RARP:

Reverse Address Resolution Protocol (RARP) is a protocol a physical machine in a local area network (LAN) can use to request its IP address. It does this by sending the device's physical address to a specialized RARP server that is on the same LAN and is actively listening for RARP requests.

Program:

Program which sends requests.

Client.java:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Client
{
public static void main(String args[])
{
try
{
DatagramSocket client=new DatagramSocket();
InetAddress addr=InetAddress.getByName("127.0.0.1");
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the Physical address (MAC):");
String str=in.readLine(); sendbyte=str.getBytes();
DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);
client.send(sender);
DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
client.receive(receiver);
String s=new String(receiver.getData());
System.out.println("The Logical Address is(IP): "+s.trim());
client.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

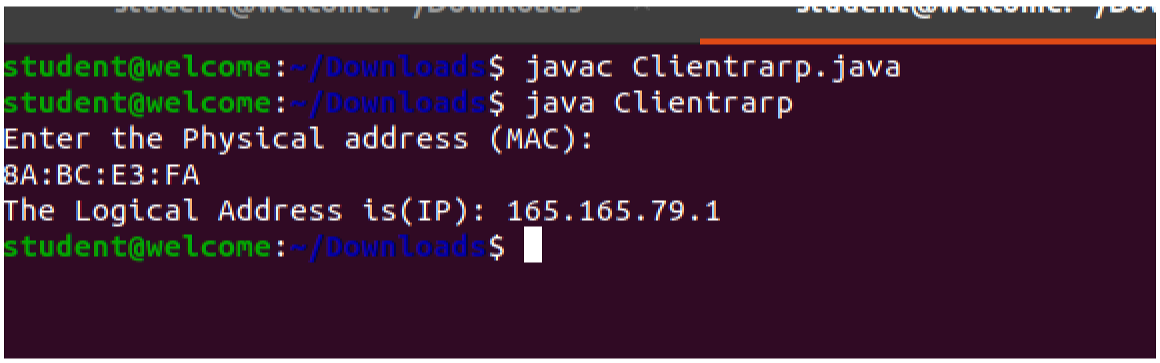
Program to send IP address based on request:

Server.java:

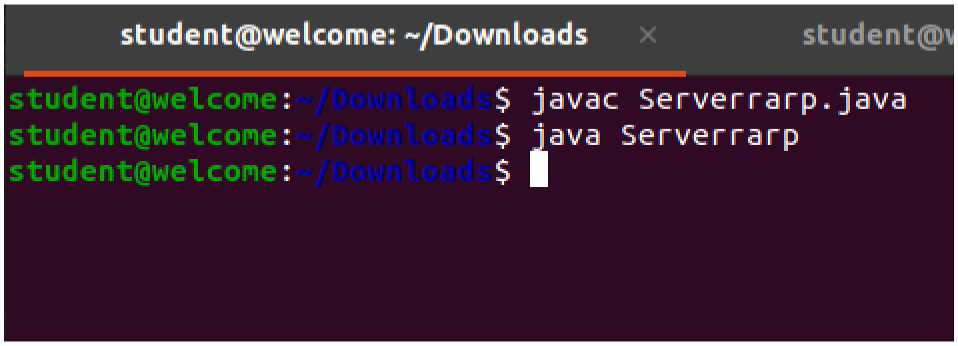
```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp
```

```
{
public static void main(String args[])
{
try
{
DatagramSocket server=new DatagramSocket(1309);
while(true)
{
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
server.receive(receiver);
String str=new String(receiver.getData());
String s=str.trim();
InetAddress addr=receiver.getAddress();
int port=receiver.getPort();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++)
{
if(s.equals(mac[i]))
{
sendbyte=ip[i].getBytes();
DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
server.send(sender);
break;
}
}
break;
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

OUTPUT:



```
student@welcome:~/Downloads$ javac Clientrarp.java
student@welcome:~/Downloads$ java Clientrarp
Enter the Physical address (MAC):
8A:BC:E3:FA
The Logical Address is(IP): 165.165.79.1
student@welcome:~/Downloads$
```



```
student@welcome:~/Downloads$ javac Serverrarp.java
student@welcome:~/Downloads$ java Serverrarp
student@welcome:~/Downloads$
```

ARP:

Client:

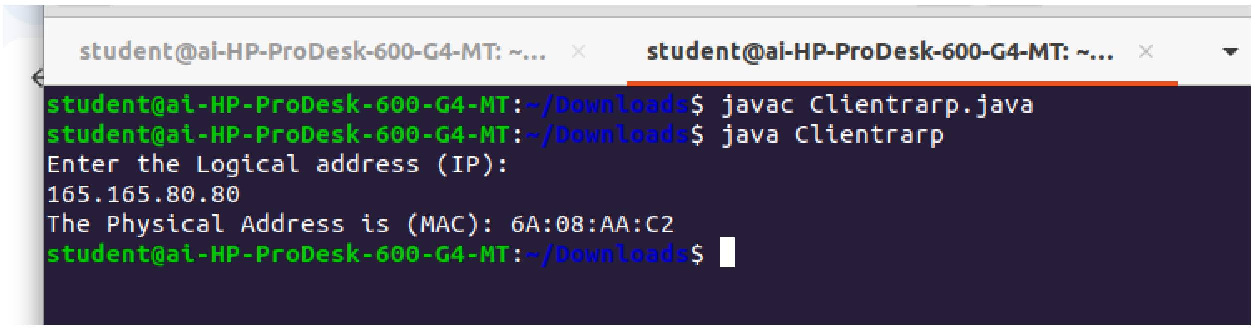
```
import java.io.*;
import java.net.*;

class Client {
    public static void main(String args[]) {
        try {
            DatagramSocket client = new DatagramSocket();
            InetAddress addr = InetAddress.getByName("127.0.0.1");
            byte[] sendbyte = new byte[1024];
            byte[] receivebyte = new byte[1024];
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter the Logical address (IP):");
            String str = in.readLine();
            sendbyte = str.getBytes();
            DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length,
addr, 1309);
            client.send(sender);
            DatagramPacket receiver = new DatagramPacket(receivebyte,
receivebyte.length);
            client.receive(receiver);
            String s = new String(receiver.getData());
            System.out.println("The Physical Address is (MAC): " + s.trim());
            client.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Server:

```
import java.io.*;
import java.net.*;
import java.util.*;

class Serverarp {
    public static void main(String args[]) {
        try {
            DatagramSocket server = new DatagramSocket(1309);
            while (true) {
                byte[] sendbyte = new byte[1024];
                byte[] receivebyte = new byte[1024];
                DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
                server.receive(receiver);
                String str = new String(receiver.getData());
                String s = str.trim();
                InetAddress addr = receiver.getAddress();
                int port = receiver.getPort();
                String ip[] = {"165.165.80.80", "165.165.79.1"};
                String mac[] = {"6A:08:AA:C2", "8A:BC:E3:FA"};
                for (int i = 0; i < ip.length; i++) {
                    if (s.equals(ip[i])) {
                        sendbyte = mac[i].getBytes();
                        DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length,
addr, port);
                        server.send(sender);
                        break;
                    }
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

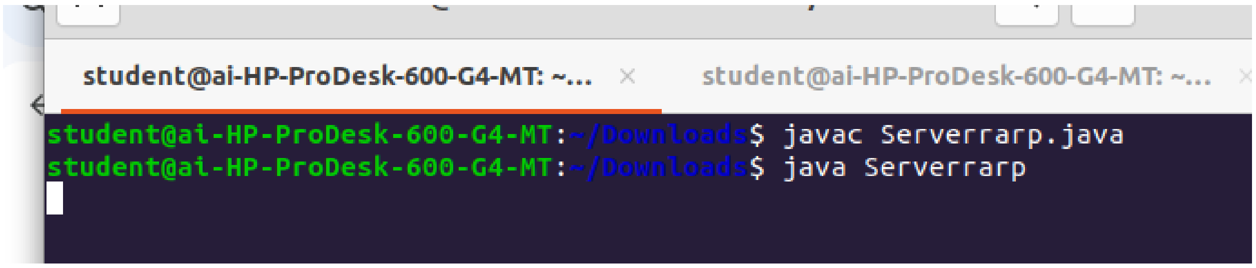
}

OUTPUT:



student@ai-HP-ProDesk-600-G4-MT: ~...   student@ai-HP-ProDesk-600-G4-MT: ~...

```
student@ai-HP-ProDesk-600-G4-MT:~/Downloads$ javac Clientrarp.java
student@ai-HP-ProDesk-600-G4-MT:~/Downloads$ java Clientrarp
Enter the Logical address (IP):
165.165.80.80
The Physical Address is (MAC): 6A:08:AA:C2
student@ai-HP-ProDesk-600-G4-MT:~/Downloads$
```



student@ai-HP-ProDesk-600-G4-MT: ~...   student@ai-HP-ProDesk-600-G4-MT: ~...

```
student@ai-HP-ProDesk-600-G4-MT:~/Downloads$ javac Serverrarp.java
student@ai-HP-ProDesk-600-G4-MT:~/Downloads$ java Serverrarp
```

Task:

IPC (Inter-Process Communication) Choice:

Sockets are my preferred method for IPC. Sockets offer a reliable and effective means of interprocess communication, particularly in a networked setting. ARP queries and answers can be broadcast over UDP (User Datagram Protocol). As UDP is lightweight and connectionless, it works well in this situation where ARP messages are usually brief and don't need to be reliable.

Data Structure Choice:

A dictionary-style structure or a hash table work well for keeping track of the mapping between IP addresses and MAC addresses. Constant-time lookup, which is effective for rapidly obtaining MAC addresses given an IP address, is provided via hash tables. We can make use of a dictionary in which the corresponding MAC addresses are the values and the IP addresses are the keys.

Simulation Steps:

1.Assigning IP Addresses:

- Each process is assigned a unique IP address during initialization.
- These IP addresses are stored in the MAC-IP mapping table maintained by one of the processes.

2.ARP Query:

- When a process needs to communicate with another process whose IP address it knows but MAC address it doesn't, it sends an ARP query.
- The querying process broadcasts an ARP request packet containing its own IP address and the target IP address it wants to communicate with.
- Other processes receive this broadcasted ARP request packet.

3.ARP Response:

- Upon receiving the ARP request packet, the process with the matching IP address sends an ARP response packet.
- The ARP response packet contains the MAC address corresponding to the requested IP address.
- This response is sent directly to the querying process.

ARP Server:

```
import java.io.Serializable;
import java.io.*;
import java.net.*;

public class ARPServer {
   private static final int ARP_PORT = 12345;

   public static void main(String[] args) {
      try (ServerSocket serverSocket = new ServerSocket(ARP_PORT)) {
         System.out.println("ARP Server started. Listening on port " + ARP_PORT);
         while (true) {
            try (Socket socket = serverSocket.accept();
                 ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
                 ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream())) {

                ARPPacket request = (ARPPacket) in.readObject();
                System.out.println("Received ARP request: " + request);

                // Process ARP request here

                // Simulate sending ARP response
                ARPPacket response = new ARPPacket();
                response.operation = Operation.ARP_RESPONSE;
                response.senderIP = request.targetIP;
                response.senderMAC = "00:11:22:33:44:55"; // Example MAC address
                response.targetIP = request.senderIP;
                response.targetMAC = "55:44:33:22:11:00"; // Example MAC address

                out.writeObject(response);
                System.out.println("Sent ARP response: " + response);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
         }
      } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

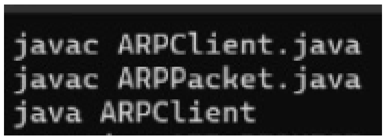
ARP Client:

```
import java.io.Serializable;
import java.io.*;
import java.net.*;

public class ARPClient {
    private static final int ARP_PORT = 12345;

    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", ARP_PORT);
             ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

            ARPPacket request = new ARPPacket();
            request.operation = Operation.ARP_REQUEST;
            request.senderIP = "192.168.1.2"; // Example sender IP
            request.senderMAC = "AA:BB:CC:DD:EE:FF"; // Example sender MAC
            request.targetIP = "192.168.1.4"; // Example target IP
            request.targetMAC = ""; // Empty MAC for request

            out.writeObject(request);
            System.out.println("Sent ARP request: " + request);

            ARPPacket response = (ARPPacket) in.readObject();
            System.out.println("Received ARP response: " + response);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

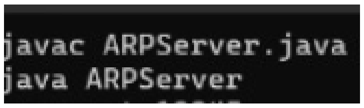
ARP Packet Class:

```
import java.io.Serializable;

// Define constants
enum Operation {
    ARP_REQUEST,
    ARP_RESPONSE
}
```

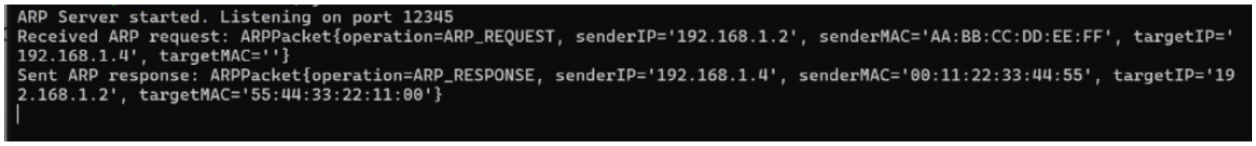
```
// Class representing an ARP packet
class ARPPacket implements Serializable {
    private static final long serialVersionUID = 1L;
    Operation operation;
    String senderIP;
    String senderMAC;
    String targetIP;
    String targetMAC;

    @Override
    public String toString() {
        return "ARPPacket{" +
                "operation=" + operation +
                ", senderIP='" + senderIP + '\'' +
                ", senderMAC='" + senderMAC + '\'' +
                ", targetIP='" + targetIP + '\'' +
                ", targetMAC='" + targetMAC + '\'' +
                '}';
    }
}
```



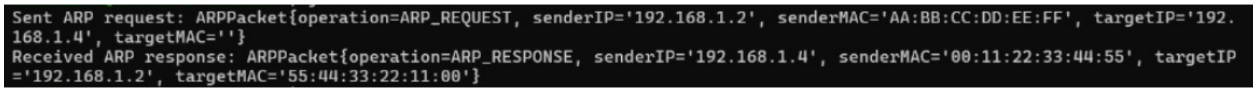
```
javac ARPClient.java
javac ARPPacket.java
java ARPClient
```



```
javac ARPServer.java
java ARPServer
```



```
ARP Server started. Listening on port 12345
Received ARP request: ARPPacket{operation=ARP_REQUEST, senderIP='192.168.1.2', senderMAC='AA:BB:CC:DD:EE:FF', targetIP='
192.168.1.4', targetMAC=''}
Sent ARP response: ARPPacket{operation=ARP_RESPONSE, senderIP='192.168.1.4', senderMAC='00:11:22:33:44:55', targetIP='19
2.168.1.2', targetMAC='55:44:33:22:11:00'}
|
```



```
Sent ARP request: ARPPacket{operation=ARP_REQUEST, senderIP='192.168.1.2', senderMAC='AA:BB:CC:DD:EE:FF', targetIP='192.
168.1.4', targetMAC=''}
Received ARP response: ARPPacket{operation=ARP_RESPONSE, senderIP='192.168.1.4', senderMAC='00:11:22:33:44:55', targetIP
='192.168.1.2', targetMAC='55:44:33:22:11:00'}
```