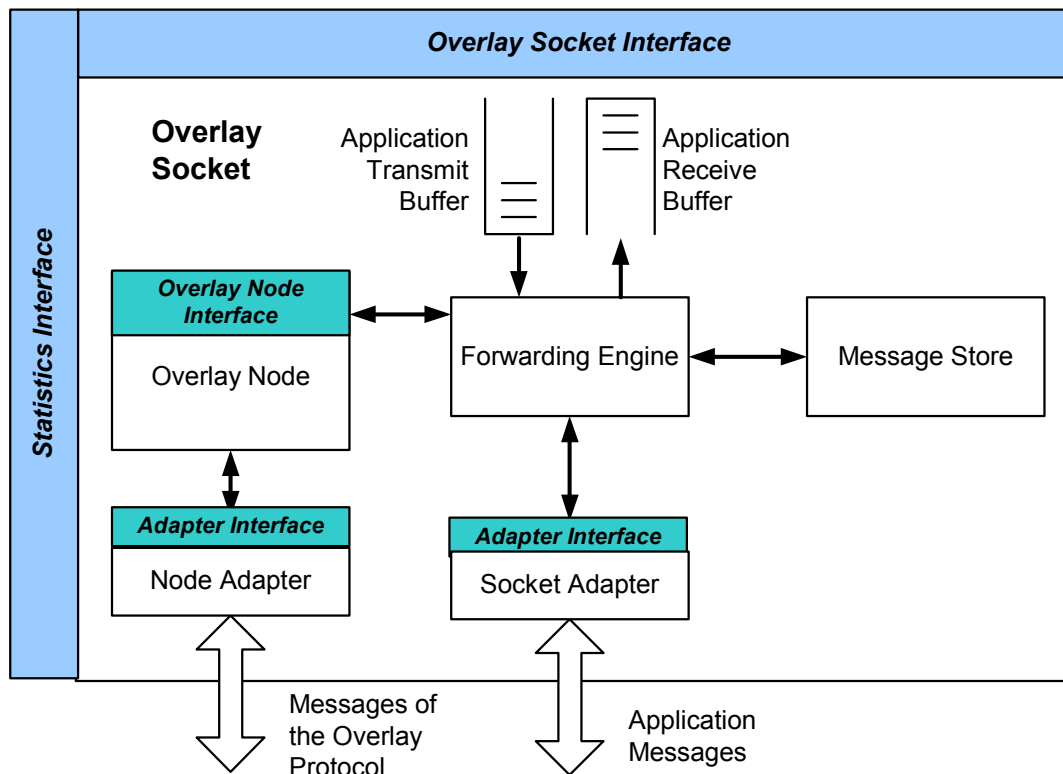


Figure 2.1.

The CO overlay socket is more complex than the CL overlay socket, because of the need to maintain TCP connections. Each overlay socket must run a TCP server (Stream socket) which accepts connection requests from neighbors in the overlay network. There are three basic ways to establish TCP connections between neighbors in the overlay network.

1. **Per-message connections:** A TCP connection is established for each message that is transmitted. Once the message is transmitted, the TCP connection is terminated.
2. **Standing connections:** A TCP connection is established between every pair of neighbors in the overlay network. The connection is established and terminated based on the content of the overlay nodes' neighborhood tables.
3. **On-demand standing connections:** Same as standing connections, but connections are only established when there is a need to transfer data between the nodes. Once a connection is established it will stay in place.

With standing TCP connections, only one TCP connection is required between two nodes that exchange data. However, since maintaining a single TCP connection involves more bookkeeping overhead than maintaining two TCP connections, one for each unidirectional data flow, HyperCast 2.0 uses two on-demand TCP connections for data transfer.

The forwarding engine of an overlay socket forwards application data. This is similar to the IP forwarding functions of a IP router. The routing information is provided by the overlay node. In HyperCast, data is transmitted along trees that are embedded in the overlay network topology. Therefore the information that is needed to forward data is the “parent node” or “child nodes” with respect to a given embedded tree. The forwarding engine supports three delivery modes: multicast, unicast, and flooding.

- **Multicast** transmissions in an overlay network are sent in a spanning tree that is embedded in the overlay network, which has the multicast sender as its root. If an overlay socket receives a multicast message, the forwarding engine passes the message to the application and forwards the message to all of its child nodes in the embedded tree. The forwarding engine queries the overlay node to obtain the set of the child nodes.
- **Unicast** transmissions are performed “upstream” in an embedded tree. The forwarding engine passes a unicast message to the overlay node’s parent in the embedded tree that has the destination as its root. The forwarding engine queries the overlay node to obtain the overlay node’s parent.
- **Flooding** messages are transmitted by an overlay node to all of its neighbors in the overlay topology, except for the neighbor from which it received the message. The forwarding engine obtains the list of the overlay node’s neighbors from the overlay node.

The forwarding process of the forwarding engine are shown in Figure 2.2. The forwarding engine decrements a Hopcount field in the message header and drops the message if the counter is zero. Then, the forwarding engine determines the delivery mode and forwards the message to the node’s neighbors in the overlay network. If the message cannot be transmitted due to flow control or congestion control constraints in the overlay socket, then it is written to the application sending buffer. Next, the forwarding engine passes the message to the application. This is done by writing the message in the application receive buffer. If the application has provided a callback function, which is executed when a new message arrives, then the message is passed to this callback function.

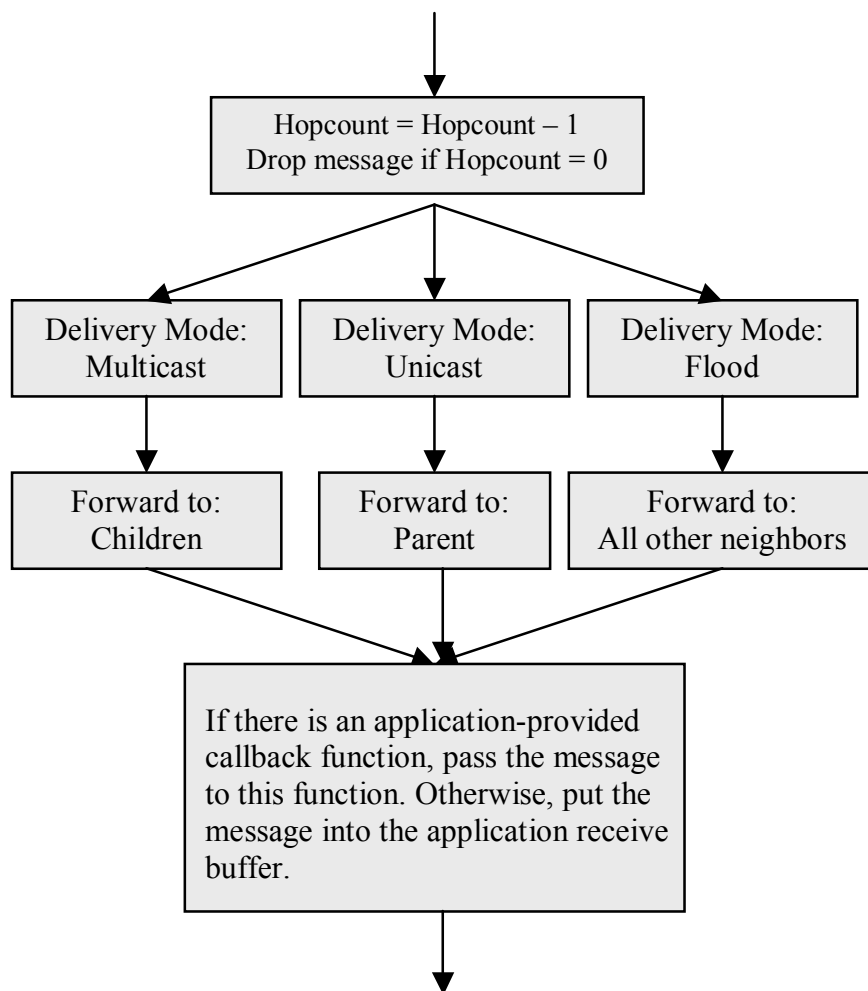


Figure 2.2. Operation of the Forwarding Engine when a message is received.

The decision about whether or not to forward a message is made based on the logical address of an overlay node. Thus, routing in overlay networks is based exclusively on logical addresses. The mapping of the logical address of the overlay network to the physical address is done with the help of the overlay node in the neighborhood table. In HyperCast 2.0, physical addresses are always IP addresses combined a port number. The overlay node's neighborhood table is updated whenever the overlay topology changes.

Each application message has a header which contains the logical address of the sender (source logical address or Src LA). For unicast messages, it also contains the logical address of the receiver (destination logical address or Dest LA). For multicast messages, all nodes that receive a message need to forward it to their children in a tree that has the node with the logical address Src LA as the root. For unicast transmission, each node needs to forward the message to its parent node in the tree that has as the root node with the logical address Dest LA. For flooding, all nodes forward the message to all of their neighbors with the exception of the neighbor from which the message was received.

The forwarding engine queries the overlay node about its "children," its "parent," and its "neighbors" for every message. The logical addresses of the "children," "parent,"

and “neighbors” may change if nodes join or leave the overlay network. Hence, two consecutive queries to obtain their logical addresses may yield different logical addresses.

2.3 PROGRAMMING WITH OVERLAY SOCKETS

Programming with overlay sockets is similar to network programming with BSD sockets. In Figure 2.3, we show a simple example which uses the HyperCast overlay socket API. Details about the API can be found in a separate chapter.

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

Figure 2.3. Example program for HyperCast.

Before creating an overlay socket, the application program creates an object to manage and configure the overlay network. The configuration object reads the configuration 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.