

JAdhoc System Design

JAdhoc System Design Manual

1) <u>AODV OPERATIONS</u>	2
AODV IMPLEMENTATION REQUIREMENTS	2
2) <u>JADHOC DESIGN: APPLICATION ARCHITECTURE</u>	4
3) <u>JADHOC DESIGN: CLASS ARCHITECTURE</u>	5
PROCEDURAL OBJECTS:	6
INFORMATIONAL OBJECTS	8
4) <u>DATA DICTIONARY</u>	11
A) CONFIGINFO	11
B) CURRENTINFO	11
C) ROUTELIST	11
D) RREQIDLIST	12
E) ROUTEENTRY	12
F) PRECURSOR	12
G) RREQIDENTRY	12
H) IPPACKET	12
I) LOGGING	13
J) RREQ	13
K) RREP	13
L) RERR	14
M) RREP-ACK	14
5) <u>OPERATING SYSTEM CONSIDERATIONS</u>	15
A) LINUX IPV4	15
B) LINUX IPV6	16

1) AODV operations

AODV uses 4 types of control messages. They are known as Route Request (RREQ), Route Reply (RREP), Route Acknowledgment (RREP-ACK) and Route Error (RERR) Sequence numbers in AODV play a key role in ensuring loop freedom. Every node maintains a monotonically increasing sequence number for itself. It also maintains the highest known sequence number for each destination in the routing table, called destination sequence number which is tagged with RREQ, RREP and RERR messages.

When an originator needs a route to a destination, it initiates a route discovery process. Route discovery involves a network wide flood of RREQ targeting the destination and waiting for a RREP. An intermediate node receiving the RREQ, first set up a reverse path to the originator using the previous hop of the RREQ as the next hop on the reverse path. If the intermediate node has a valid route to the destination and 'D' bit [3] is not set in the RREQ, the intermediate node generates a RREP to the originator and a gratuitous RREP to the destination; else the RREQ is re-broadcast. Duplicate copies of RREQs received at any node are discarded (duplicate copies are identified by RREQ ID maintained for the originator). When the destination receives a RREQ, it also generates a RREP. The RREP is routed back to the originator via the reverse path. As the RREP proceeds towards the originator, a forward path to the destination is established. The originator can send RREP-ACK to the destination upon receiving the RREP in order to make certain of the reliability of the bi-directional path. Each node which has active routes, starts multicasting a special RREP message of whose Time To Live (TTL) field is set to 1. This is called a HELLO message. The link layer connectivity to the immediate neighbors can be detected using the link layer information or listening to HELLO messages. When a node detects the link failure, a route error (RERR) is sent back via separately maintained precursor lists to all originators that are affected by failed link. When the originator receives the RERR, it can start a route discovery process, again. Unused routes in the routing table are expired using a timer-based technique.

AODV Implementation Requirements

The AODV protocol is a reactive protocol and therefore it must react to the changes in the IP environment to set the host computer's routing environment to the needs of the other nodes in the AODV network on demand. The above mentioned behavior of the AODV protocol requires, a certain set of requirements for which, the programming language used must provide capabilities. Following are these requirements and how they have been addressed.

1) Unicast and Broadcast Communications: The AODV controlling messages are UDP packets and should be destined to the 654 port of the recipient. The RREQ and RERR messages, which need to be heard by all the nodes in the vicinity, have to be broadcast. The other 2 messages of AODV, the RREP and RREP-ACK require unicast transmission. The Java language allows the sending of messages in this manner using the classes in the java.net package¹.

¹Sun Microsystems, Custom networking [on line]. Available: <http://java.sun.com/docs/books/tutorial/networking/index.html>

JAdhoc System Design

2) Listen to IP Traffic on a Given Network Interface: The protocol handler requires listening to IP traffic on the AODV capable network interface for the following reasons.

- a) To initiate a route discovery when the originating node requires to communicate with another node in the AODV network,
- b) To obtain the activity information of each of the currently available routes (which were created due to the requirement of the host node or other nodes in the AODV network),
- c) To listen to AODV control messages.

This capability is obtained by using a class library called Jpcap². This is a Java wrapper for the packet capture interface available in most of the operating systems of today.

3) Manipulation of the Routing Table: The AODV protocol requires that the routing table be set according to the requirements of the nodes in the AODV network. This requires the protocol to instruct the operating system to set and remove routing entries. The Java language does not contain facilities that provide the direct access of the routing table. Therefore, it should use system commands to manipulate the routing table.

4) Support a Timer Mechanism: The AODV protocol requires that a set of timers are made active to manage the different lifetimes of different processes. Some of these timers perform different tasks after they reach a certain time and expire themselves. Another of these, continue to be active until the protocol handler is active. Java provides multi-threading capabilities³ that can be used to implement this timer behavior.

- a) *Discovery Minder*– Manages the route discovery process, by sending RREQs until a route is made to a destination or until the process decides that the destination is not reachable. The ‘not reachable’ decision is made by using the Expanding Ring Search (ERS) mechanism or non-ERS mechanism.
- b) *RREQ Minder*– Manages the expiration of RREQ IDs used by each of the RREQs generated in route discovery processes
- c) *Hello Minder*– Manages the sending of HELLO messages to the immediate neighbours
- d) *Route Minder*– Manages the lifetime of an active route
- e) *Delete Minder*– Manages the lifetime of a route about to be deleted from the cached routing table.
- f) *Buffer Minder*- Manages the packet buffering and releasing

² K. Fujii, Java package for packet capture. Available: <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html>

³John Zukowski, “Mastering Java 2”, 1st ed, New Delhi, BPB, 2000, ch. 8

JAdhoc System Design

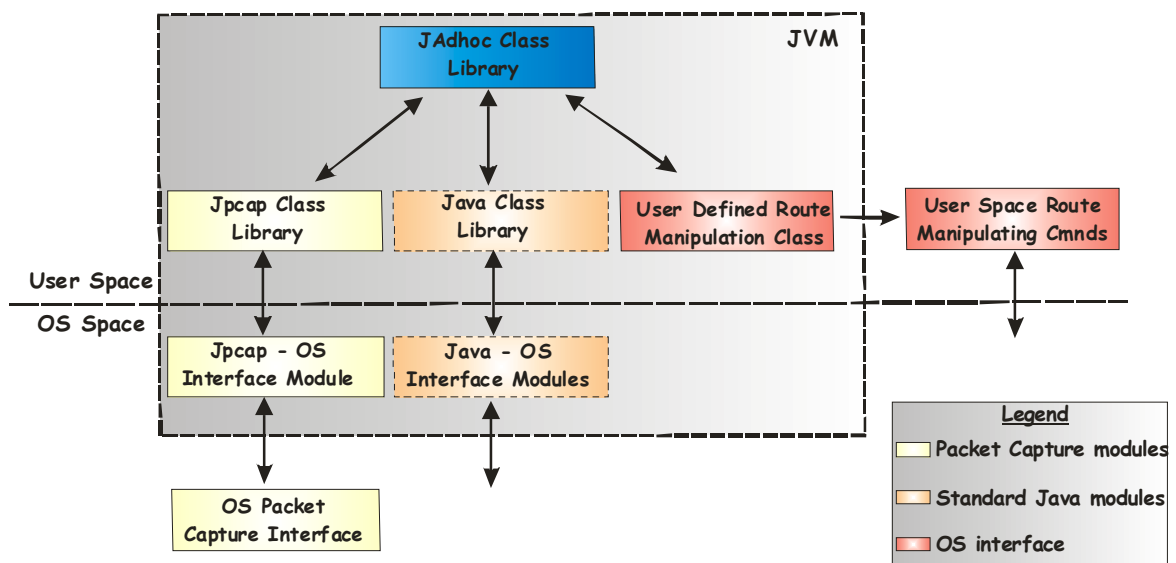


Figure 1 Application architecture of JAdhoc

2) JAdhoc Design: Application Architecture

This protocol handler was implemented in Java and named as JAdhoc. It consists of a class library, called JAdhoc that provides the AODV operations as explained in section 1. This module interacts with the Java class library, Jpcap class library and the user defined route manipulation class. The Fig. 1 provides a high level overview of the different components of this architecture.

JAdhoc Class Library: The JAdhoc Class Library (fig. 1) provides a set of classes that serves as the AODV protocol handler. In general, it consists of the class categories of Central Coordination, Network Interfacing, Threading, User Interfacing and Operating System Interfacing. The contents of this class library are explained in detail in the next section.

Jpcap Class Library and Jpcap-OS Interface Module: These 2 modules provide a Java Native Interface (JNI) based interface to the packet capture module of an operating system. The Jpcap Class Library provides the Java interface to the upper level applications while the Jpcap-OS Interface Module provides the OS native module to interact with the packet capture functions.

User Defined Route Manipulation Class: This provides route environment manipulation commands to the JAdhoc class library. The JAdhoc class library provides a Java interface, that need to be implemented to provide route manipulation functions. This is the only area of this protocol handler that does not provide portable code. To make this as easy as possible, a plug-in like mechanism was adopted, where a single class needs to be implemented for every platform on which the protocol handler is executed. The methods that need to be implemented are specified in a Java interface. This has the same interfacing characteristics of any Java Data Base Connectivity

JAdhoc System Design

(JDBC) driver⁴.

JAdhoc uses Jpcap to listen to AODV controlling messages and IP packets that traverse a given network interface. When a node sends a packet to a destination, that does not contain a route entry on the node's routing table, it uses the gateway specified in the default route. JAdhoc knows to start a route discovery based on following methods.

Unix Environment: JAdhoc sets the default route to the loop back interface. By doing this, JAdhoc can figure that any packet that comes to the loop back interface with the destination address which is not the 127.0.0.1, is in need of a route. This will result in the initiation of a route discovery process as defined in the section 1. While the route discovery is in progress, all packets that come to the loop back interface are buffered, to be dispatched once the route is made. JAdhoc monitors the given network interface and the loop back interface continuously. When a packet comes to the given network interface, it considers this to be a packet that is using an existing route and update the lifetime of a valid route.

Windows environment: The above method can't be adopted in windows environment, since it does not provide to set the default route to the loop back interface. Therefore, JAdhoc sets the MAC address of the default gateway to 00:00:00:00:00:00. This will result in the interface sending out packets that do not have routes with MAC address 00:00:00:00:00:00. These packets are picked by Jpcap, buffered and the protocol handler initiates the route discovery.

3) JAdhoc design: Class Architecture

The class architecture of JAdhoc adopts a hybrid design pattern architecture, consisting of the Gang Of Four (GOF) Design Patterns⁵ and J2EE Pattern Catalog⁶. In general, there are 2 categories of objects. The objects which are related to managing the procedural tasks of the AODV Protocol Handler, are referred to as Procedural Objects and secondly, the objects that relate to holding information, are referred to as Informational Objects. In addition to these objects, it also uses the objects of the Java Collections class library (Hash Maps and Link Lists) extensively, to manage different packet queues, route lists and precursor lists. Following is a general outline of the objects used in the JAdhoc protocol handler.

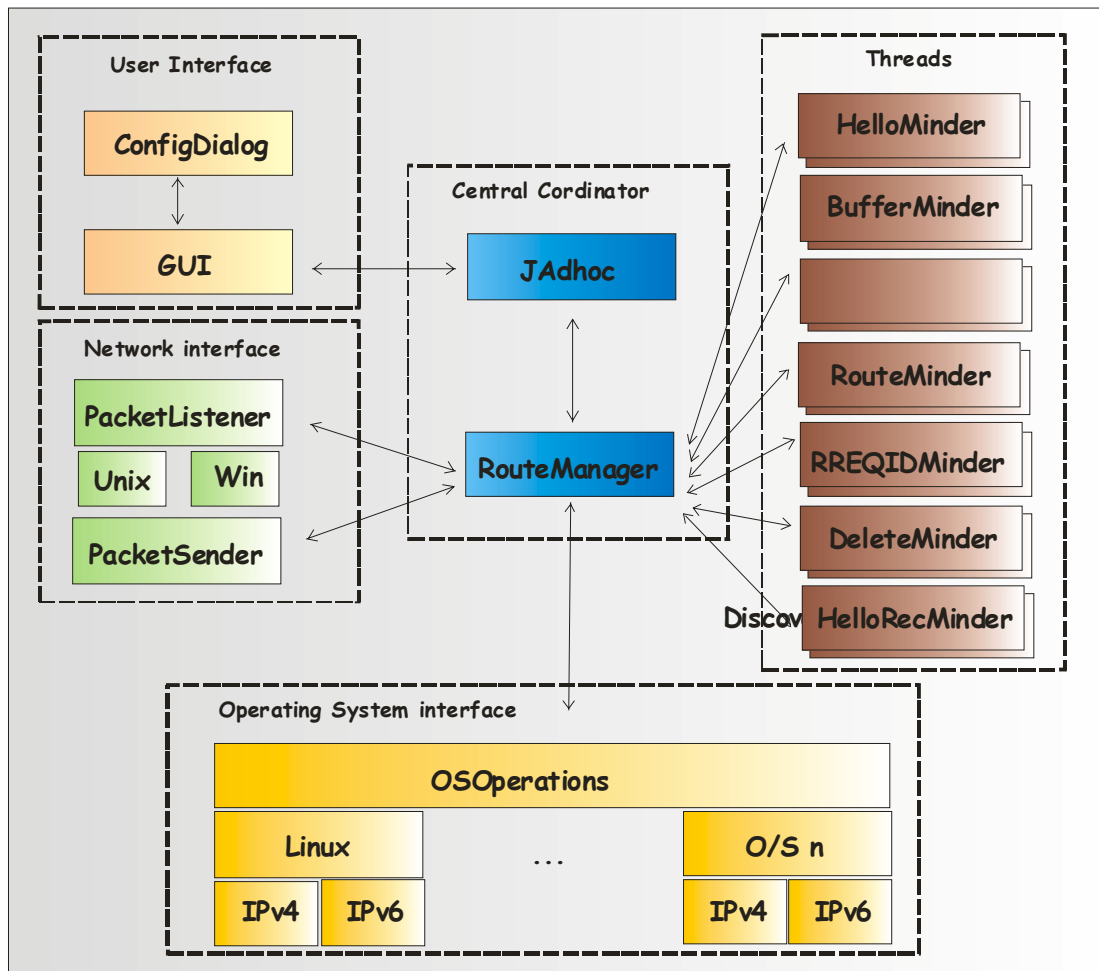
Figure 2 Class Architecture of JAdhoc

⁴ K. Walrah, M. Campione and A. Hum, "The java tutorial continued", Addison-Wesley, 1998, pp. 291-332.

⁵ J. W. Cooper, "The design patterns Java companion", New York: Addison-Wesley, 1998

⁶ Sun Microsystems, J2EE design patterns [Online]. Available: <http://java.sun.com/blueprints/patterns>.

JAdhoc System Design



Procedural Objects:

These objects, as explained earlier, refer to objects that manage the operations of the AODV Protocol Handler. They include listeners, message dispatchers, list manipulators, minders and a node's routing environment manipulator. The Fig. 2 provides a graphical view of the procedural objects of JAdhoc with their groupings and the inter-relationships.

JAdhoc – The starting object of JAdhoc. This is a Singleton and creates the RouteManager and the GUI objects. It passes itself to these 2 objects, to act as the intermediary between requests of these 2 objects. The requests that come from the RouteManager is to refresh the GUI and requests that come from the GUI is to obtain current state of the protocol handler. The GUI object creation is conditional and this is based on whether the application runs on GUI mode or non-GUI mode.

GUI– This object manages the graphical user interface component of JAdhoc. It is programmed

JAdhoc System Design

using the Swing classes⁷ of Java. The user interface provides the facilities for the user to start, stop and configure the protocol handler and provides a grid that displays information about the current status of each route (refer fig. 2). This is a Singleton type object. To configure JAdhoc, this object creates a ConfigInfo object providing itself as the parent.

ConfigDialog– This object, a Singleton, is invoked when the user requires updating the configuration information. It provides a grid that lists the parameter values and the user can change these values and update.

RouteManager– This object is the central object of the protocol handler that initiates and coordinates the activities of JAdhoc. It is a Singleton and a Mediator. When JAdhoc starts, it creates the PacketListener, PacketSender, HelloMinder, RREQIDMinder and the OSOperations objects. When JAdhoc is running, it creates DiscoveryMinder, RouteMinder and DeleteMinder objects based on the different states of each of the routes. When creating each of these objects, it provides itself as a parameter. These objects call the services made available with the RouteManager to perform different activities. Some of the methods of this Singleton object are synchronized. This ensures the thread safety of common services used by different threads.

PacketListener – This object listens to packets that traverse the network interface of the host node. It is an independent Thread. It is structured on a Singleton pattern. It has internal methods to distinguish between the operations of IPv4 & IPv6. This object uses the Jpcap class library to retrieve packets from the given network interface. There are 3 types of packets; packets that come to the loop back interface, packets that are AODV control messages (protocol UDP & destination port 654), and other packets. For every one of these types, it calls the services provided in the RouteManager object.

PacketSender – This object, a Singleton allows the sending of AODV control messages and IP packets which were buffered when discovering routes. For AODV messages, it uses a Java socket where messages can either be multicast or unicast. For IP packets, it uses Jpcap.

HelloMinder– This object is a Singleton that manages the generation of AODV HELLO messages. These messages are RREPs, but with some specific information. This is an independent thread that sends the HELLO messages every number of seconds as specified in the ConfigInfo object (HELLO_INTERVAL). This sends HELLO messages only if an active route is present.

BufferMinder- This is a thread object that decides the time of releasing the buffered packets, once the route is actually made in the Kernel.

RouteMinder– This is a thread object that manages a lifetime of a route, when it is active. This thread checks a routing entry's expiry time regularly and if the time has expired, it statuses the route as being expired and removes it from the OS routing table. These objects are instantiated by the RouteManager, when a route becomes active.

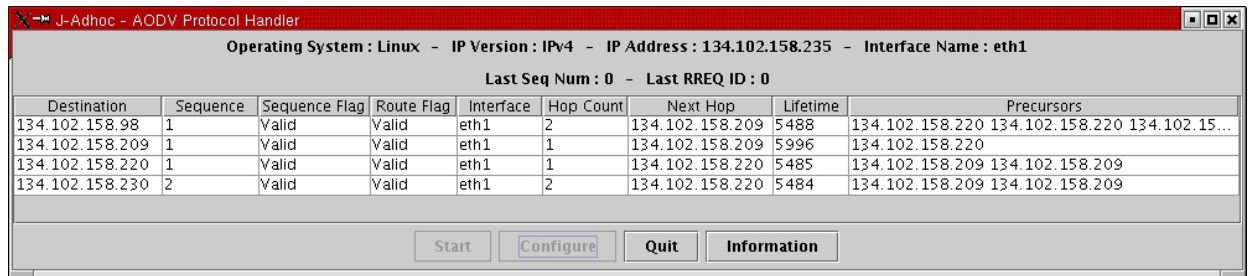
RREQIDMinder– This object, a Singleton is a thread that manages the RREQ ID numbers so that they are not acted upon over and over again. Each RREQ ID has a specified lifetime as

⁷ K. Walrah and M. Campione, "The JFC swing tutorial", 3rd ed, New York: Addison-Wesley, 2000, ch. 22-32.

JAdhoc System Design

defined in the ConfigInfo object (PATH_DISCOVERY_TIME).

DiscoveryMinder– This is a thread object that controls the dissemination of RREQs when performing a route discovery. It stays active until the route is made or the discovery process is expired. A discovery minder is created for each destination which requires a route.



The screenshot shows a window titled "J-Adhoc - AODV Protocol Handler". The status bar indicates: "Operating System : Linux - IP Version : IPv4 - IP Address : 134.102.158.235 - Interface Name : eth1". Below this, it shows "Last Seq Num : 0 - Last RREQ ID : 0". The main area contains a table with the following data:

Destination	Sequence	Sequence Flag	Route Flag	Interface	Hop Count	Next Hop	Lifetime	Precursors
134.102.158.98	1	Valid	Valid	eth1	2	134.102.158.209	5488	134.102.158.220 134.102.158.220 134.102.15...
134.102.158.209	1	Valid	Valid	eth1	1	134.102.158.209	5996	134.102.158.220
134.102.158.220	1	Valid	Valid	eth1	1	134.102.158.220	5485	134.102.158.209 134.102.158.209
134.102.158.230	2	Valid	Valid	eth1	2	134.102.158.220	5484	134.102.158.209 134.102.158.209

At the bottom of the window, there are four buttons: "Start", "Configure", "Quit", and "Information".

Figure 3 Graphical User Interface of GUI

DeleteMinder– This is a thread object that manages an invalid route during its deletion lifetime. Each invalid route has a specified lifetime as defined in the ConfigInfo object (DELETE_PERIOD). Once the DELETE_PERIOD expires the invalid entry is removed from the cached routing table.

HelloReceiptMinder- This is a thread object that listens to the Hello messages of each active neighbor in order to detect the link break as defined in RFC 3561.

OSOperations– This object provides all the services required to manipulate the routing environment on the host. This is a Proxy and a Singleton. As a Proxy, it passes the routing environment manipulation commands to another object which provides services specific to a configured operating system and IP version.

Informational Objects

The above description provides a brief explanation about the procedure oriented objects of JAdhoc. In addition to these, there are a set of objects that hold different information that is passed between the above mentioned objects. These are referred to here as Informational Objects. In J2EE Pattern Catalog terminology, they are called Transfer Objects. The key characteristic of these Informational Objects is that they are passed between different Procedural Objects, when they perform tasks. Following is a list of these simple, data oriented, Informational Objects.

ConfigInfo – This object holds all the configuration information obtained from the configuration file. It is given to all the previously mentioned objects to extract specific configuration information. This object is created by the JAdhoc object.

CurrentInfo – This object holds information that is unique to an invocation of JAdhoc. These include the current values of the RREQID and Sequence Number of the host node.

RouteList – This object manages a Hash Map that holds information about routes. It contains a set of methods to manipulate the hash map.

JAdhoc System Design

RREQIDList – This is a wrapper object for a Link List that holds entries related to RREQ IDs that have been used in route discoveries. These RREQ IDs have a lifetime and they are removed from this list when the lifetime expires.

RouteDiscoveryList – This is again a wrapper, but of a Hash Map. This Hash Map holds information related to destinations for which route discovery processes are currently active. These IP addresses are removed, when the route is made or when the route discovery expires.

RouteEntry – A single object holds information related to a single route. There will be many of these objects instantiated and they will be entries in the *RouteList* (mentioned above).

RREQIDEntry – This object holds information related to a single RREQ ID. There can be many of these objects, one per every RREQID used, placed in the *RREQIDList*. These will be invalidated when a certain time expires (PATH_DISCOVERY_TIME).

RouteDiscoveryEntry – This object holds information related to a single route discovery process. It includes the destination IP address and the last RREQ message sent.

Logging – This object provides services to log all activities of the Protocol Handler. It uses a text file to place all activity information. The name of the log file and whether logging is enabled is specified in the configuration file.

AODVMessage – This is the parent class of all AODV controlling messages. It holds the common information related to an AODV controlling message such as destination/source IP addresses, TTL, network interface, etc.

RREQ – Object that holds information related to a RREQ message. RREQ messages can be created due to the host node performing a route discovery or other nodes performing route discoveries.

RREP– Object that holds information related to a RREP message. These messages can be created by the local node wanting to respond to a route discovery or due to other nodes responding to route discoveries.

RERR– Object that holds information related to a RERR message. These messages can get created due to the host node wanting to send the RERR due to a link break to a neighboring node or other nodes doing so.

RREP-ACK– Object to hold information of a RREP-ACK message, which is generated if the RREP message has the Acknowledgement flag set.

HELLO– This is a special RREP message that holds a set of fixed information. It subclasses the RREP class.

IPpkt– This object holds information related to an IP packet, including the MAC information. These objects are created by the PacketListener for every non-AODV packet that it hears on the

monitoring interface.

4) Data Dictionary

The Protocol Handler consists of a set of objects that are termed as Transfer Objects. The responsibility of these objects is to hold all information related to the Protocol Handler. These objects would be passed all over the other objects.

a) ConfigInfo

Attribute Name	Data Type
Execution Mode (GUI or no-GUI mode)	String
Operating System (Linux, O/S 2, O/S 3, ... O/S n)	
IP Version (IPv4 or IPv6)	
IP Address	
Interface Name	
IP Address of the Gateway in Default Route	
MAC Address of Gateway in Default Route (default value – FF:FF:FF:00:00:00)	
Logging Status (Yes or No)	
Log File (/var/log/jadhoc.log)	
Path to System Commands (\sbin)	
Only Destination Flag (Yes or No)	
Gratuios RREP Flag (Yes or No)	

b) CurrentInfo

Attribute Name	Data Type
Sequence Number	int
AODVMsgSocket	DatagramSocket
ActivityLog	Logging

c) RouteList

Attribute Name	Data Type
RouteList	HashMap

d) RREQIDList

Attribute Name	Data Type
RREQIDList	HashMap

e) RouteEntry

Attribute Name	Data Type
Destination IP Address	String
Destination Sequence Number	int
Valid Destination Sequence Number	int
Route Status (Valid, invalid, repairable, being repaired)	String
Network Interface	String
Hop Count	int
Next Hop (IP Address)	String
List of Precursors	HashMap
Lifetime	int
Route Discovery Status (Route being discovered or not)	String

f) Precursor

Attribute Name	Data Type
Precursor (IP Address)	String

g) RREQIDEntry

Attribute Name	Data Type
Destination IP Address	String
Destination Sequence Number	Int

h) IPPacket

Attribute Name	Data Type
Source IP Address (IPv4, IPv6)	String
Destination IP Address (IPv4, IPv6)	String
Protocol	int
Source Port	int

JAdhoc System Design

Destination Port	int
Source MAC Address	String
Destination MAC Address	String

i) Logging

Attribute Name	Data Type
Log File IO Stream	IOStream

j) RREQ

Attribute Name	Data Type
Message Transmission Mode (Multicast or Unicast)	String
Unicast IP Address	String
Join Flag	String
Repair Flag	String
Gratuitous RREP Flag	String
Destination Only Flag	String
Hop Count	int
RREQ ID	int
Destination IP Address	String
Destinatin Sequence Number	int
Originator IP Address	String
Originator Sequence Number	int

k) RREP

Attribute Name	Data Type
Message Transmission Mode (Multicast or Unicast)	String
Unicast IP Address	String
Repair Flag	String
Acknowledgement Required Flag	String
Prefix Size	int
Hop Count	int
Destination IP Address	String
Destinatin Sequence Number	int
Originator IP Address	String
Lifetime	int

JAdhoc System Design

l) RERR

Attribute Name	Data Type
Message Transmission Mode (Multicast or Unicast)	String
Unicast IP Address	String
No Delete Flag	String
Destination Count	int
Unreachable Destination IP Address	String
Unreachable Destinatin Sequence Number	int

m) RREP-ACK

Attribute Name	Data Type
Message Transmission Mode (Multicast or Unicast)	String
Unicast IP Address	String

5) Operating System Considerations

The OperatingSystem object provides all the O/S specific services to manipulate the machine's routing environment. This object calls another object which caters to the command set of a specific operating System. This object in turn calls another object, which performs the actual O/S command. This last object is specific to the IP version. This section specifies the command sequence for different operating systems, considering whether IPv4 or IPv6.

In general, there are 3 operating system services that these objects require. They are,

- Set of the routing environment
- Addition of a route
- Removal of a route

a) Linux IPv4

Set Routing Environment

Run **ip neigh** command and set MAC address FF:FF:FF:00:00:00 to the given gateway. E.g.

```
ip neigh add 134.102.158.1 lladdr ff:ff:ff:00:00:00 dev eth1
```

Addition of a Route

Run **ip route add** command to insert a route entry in the routing table. If the route is to an immediate node (1 hop), then the via option that specifies the gateway is not required. If it is another node (i.e. multi-hop), then the via and the net hop's IP address should be specified. E.g.

```
ip route add 134.102.183.34/32 via 134.102.34.1 dev eth1  
ip route add 134.102.183.34/32 dev eth1
```

Removal of a Route

Run **ip route del** command to delete a route from the routing table. It is the same as the route add command, but with add replaced by del. E.g.

```
ip route del 134.102.183.34/32 via 134.102.34.1 dev eth1  
ip route del 134.102.183.34/32 dev eth1
```

b) Linux IPv6

Set Routing Environment

Run **ip -6 neigh** command and set MAC address FF:FF:FF:00:00:00 to the given gateway. E.g.

```
ip -6 neigh add 1234:5678::1245 lladdr ff:ff:ff:00:00:00 dev eth1
```

Addition of a Route

Run **ip route add** command to insert a route entry in the routing table. If the route is to an immediate node (1 hop), then the via option that specifies the gateway is not required. If it is another node (i.e. multi-hop), then the via and the net hop's IP address should be specified. E.g.

```
ip -6 route add 3214:2314::5467/128 nexthop via 5678:4356:789::3456 dev eth1
ip -6 route add 3214:2314::5467/128 dev eth1
```

Removal of a Route

Run **ip route del** command to delete a route from the routing table. It is the same as the route add command, but with add replaced by del. E.g.

```
ip -6 route del 3214:2314::5467/128 nexthop via 5678:4356:789::3456 dev eth1
ip -6 route del 3214:2314::5467/128 dev eth1
```