URMILA SV 21MIS1098


VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

# SWE2002 – COMPUTER NETWORKS

**Fall Inter Semester 2022-23**

## LAB ASSESSMENT - III

**Submitted by:**

Urmila SV-21MIS1098

**BRANCH**
INTEGRATED MTECH (SOFTWARE ENGINEERING)
MTECH5
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

**FACULTY**
Prof. Sofia Nishath

URMILA SV 21MIS1098

(a) Write a program to implement a simple message transfer from client to server using UDP sockets.

**Problem Definition :**We want to implement a simple messaging system between a client and a server using UDP sockets in Java. The client should be able to send a message to the server, and the server should be able to receive the message, modify it (in this case, convert it to uppercase), and send it back to the client.

**Algorithm :**

1. Create a server socket and a client socket.
2. Bind the server socket to a specific port number.
3. Create a buffer to hold incoming data.
4. Wait for incoming data from the client.
5. Receive the data from the client and store it in the buffer.
6. Convert the data to uppercase (or perform any other desired modifications).
7. Send the modified data back to the client.
8. Repeat steps 4-7 as necessary.
9. Close the sockets when finished.

Server:

```
import java.net.*;

public class UDPserver_urmila {
    public static void main(String args[]) throws Exception {
        try (DatagramSocket serverSocket = new DatagramSocket(1234)) {
            byte[] receiveData = new byte[1024];
            byte[] sendData = new byte[1024];
            while (true) {
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                serverSocket.receive(receivePacket);
                String sentence = new String(receivePacket.getData());
                InetAddress IPAddress = receivePacket.getAddress();
                int port = receivePacket.getPort();
                String capitalizedSentence = sentence.toUpperCase();
                sendData = capitalizedSentence.getBytes();
```

```
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
IPAddress, port);
                serverSocket.send(sendPacket);
            }
        }
    }
}
```

Client

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

public class UDPClient_urmila {
   public static void main(String args[]) throws Exception {
      BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
      DatagramSocket clientSocket = new DatagramSocket();
      InetAddress IPAddress = InetAddress.getByName("localhost");
      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];
      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress,
1234);
      clientSocket.send(sendPacket);
      DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
      clientSocket.receive(receivePacket);
      String modifiedSentence = new String(receivePacket.getData());
      System.out.println("FROM SERVER:" + modifiedSentence);
      clientSocket.close();
   }
}
```

Output:

```
PS C:\Users\HP\Downloads\21MIS1098> javac UDPserver_urmila.java
PS C:\Users\HP\Downloads\21MIS1098> java UDPserver_urmila.java
```

URMILA SV 21MIS1098

```
PS C:\Users\HP\Downloads\21MIS1098> javac UDPClient_urmila.java
PS C:\Users\HP\Downloads\21MIS1098> java UDPClient_urmila.java
hi this is Urmila
FROM SERVER:HI THIS IS URMILA
PS C:\Users\HP\Downloads\21MIS1098>
```

(b) Write a UDP-based server code to broadcast a message to the nodes in a LAN.

**Problem Definition :** We want to implement a UDP-based server that can broadcast a message to all nodes in a LAN using Java. The server should be able to send a message to the broadcast IP address of the LAN, and all nodes connected to the LAN should receive the message.

**Algorithm :**

1. Create a server socket.
2. Get the broadcast IP address of the LAN.
3. Create a buffer to hold the message to be sent.
4. Convert the message to bytes and store it in the buffer.
5. Create a datagram packet with the buffer, the length of the buffer, the broadcast IP address, and a specific port number.
6. Send the datagram packet using the server socket.
7. Close the server socket when finished.

Server

```
import java.net.*;

public class UDPServer2_urmila {
   public static void main(String args[]) throws Exception {
      DatagramSocket serverSocket = new DatagramSocket();
      InetAddress broadcastIP = InetAddress.getByName("255.255.255.255");
```

```
      byte[] sendData=new byte[1024];
      String sentence = "Hello World";
      sendData=sentence.getBytes();
      DatagramPacket sendPacket=new DatagramPacket(sendData, sendData.length, broadcastIP,
1234);
      serverSocket.send(sendPacket);
      System.out.println("Message sent to LAN broadcast address");
      serverSocket.close();
   }
}
```

Output:

```
PS C:\Users\HP\Downloads\21MIS1098> javac UDPServer2_urmila.java
PS C:\Users\HP\Downloads\21MIS1098> java UDPServer2_urmila.java
Message sent to LAN broadcast address
PS C:\Users\HP\Downloads\21MIS1098>
```

c) Write a code to implement border gateway protocol (BGP)

**Problem Definition :** We want to create a simple implementation of the Border Gateway Protocol (BGP) in Java. BGP is a protocol used by routers to exchange routing information and determine the best path for data to travel between networks. In this implementation, we will create a BGPProtocol class that simulates the BGP route exchange process between two routers. We will also create a Router class that represents a BGP router and can advertise routes to other routers and receive routes from them.

**Algorithm :**

1. Create a Router class with a constructor that takes a router ID and initializes anempty routing table.
2. Add methods to the Router class that allow it to advertise a route to other routersand receive routes from them.
3. Create a BGPProtocol class with an empty constructor that initializes an empty

router map.

4. Add a method to the BGPProtocol class that allows a router to be added to themap.

5. Add a method to the BGPProtocol class that simulates the BGP route exchange process between all routers in the map.

6. Within the BGPProtocol.exchangeRoutes() method, loop through each router inthe map and simulate the BGP message exchange and route update logic.

7. When a router receives a new route from another router, it should add it to itsrouting table.

8. When a router advertises a route to another router, it should send a BGPmessage containing the route information to the other router.

9. After all routers have exchanged routes, the BGPProtocol.exchangeRoutes() method should terminate.

10. Test the implementation by creating two Router objects and adding them to aBGPProtocol object.

11. Advertise some routes from one Router object to the other using theRouter.advertiseRoute() method.

12. Call the BGPProtocol.exchangeRoutes() method to simulate the BGP route exchange process between the two routers.

13. Use the Router.getNextHop() method to retrieve the next hop for a specificnetwork prefix from one of the routers' routing tables.

Server:

```
import java.util.HashMap;
import java.util.Map;

// Represents a BGP router
class Router {
    private String routerId;
    private Map<String, String> routingTable;

    public Router(String routerId) {
        this.routerId = routerId;
        this.routingTable = new HashMap<>();
    }
```

```
    public String getId() {
        return routerId;
    }

    public void advertiseRoute(String networkPrefix, String nextHop) {
        routingTable.put(networkPrefix, nextHop);
    }

    public String getNextHop(String networkPrefix) {
        return routingTable.get(networkPrefix);
    }
}

// Represents the BGP protocol handler
class BGPProtocol {
    private Map<String, Router> routers;

    public BGPProtocol() {
        this.routers = new HashMap<>();
    }

    public void addRouter(Router router) {
        routers.put(router.getId(), router);
    }

    public void exchangeRoutes() {
        for (Router router : routers.values()) {
            // Simulate BGP route exchange process
            // Implement BGP message exchange and route update logic here
        }
    }
}

public class BGPimplementation_urmila {
    public static void main(String[] args) {
        Router router1 = new Router("R1");
        Router router2 = new Router("R2");

        BGPProtocol bgp = new BGPProtocol();
        bgp.addRouter(router1);
        bgp.addRouter(router2);

        // Configure routing information
        router1.advertiseRoute("192.168.0.0/24", "192.168.1.1");
```

```
        router2.advertiseRoute("10.0.0.0/8", "10.1.1.1");

        // Exchange routes between routers
        bgp.exchangeRoutes();

        // Get next hop for a network prefix
        String nextHop = router1.getNextHop("10.0.0.0/8");
        System.out.println("Next hop for 10.0.0.0/8: " + nextHop);
    }
}
```

Output:

```
PS C:\Users\HP\Downloads\21MIS1098>  & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\HP\AppD
ata\Roaming\Code\User\workspaceStorage\b1be924e06459db78303064f817c7bf8\redhat.java\jdt_ws\21MIS1098_21cb980a\bin' 'BGPimplementation_urmila'
Next hop for 10.0.0.0/8: null
PS C:\Users\HP\Downloads\21MIS1098>
```

(b) Write a UDP server program that allows multiple clients to connect simultaneously. Each client should receive a unique identifier when they connect, and the server should be able to handle multiple client requests concurrently.

**Problem Definition :** We want to create a simple client-server application using the User Datagram Protocol (UDP) in Java. The server should listen for incoming client requests on a specific port and send a unique identifier to each client when they connect. The server should also process each client request in a separate thread and send a response back to the client. The client should connect to the server, send a request, and receive a response from the server.

**Algorithm for Server Code:**

1. Create a DatagramSocket on a specific port number.
2. Create an empty byte array to store incoming data.
3. Create an empty list to store client IP addresses and IDs.
4. Start an infinite loop to listen for incoming client requests.

5. Receive an incoming DatagramPacket from the client.
6. Check if the client IP address is already in the list of clients.
7. If the client is new, add their IP address to the list of clients and assign them a unique ID.
8. Send the unique ID back to the client in a DatagramPacket.
9. Print a message to the console indicating the new client has connected.
10. Create a new thread to handle the client request.
11. Start the new thread.
12. Go back to step 5.

**Algorithm for Client Code:**

1. Create a DatagramSocket for the client.
2. Convert the message to be sent into a byte array.
3. Create a DatagramPacket containing the byte array and the server IP address and port number.
4. Send the DatagramPacket to the server.
5. Receive an incoming DatagramPacket from the server.
6. Convert the data in the DatagramPacket to a string.
7. Print the string to the console as the response from the server.
8. Close the DatagramSocket.

Server:

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

public class UDPServer4_urmila {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        List<InetAddress> clients = new ArrayList<>();
        List<Integer> ids = new ArrayList<>();
        int id = 0;

        System.out.println("Server started on port 9876");

        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            InetAddress clientAddress = receivePacket.getAddress();

            if (!clients.contains(clientAddress)) {
                clients.add(clientAddress);
                ids.add(id);
                id++;
                System.out.println("New client connected: " + clientAddress);
            }

            int clientId = ids.get(clients.indexOf(clientAddress));
            String message = "Your unique identifier is " + clientId;
            byte[] sendData = message.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress, receivePacket.getPort());
            serverSocket.send(sendPacket);

            System.out.println("Received request from client " + clientId + ":
" + new String(receivePacket.getData()));

            Thread requestHandler = new Thread(new
RequestHandler(receivePacket, clientId));
            requestHandler.start();
        }
    }
}

class RequestHandler implements Runnable {
```

```
    private DatagramPacket packet;
    private int clientId;

    public RequestHandler(DatagramPacket packet, int clientId) {
        this.packet = packet;
        this.clientId = clientId;
    }

    public void run() {
        try {
            String requestData = new String(packet.getData()).trim();
            System.out.println("Processing request from client " + clientId +
": " + requestData);

            // Process request here

            String responseData = "Response from server";
            byte[] sendData = responseData.getBytes();
            DatagramSocket socket = new DatagramSocket();
            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, packet.getAddress(), packet.getPort());
            socket.send(sendPacket);
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Client:

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

public class UDPClient4_urmila {
    public static void main(String args[]) throws Exception {
        InetAddress serverAddress = InetAddress.getByName("localhost");
        DatagramSocket clientSocket = new DatagramSocket();

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        // Send request from client 1
        String message1 = "Hello from client 1";
        sendData = message1.getBytes();
        DatagramPacket sendPacket1 = new DatagramPacket(sendData,
sendData.length, serverAddress, 9876);
```

```
        clientSocket.send(sendPacket1);

        DatagramPacket receivePacket1 = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket1);
        String response1 = new String(receivePacket1.getData()).trim();
        System.out.println("Response from server to client 1: " + response1);

        // Send request from client 2
        String message2 = "Hello from client 2";
        sendData = message2.getBytes();
        DatagramPacket sendPacket2 = new DatagramPacket(sendData,
sendData.length, serverAddress, 9876);
        clientSocket.send(sendPacket2);

        DatagramPacket receivePacket2 = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket2);
        String response2 = new String(receivePacket2.getData()).trim();
        System.out.println("Response from server to client 2: " + response2);

        // Send request from client 3
        String message3 = "Hello from client 3";
        sendData = message3.getBytes();
        DatagramPacket sendPacket3 = new DatagramPacket(sendData,
sendData.length, serverAddress, 9876);
        clientSocket.send(sendPacket3);

        DatagramPacket receivePacket3 = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket3);
        String response3 = new String(receivePacket3.getData()).trim();
        System.out.println("Response from server to client 3: " + response3);

        clientSocket.close();
    }
}
```

Output:

```
PS C:\Users\HP\Downloads\21MIS1098> javac UDPServer4_urmila.java
PS C:\Users\HP\Downloads\21MIS1098> java UDPServer4_urmila.java
Server started on port 9876
New client connected: /127.0.0.1
Received request from client 0: Hello from client 1
Received request from client 0: Hello from client 2
Processing request from client 0: Hello from client 2
Received request from client 0: Hello from client 3
Processing request from client 0: Hello from client 3
Processing request from client 0: Hello from client 3
```

```
PS C:\Users\HP\Downloads\21MIS1098> javac UDPClient4_urmila.java
PS C:\Users\HP\Downloads\21MIS1098> java UDPClient4_urmila.java
Response from server to client 1: Your unique identifier is 0
Response from server to client 2: Your unique identifier is 0
Response from server to client 3: Your unique identifier is 0
PS C:\Users\HP\Downloads\21MIS1098>
```