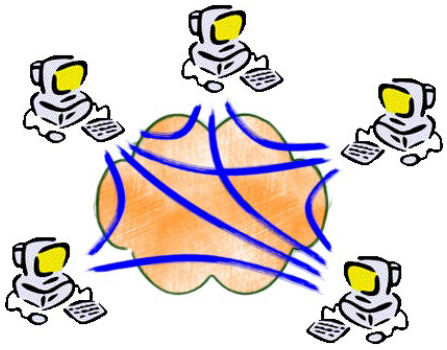
Overlay Socket Tutorial



Overview

1. Terminology:

- Overlay socket, overlay network

2. User perspective:

- The API: Writing programs with an overlay socket
- Managing Properties of an overlay network
  a) Attributes
  b) Property file
  c) Starting/Joining an overlay network
  d) Overlay manager

3. Design

- Components of an overlay socket
- Data Forwarding
- Overlay node
- CO, CL sockets

4. Monitor and control infrastructure

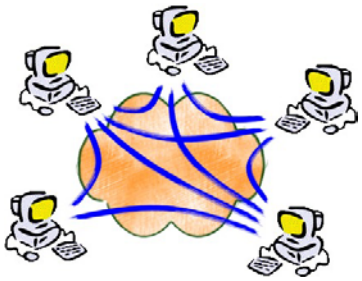
- Statistics interfaces
- Portal, Manager

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.

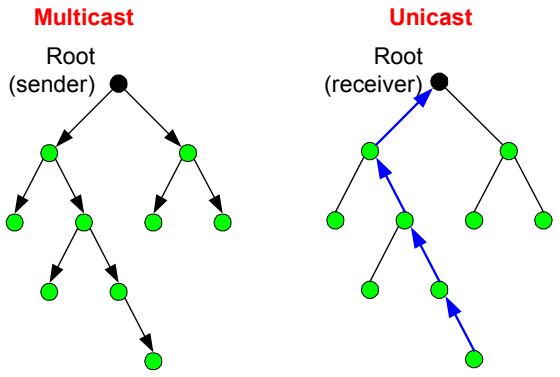
Overlay Network

- An **overlay network** is a collection of overlay sockets (OL Sockets), where the overlay sockets are connected with an overlay protocol
- In an overlay network, nodes exchange data with neighbors in the overlay network
- Data is exchanged using trees that are embedded in the overlay network
- Each Overlay network has a “unique ID”
- Overlay sockets in the same overlay network have a common set of “attributes”



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



DENALI

Socket Based API

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

```
//Generate the configuration object
OverlayManager om = new OverlayManager("hypercast.prop");
String overlayID = om.getDefaultProperty("MyOverlayID")
OverlaySocketConfig config = new
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();
```

DENALI

Some methods of the API

Overlay Operations

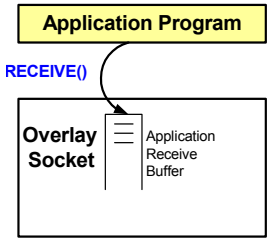
- void joinGroup() — Starts an attempt to join an overlay network
- leaveGroup() — Leaves an overlay

Send an overlay message from this socket:

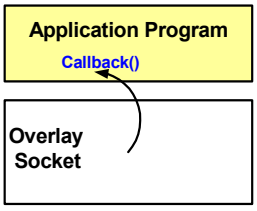
void sendToAll(m)	Sends (multicasts) an application message to all overlay sockets in the overlay network
void sendToChildren(m, root)	Sends an application message to children with respect to an embedded tree with given root
void sendToAll(m)	Sends an application message to all neighbors
void sendToParent(m, 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



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
  - Example:

```
OverlayID = 224.228.19.78/9472
KeyAttributes = Socket,Node,SocketAdapter
SocketAdapter = TCP
Node = HC2-0
```

- Attributes are key attributes or configurable attributes
  - Key attributes cannot be modified
    - The following are always key attributes: `OverlayID, KeyAttributes`
    - Other key attributes are specified as a list in `KeyAttributes`
  - Configurable attributes are “not essential” and can be changed
- Attributes can have subattributes
- **Creation of an overlay network ties an overlay ID to a set of key attributes**

Property File

- Attributes and their values are stored in a **property file** (default: “hypercast.prop”)

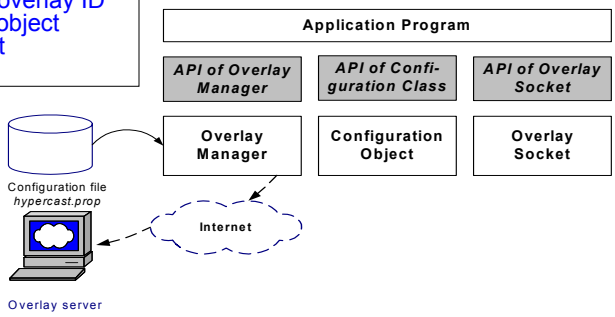
```
# This is the Hypercast Configuration File
#
# (c) University of Virginia 2001

# LOG FILE:
LogFileName = stderr

# ERROR FILE:
ErrorFileName = stderr

# OVERLAY Server:
OverlayServer =

# OVERLAY ID:
OverlayID = 224.228.19.78/9472
KeyAttributes = Socket,Node,SocketAdapter

# SOCKET:
Socket = HCast2-0
HCAST2-0.TTL = 255
HCAST2-0.ReceiveBufferSize = 200
HCAST2-0.ReadTimeout = 0

...
```

```
# SOCKET ADAPTER:
SocketAdapter = TCP
SocketAdapter.TCP.MaximumPacketLength = 16384
SocketAdapter.UDP.MessageBufferSize = 100

# NODE:
Node = HC2-0
HC2-0.SleepTime = 400
HC2-0.MaxAge = 5
HC2-0.MaxMissingNeighbor = 10
HC2-0.MaxSuppressJoinBeacon = 3


# NODE ADAPTER:
#
NodeAdapter = UDPMulticast

NodeAdapter.UDP.MaximumPacketLength = 8192
NodeAdapter.UDP.MessageBufferSize = 18
NodeAdapter.UDPServer.UdpServer0 = 128.143.71.50:8081
NodeAdapter.UDPServer.MaxTransmissionTime = 1000
NodeAdapter.UDPMulticastAddress = 224.242.224.243/2424
```

Starting/Joining an overlay network

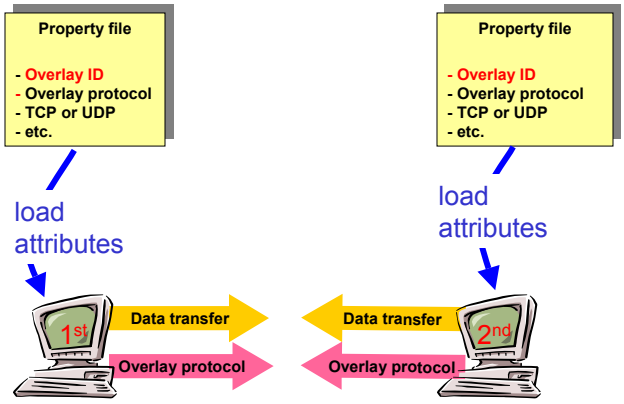
```
1. OverlayManager om = new OverlayManager("hypercast.prop");
2. String overlayID = om.getDefaultProperty("OverlayID")
3. OverlaySocketConfig config = new om.getOverlaySocketConfig(overlayID);
4. OL_Socket socket = config.createOverlaySocket(callback);
5. socket.joinGroup();
```

1. Read attributes from property file
2. Match attributes and overlay ID
3. Create configuration object
4. Create overlay socket
5. Join overlay



DENALI

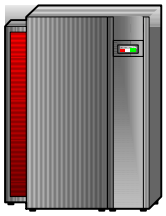
Starting an Overlay



DENALI

Overlay Server

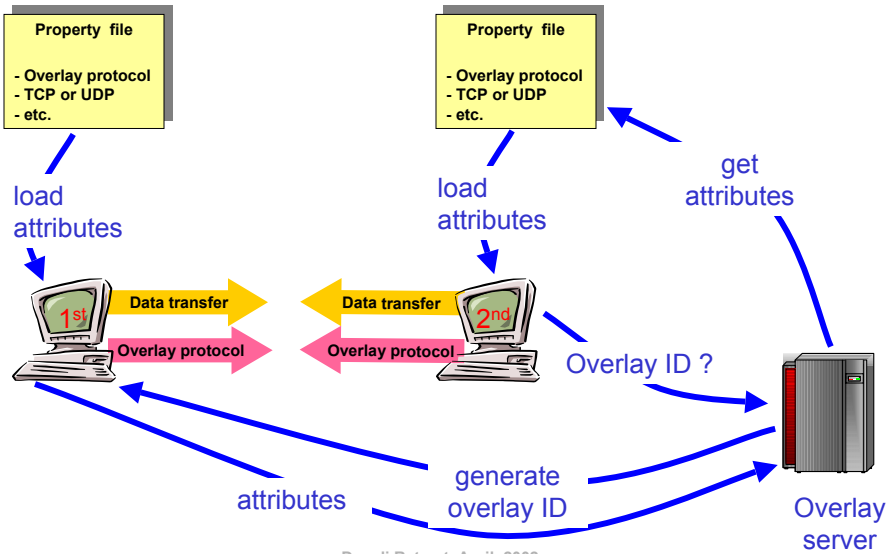
- Overlay server can help with the management of overlay attributes
  - Can generate Overlay IDs
  - Can store attributes
  - Can respond to queries for attributes
  - Can provide access control to attributes
- Overlay server is implemented as a minimal http server
- Attribute in the property file tells if an overlay server is used or not



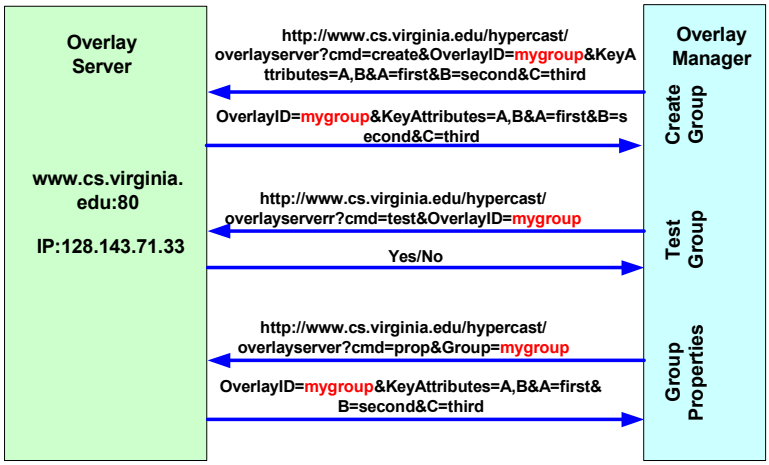
Overlay server

DENALI

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 properties of an overlay network
- Property files stores attributes

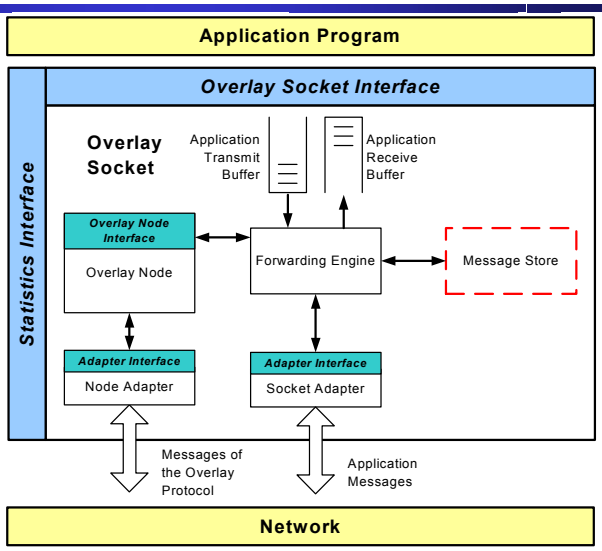
- Overlay is started from property file
- Attributes of an overlay can be stored at overlay server
  - Interface to overlay server uses HTTP and CGI queries

Some Features

- Design separates overlay maintenance (overlay protocol) from data transport
- Data transport over UDP or TCP
- Data transport uses formatted messages, modeled after IPv6
- Current implementation is in Java (SDK 1.2 or higher)

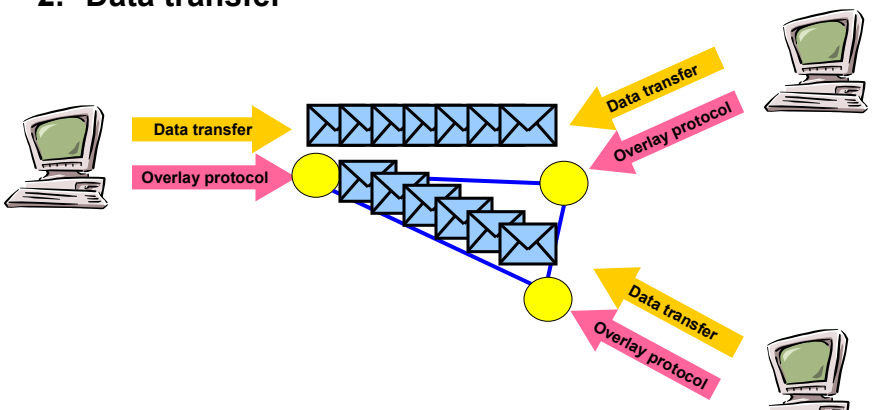
Overlay Socket: Components

- Components are configured when the overlay socket is created
- Two transport level ports are used:
  - Data transfer
  - Overlay protocol



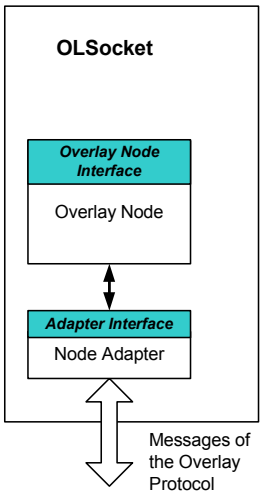
Separation of overlay protocol from data transfer

- **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 UDP message with neighbors in the overlay network

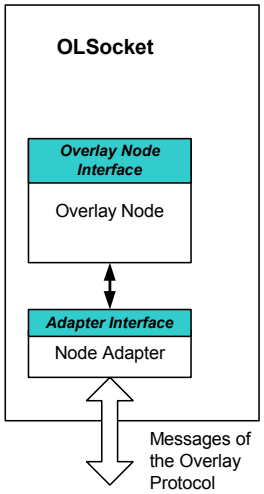


Overlay Node

- Each overlay node maintains a neighborhood table which contains a list of its neighbors in the overlay network
- Each entry of a neighborhood table contains:
  - the logical address of the neighbor
  - physical address of the neighbor

Logical address	*Physical address*
(x, y)	*128.143.137.21 / 2233*
(a, d)	*128.143.71.144 / 2567*
…	*…*

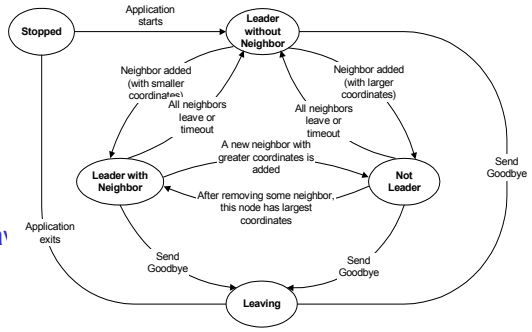
- All overlay protocols that must be able to compute:
  Given the logical address of some overlay node R, each overlay node with logical address A must be able to compute the logical address of A's parent and child nodes in an embedded tree which has R as the root.



DENALI

Overlay Protocol

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



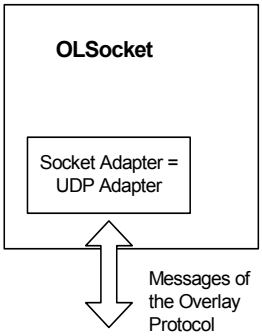
- Message format of DT protocol

1 byte	4 bytes	14 bytes	14 bytes	14 bytes	14 bytes
Type	OverlayID Hash	SRC	DST	ADDR1	ADDR2

DENALI

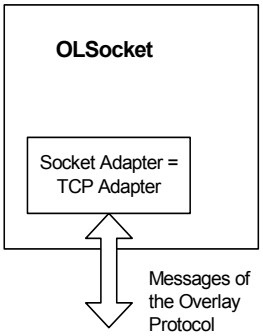
Types of Sockets

Connectionless (CL) overlay sockets



Application messages are exchanged as UDP unicast datagrams between neighbors in the overlay network.

Connection-Oriented (CO) overlay sockets



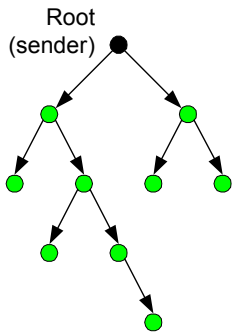
TCP connections are used to exchange application messages between neighbors in the overlay network

DENALI

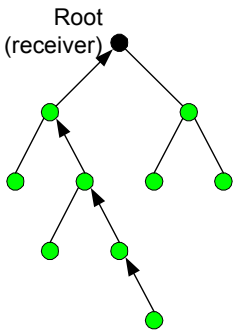
Unicast and Multicast in overlays

- Unicast and multicast is done using trees that are embedded in the overlay network.
- Requirement: Overlay node must be able to compute the child nodes and parent node with respect to a given root

Multicast



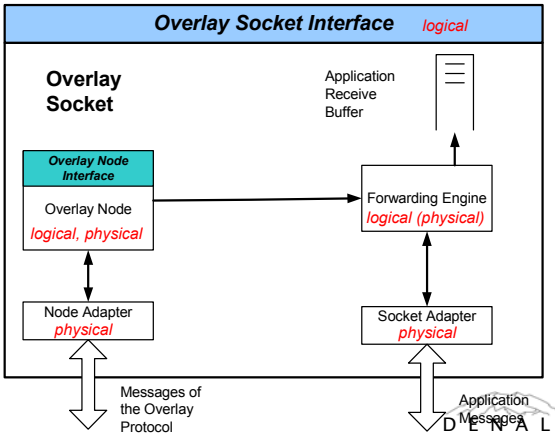
Unicast



DENALI

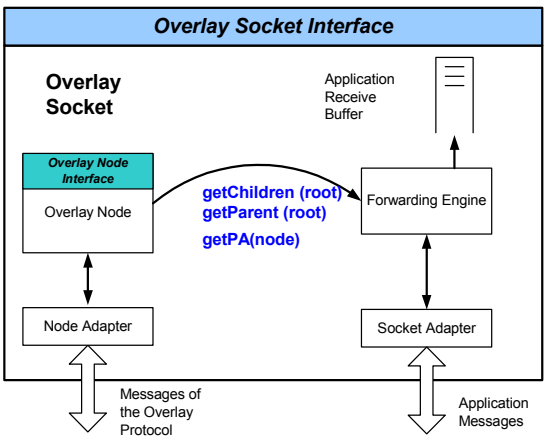
Logical addresses vs. physical addresses

- **Logical address (LA):** overlay specific addresses, e.g., coordinates in DT protocol
- **Physical address (PA)**: transport level address, e.g., IP address + port number



Forwarding Engine

- Forwarding Engine performs functions of a “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”



Send

SendToAll(Data) {

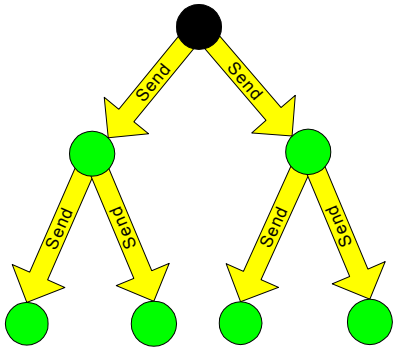
// Build the message

// Get the list of children from overlay node

// Get physical address of children

// Send message to children nodes

}

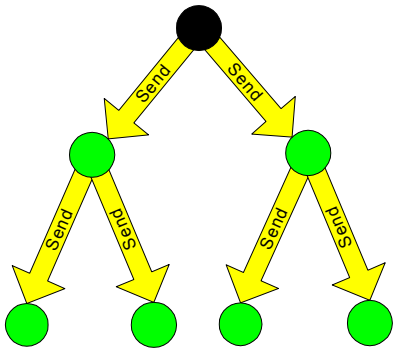


DENALI

Receive and Forward

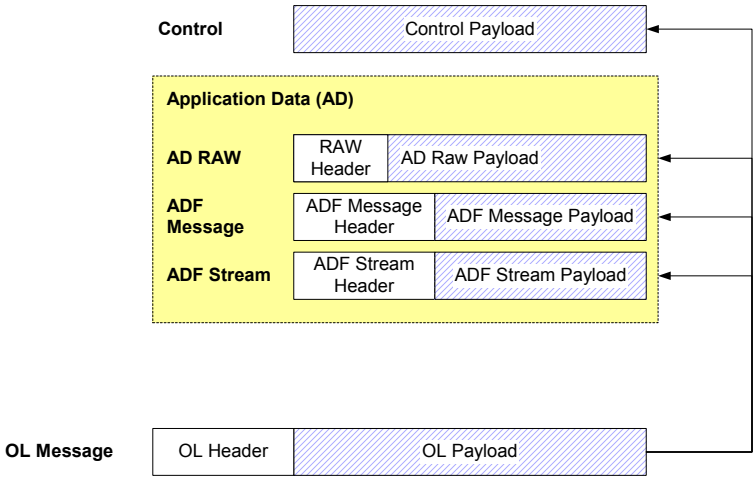
OL_Forward() {
// 1. Forward packet
// Determine the children in the tree
// Send datagram to children nodes

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



DENALI

Message Formats



DENALI

OL Header

Loosely modeled after IPv6 → minimal header with extensions

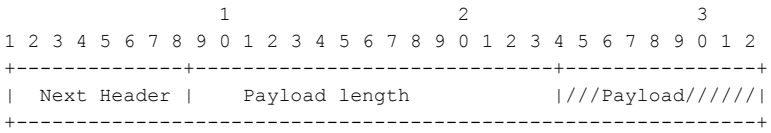
- **Common Header of "Overlay Message":**

```
                     1                   2                   3
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-------+---------------+---+---+---------------------------------+
|Version|LAS|Dmd| Traffic Class |   Flow Label    | Next Header   |
+-------+---------------+---+---+---------------------------------+
|  OL Message Length            |          Hop Limit              |
+-------------------------------+---------------------------------+
|                           Src LA                                |
+-----------------------------------------------------------------
|                          Dest LA                                |
+-----------------------------------------------------------------+
```

Version (4 bit):	Version of the protocol (current Version is 0x0)
LAS (2 bit):	Size of logical address field
Dmd (4bit)	Delivery Mode (Multicast, Flood, Unicast, Anycast)
Traffic Class (8 bit):	Specifies Quality of Service class (default: 0x00)
Flow Label (8 bit):	Flow identifier
Next Header (8 bit)	Specifies the type of the next header following this header
OL Message Length (8 bit)	Specifies the type of the next header following this header.
Hop Limit (16 bit):	TTL field
Src LA ((LAS+1)*4 bytes)	Logical address of the source
Dest LA ((LAS+1)*4 bytes	Logical address of the destination

DENALI

Message Formats: “Raw Message”



- **Payload Length:** Length of the Payload field in bytes

DENALI

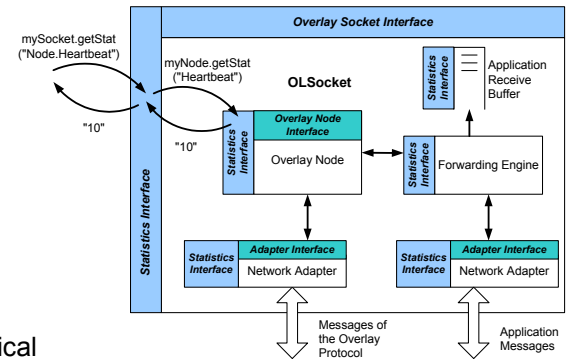
Monitor and Control Infrastructure

- Loosely modeled after SNMP, but more modern
  - Each Socket component collects statistics
  - Statistics are accessible via a statistics interface
  - Statistics are accessed at a portal by a monitor
  - Monitor and portal send queries and responses in XML messages

DENALI

Statistics Interface

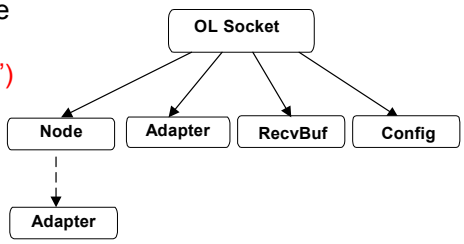
- Each component of socket provides statistics
- Statistics are accessed and changes with 3 calls:
  - getStat()
  - setStat()
  - getSchema()
- All parameters are strings
- Statistics have an hierarchical structure



DENALI

Naming of statistics

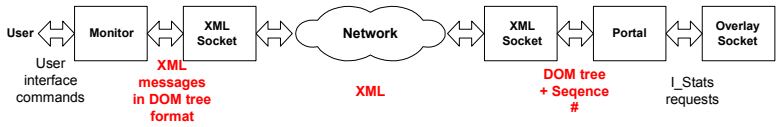
- Statistics are given as name-value pairs: (“mySocket.Node.Heartbeat”, “20”) indicates that 20 is the value of Heartbeat in the overlay node component of an overlay socket.



- Calling mySocket.getStat(“Node.Heartbeat”) to the overlay socket requests the value of the statistics “Heartbeat” in the overlay node component of an overlay socket with name “mySocket”

DENALI

XML



- When transmitted over the network all statistics requests and changes are sent as XML documents
- 3 types of XML messages: GetValue, SetValues, GetSchema
- We use schemas to describe format of documents
- Possible interaction:
  - What statistics do you have (“GetSchema”)
  - Look at schema and ask for specific value (“GetValue”)

DENALI

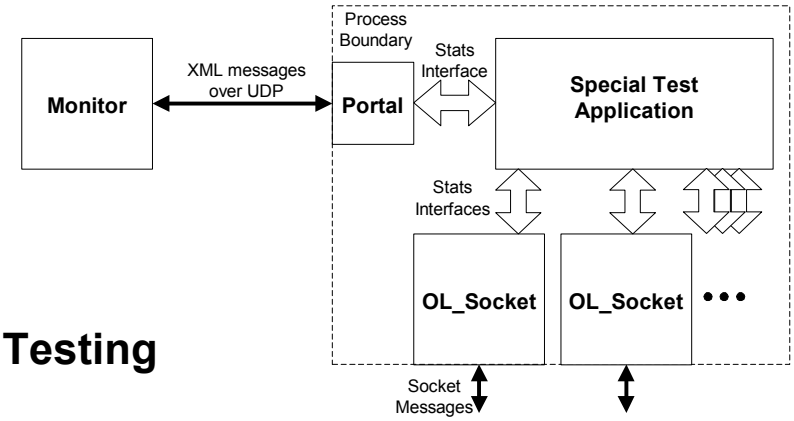
Comparison with SNMP

Portal	Agent
Monitor	Monitor
Schema	MIB
Hierarchical names	Object identifier
XML messages	SNMP protocol

DENALI

Running and Monitoring Experiments

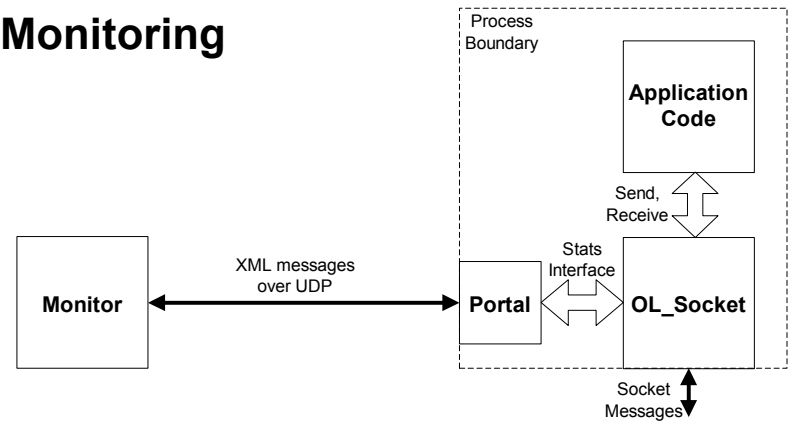
Testing



DENALI

Running and Monitoring Experiments

Monitoring



DENALI

Dynamic Discovery of Monitors