Overlay Socket Tutorial
J. Liebeherr

HyperCast Overlay Topologies

- Applications organize themselves to form a logical overlay network with a given topology
- Data is forwarded along the edges of the overlay network

Hypercube
Delaunay triangulation
Spanning tree
(for mobile ad hoc)
Programming in HyperCast: Overlay Socket

- Socket-based API
- UDP (unicast or multicast) or TCP
- Supports different semantics for transport of data
- Supports different overlay protocols
- Implementation in Java

Network of overlay sockets

- An overlay network is a collection of overlay sockets
- Overlay sockets in the same overlay network have a common set of attributes
- Each overlay network has an overlay ID. Overlay ID is a key to access attributes of overlay
- Overlay socket is an endpoint of communication in an overlay network
- Nodes exchange data with neighbors in the overlay topology
Overlay Socket

- An overlay socket (OL Socket) is an endpoint for communication in an overlay network.
- An overlay socket provides application programs an interface for communications over an overlay network.
- The application programming interface (API) of an OL Socket offers applications the ability to:
  - create overlay;
  - 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.

Network of overlay sockets

- An overlay network is a collection of overlay sockets.
- Overlay sockets in the same overlay network have a common set of attributes.
- Each overlay network has an overlay ID. Overlay ID is a key to access attributes of overlay.
- Overlay socket is an endpoint of communication in an overlay network.
- Nodes exchange data with neighbors in the overlay topology.
Unicast and Multicast in overlays

- Unicast and multicast is done using trees that are embedded in the overlay network.

- **Requirement:**
  Must be able to compute the child nodes and parent node with respect to a given root.

---

Socket Based API

- Tries to stay close to Socket API for UDP Multicast
- Note: This program does not depend on overlay topology

```java
String MyString = new String("Hello World");

//Create an object that contains configuration parameters
OverlaySocketConfig ConfObj = OverlaySocketConfig.createOLConfig("hypercast.xml");

//Create the overlay socket
I_OverlaySocket MySocket = ConfObj.createOverlaySocket(null);

//Overlay socket joins the overlay
MySocket.joinOverlay();

//Create an application message with "Hello World" payload
I_OverlayMessage msg = socket.createMessage(MyString.getBytes(), MyString.getBytes().length);

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

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

//Extract the payload
byte[] data = msg.getMessageData();
```
Some methods of the API

**Overlay Operations**
- void joinOverlay() Starts an attempt to join an overlay network
- leaveOverlay() Leaves an overlay

**Send an overlay message from this socket:**
- void sendToAll(m) Sends (multicasts) an application message to all overlay sockets in the overlay network
- void sendToChildren(m, root) Sends an application message to children with respect to an embedded tree with given root
- void sendToAll(m) Sends an application message to all neighbors
- void sendToParent(m, root) Sends an application message to parent node with respect to an embedded tree with given root
- void sendToNode(m, destination) Sends an application message to a specified node with a given logical address
- void sendFlood(m) Sends an application message using “flooding”, i.e., the message is forwarded to all neighbors with exception of the node from which the message was received

Reading with/without callbacks

- **Synchronous receive**
  - Receive operation blocks if there is no data waiting

- **Asynchronous receive:**
  - Application supplies callback function
Available extensions

- **Message-based API:** See previous example
- **Notifications:** Overlay socket can report events to an application program.
- **HCAST Application:** This is the most simple API. Use this if your program only uses a single overlay socket.
- **Message-based APIs with Enhanced Services:** The basic-message can be enhanced by using one of the many services provided by the message store, such as in-sequence delivery, synchronization, reliability.
- **Stream API:** The Stream API supports a byte-oriented byte stream with in-sequence delivery.
- **Secure socket API:** This message-based API provides integrity and/or privacy of data transmitted between peers in the overlay network, and provides integrity for the transmission of overlay protocols.

Summary: API

- API is based on Berkeley Sockets
- Application program can be left unaware of overlay network topology
- Application only works with the addresses used by the overlay (logical addresses). Application does not know transport layer addresses (physical addresses)
- How does the program know what type of overlay to start or to join?
  → **Overlay ID and Attributes**
Overlay ID

- An overlay network is uniquely identified by an overlay identifier (Overlay ID)
  - The overlay ID should be a globally unique identifier, e.g., IP address + timestamp: “128.143.71.29:997831668759”
  - No assumption on specific format of overlay ID
  - Uniqueness is not enforced

- Overlay ID is used as a key to access the properties ("attributes") of an overlay network

- Overlay ID can be created by application or by a server

Attributes

- An overlay socket is characterized by a set of attributes that specify the components of an overlay sockets

- Creation of an overlay network ties an overlay ID to a set of attributes

- Attributes are defined in an XML file

- Use the XML Configurator to write configuration files (Any file created by the XML Configurator should result in a valid set of attributes).
Starting an Overlay

- Configuration file:
  - Overlay ID
  - Overlay protocol
  - TCP or UDP
  - etc.

- Load attributes

- Data transfer

Overlay Server

- Overlay server can help with the management of overlay attributes
  - Can generate Overlay IDs
  - Can store attributes
  - Can respond to queries for attributes
  - Can provide access control to attributes

- Overlay server is implemented as a minimal http server

- Attribute in the configuration file tells if an overlay server is used or not
Starting an Overlay with Overlay server

Interactions between overlay server and overlay manager
Summary: Managing properties of overlay

- Overlay ID is an (unique) identifiers for an overlay network
- Attributes specify configuration of an overlay socket
- Configuration file stores attributes

- Overlay is started from configuration file
- Attributes of an overlay can be stored at overlay server
  - Interface to overlay server uses HTTP

Some Features

- Design separates overlay maintenance (overlay protocol) from data transport
Overlay Socket: Components

- Components are configured when the overlay socket is created.
- Two transport level ports are used:
  - Data transfer
  - Overlay protocol
- Adapters are available for IP networks, and can be built for any type of underlay network (IPv6, no IP).

Separation of overlay protocol from data transfer

- Each overlay socket has two communication ports:
  1. Protocol to manage the overlay (overlay protocol)
  2. Data transfer
Overlay Node

- The overlay node adds and maintains the membership of an overlay socket in an overlay network

- Overlay nodes runs an overlay protocol (e.g., Delaunay triangulation)
  - Rendezvous with other overlay nodes using servers, broadcast or buddy lists
  - Overlay exchanges messages with neighbors in the overlay network

<table>
<thead>
<tr>
<th>Logical address</th>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x, y)</td>
<td>128.143.137.21 / 2233</td>
</tr>
<tr>
<td>(a, d)</td>
<td>128.143.71.144 / 2567</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Overlay Protocol

- Overlay node is the only component that knows the overlay protocol, and the overlay protocol message format
- Current overlay protocols have small finite state machine
- Message format of DT protocol

<table>
<thead>
<tr>
<th>Type</th>
<th>OverlayID</th>
<th>SRC</th>
<th>DST</th>
<th>ADDR1</th>
<th>ADDR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>4 bytes</td>
<td>14 bytes</td>
<td>14 bytes</td>
<td>14 bytes</td>
<td>14 bytes</td>
</tr>
</tbody>
</table>

Logical addresses vs. physical addresses

- **Logical address (LA):** overlay specific addresses, e.g., coordinates in DT protocol
- **Physical address (PA):** address of the overlay socket in the underlay network, e.g., IP address + port number
Forwarding Engine

- Forwarding Engine performs functions of an application layer "router".

- Forwarding Engine makes forwarding decisions with logical addresses.

- Forwarding engine forwards data by requesting "children" and "parent" in a tree with respect to a "root".

Overlay Socket Interface

Send

ForgotToAll(Data) {

// Build the message

// Get the list of children from overlay node

// Get physical address of children

// Send message to children nodes

}
Receive and Forward

OL_Forward() {
// 1. Forward packet
// Determine the children in the tree
// Send datagram to children nodes

// 2. Pass packet on to application
if (UpCallFunction available)
    CallBackforReceive.receiveMessage (RecvdDatagram.Data);
else
    ApplRecvBuffer.Write(RecvdDatagram.Data);
}

Message Formats

<table>
<thead>
<tr>
<th>Overlay message header</th>
<th>Extension</th>
<th>...</th>
<th>Extension</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4 bits)</td>
<td>DMode (4 bits)</td>
<td></td>
<td>Traffic Class (8 bits)</td>
<td>Flow Label (16 bits)</td>
</tr>
<tr>
<td>Overlay Message Length (16 bits)</td>
<td>Hop Limit (16 bits)</td>
<td></td>
<td>Extension Type (8 bits)</td>
<td>LA Size (8 bits)</td>
</tr>
<tr>
<td>Previous Hop Logical Address (variable)</td>
<td>Destination Logical Address (only if DMode=0x3, variable)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Message Formats

Monitor and Control System

- Loosely modeled after SNMP (but more powerful):
  - Each socket component collects statistics
  - Statistics are accessible via a statistics interface
  - Statistics are accessed at a portal by a monitor
  - Statistics are transmitted as XML data
  - Monitor and portal send queries and responses in XML messages
Comparison with SNMP

<table>
<thead>
<tr>
<th>Portal</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>Monitor</td>
</tr>
<tr>
<td>Schema</td>
<td>MIB</td>
</tr>
<tr>
<td>Hierarchical names</td>
<td>Object identifier</td>
</tr>
<tr>
<td>XML messages</td>
<td>SNMP protocol</td>
</tr>
</tbody>
</table>

Monitors and Portals

Application

Monitor Application

Monitor

Portal

Monitor protocol messages

Retrieve

Modify

Retrieve

Modify

Statistics

Optional in application

Overlay Socket

Protocol and Overlay Messages
Monitors and Portals

Monitor Application

Monitor protocol messages

Portal

Application

Statistics (optional in application)

Retrieve

Modify

Overlay Socket

Protocol and Overlay Messages

Monitor Overlay Network

Monitor Application

Monitor Overlay Network

Overlay network (being monitored)

HyperCast-Army meeting

DENALI
Statistics Interface

- Each component of socket provides statistics
- Statistics are accessed and changes with 3 calls:
  - getStat(), setStat(), getSchema()
- All parameters are XML DOM data structures

Hierarchy of statistics
Accessing Statistics

- Statistics are accessed using XPath expressions
- Addressing the number of bytes sent: /Socket/Node/NodeAdapter/UBytesSent
- Addressing all statistics of the overlay node: /Socket/Node

Query for statistics

```xml
<GetQuery Src="100011" Dest="101010" MsgID="13" TimeStamp="100516">
  <Stats index="0" xpath="/Socket/Node/NodeAdapter/UPacketsSent" />
  <Stats index="1" xpath="/Socket/Node/NodeAdapter/UBytesSent" />
</GetQuery>
```
Processing a query at a portal

- Query is processed and translated in an access to an object with the statistics API

Response to query

```xml
<GetReply Src="101010" Dest="100011" MsgID="13"
TimeStamp="106340">
  <Stats index="0" xpath="/Socket/Node/NodeAdapter/UPacketsSent">
    <UPacketsSent>120</UPacketsSent>
  </Stats>
  <Stats index="1" xpath="/Socket/Node/NodeAdapter/UBytesSent">
    <UBytesSent>120</UBytesSent>
  </Stats>
</GetReply>
```
RunController and RunServer

• Two applications that use the monitor and control system
• Applications are used for measurement experiments and for visualization

Commands of the RunController

• Command line interface at RunController
• Commands get translated into monitor protocol messages to portals

> available
> create_experiment 10
> slow_start_experiment 5
> wait_until_stable
> get_bw_results
> stop_experiment
> kill remote servers
RunController GUI

- Graphical user interface for the RunController application