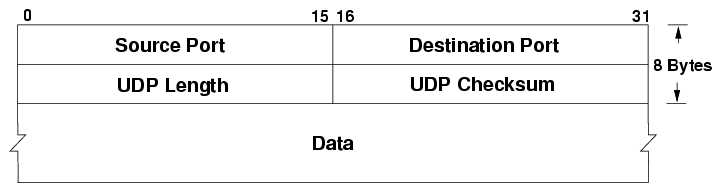
**EXPERIMENT NO. 7**

***Aim:*** To transfer File using User Datagram Protocol (UDP)

***Theory:***

* The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite.
* UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection.
* UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.
* UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level.
* Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.
* A number of UDP's attributes make it especially suited for certain applications.
* It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.
* It provides datagrams, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
* It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
* It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV
* The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
* Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol.



The UDP header consists of 4 fields, each of which is 2 bytes (16 bits).The use of the fields "Checksum" and "Source port" is optional in IPv4.In IPv6 only the source port is optional.

**Source port number**

This field identifies the sender's port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero. If the source host is the client, the port number is likely to be an ephemeral port number. If the source host is the server, the port number is likely to be a well-known port number.

**Destination port number**

This field identifies the receiver's port and is required. Similar to source port number, if the client is the destination host then the port number will likely be an ephemeral port number and if the destination host is the server then the port number will likely be a well-known port number.

**Length**

A field that specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes because that is the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes (65,535 − 8 byte UDP header − 20 byte IP header). In IPv6 Jumbograms it is possible to have UDP packets of size greater than 65,535 bytes. RFC 2675 specifies that the length field is set to zero if the length of the UDP header plus UDP data is greater than 65,535.

**Checksum**

The checksum field is used for error-checking of the header and data. If no checksum is generated by the transmitter, the field uses the value all-zeros. This field is not optional for IPv6.

***Program:***

**FileEvent.java**

import java.io.Serializable;

public class FileEvent implements Serializable {

public FileEvent() {

}

private static final long serialVersionUID = 1L;

private String destinationDirectory;

private String sourceDirectory;

private String filename;

private long fileSize;

private byte[] fileData;

private String status;

public String getDestinationDirectory() {

return destinationDirectory;

}

public void setDestinationDirectory(String destinationDirectory) {

this.destinationDirectory = destinationDirectory;

}

public String getSourceDirectory() {

return sourceDirectory;

}

public void setSourceDirectory(String sourceDirectory) {

this.sourceDirectory = sourceDirectory;

}

public String getFilename() {

return filename;

}

public void setFilename(String filename) {

this.filename = filename;

}

public long getFileSize() {

return fileSize;

}

public void setFileSize(long fileSize) {

this.fileSize = fileSize;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public byte[] getFileData() {

return fileData;

}

public void setFileData(byte[] fileData) {

this.fileData = fileData;

}

}

### **Server.java**

import java.io.\*;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

public class Server {

private DatagramSocket socket = null;

private FileEvent fileEvent = null;

public Server() {

}

public void createAndListenSocket() {

try {

socket = new DatagramSocket(9876);

byte[] incomingData = new byte[1024 \* 1000 \* 50];

while (true) {

DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length);

socket.receive(incomingPacket);

byte[] data = incomingPacket.getData();

ByteArrayInputStream in = new ByteArrayInputStream(data);

ObjectInputStream is = new ObjectInputStream(in);

fileEvent = (FileEvent) is.readObject();

if (fileEvent.getStatus().equalsIgnoreCase("Error")) {

System.out.println("Some issue happened while packing the data @ client side");

System.exit(0);

}

createAndWriteFile(); // writing the file to hard disk

InetAddress IPAddress = incomingPacket.getAddress();

int port = incomingPacket.getPort();

String reply = "Thank you for the message";

byte[] replyBytea = reply.getBytes();

DatagramPacket replyPacket =

new DatagramPacket(replyBytea, replyBytea.length, IPAddress, port);

socket.send(replyPacket);

Thread.sleep(3000);

System.exit(0);

}

} catch (SocketException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public void createAndWriteFile() {

String outputFile = fileEvent.getDestinationDirectory() + fileEvent.getFilename();

if (!new File(fileEvent.getDestinationDirectory()).exists()) {

new File(fileEvent.getDestinationDirectory()).mkdirs();

}

File dstFile = new File(outputFile);

FileOutputStream fileOutputStream = null;

try {

fileOutputStream = new FileOutputStream(dstFile);

fileOutputStream.write(fileEvent.getFileData());

fileOutputStream.flush();

fileOutputStream.close();

System.out.println("Output file : " + outputFile + " is successfully saved ");

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

Server server = new Server();

server.createAndListenSocket();

}

}

Client.java

import java.io.\*;

import java.net.\*;

public class Client {

private DatagramSocket socket = null;

private FileEvent event = null;

private String sourceFilePath = "E:/temp/images/photo.jpg";

private String destinationPath = "C:/tmp/downloads/udp/";

private String hostName = "localHost";

public Client() {

}

public void createConnection() {

try {

socket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName(hostName);

byte[] incomingData = new byte[1024];

event = getFileEvent();

ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

ObjectOutputStream os = new ObjectOutputStream(outputStream);

os.writeObject(event);

byte[] data = outputStream.toByteArray();

DatagramPacket sendPacket = new DatagramPacket(data, data.length, IPAddress, 9876);

socket.send(sendPacket);

System.out.println("File sent from client");

DatagramPacket incomingPacket = new DatagramPacket(incomingData, incomingData.length);

socket.receive(incomingPacket);

String response = new String(incomingPacket.getData());

System.out.println("Response from server:" + response);

Thread.sleep(2000);

System.exit(0);

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (SocketException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public FileEvent getFileEvent() {

FileEvent fileEvent = new FileEvent();

String fileName = sourceFilePath.substring(sourceFilePath.lastIndexOf("/") + 1, sourceFilePath.length());

String path = sourceFilePath.substring(0, sourceFilePath.lastIndexOf("/") + 1);

fileEvent.setDestinationDirectory(destinationPath);

fileEvent.setFilename(fileName);

fileEvent.setSourceDirectory(sourceFilePath);

File file = new File(sourceFilePath);

if (file.isFile()) {

try {

DataInputStream diStream = new DataInputStream(new FileInputStream(file));

long len = (int) file.length();

byte[] fileBytes = new byte[(int) len];

int read = 0;

int numRead = 0;

while (read < fileBytes.length && (numRead = diStream.read(fileBytes, read, fileBytes.length - read)) >= 0) {

read = read + numRead;

}

fileEvent.setFileSize(len);

fileEvent.setFileData(fileBytes);

fileEvent.setStatus("Success");

} catch (Exception e) {

e.printStackTrace();

fileEvent.setStatus("Error");

}

} else {

System.out.println("path specified is not pointing to a file");

fileEvent.setStatus("Error");

}

return fileEvent;

}

public static void main(String[] args) {

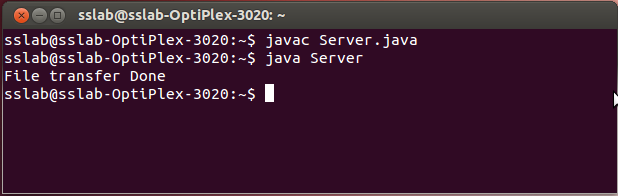
Client client = new Client();

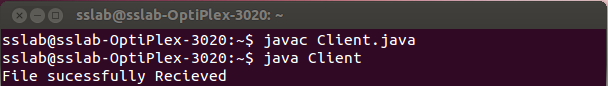
client.createConnection();

}

}

***Output:***





***Conclusion:***

Hence, we have studied that UDP (User Datagram Protocol) is an alternative communication protocol to TCP used primarily for establishing low-latency and loss tolerating connections between applications on the INTERNET. Both UDP and TCP run on the top of the INTERNET protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called Datagrams.