University of Dhaka

Department of Computer Science & Engineering

CSE-3111: Computer Networking Lab
Year: 3rd Semester: 1st

Lab Report No: 02

Lab Report Name: Implementing File Transfer using Socket Programming and HTTP GET/POST requests

Submission date: 25.09.2025

Submitted by:

Srabon Aich (Roll-15)
Abantika Paul (Roll-21)

Submitted to:

Dr. Shabbir Ahmed, Professor, Dept. of CSE, DU
Mr. Palash Roy, Lecturer, Dept. of CSE, DU

Contents

1 Introduction

File transfer is a fundamental network operation. Two common approaches are: direct socket programming using TCP/UDP, and higher-level HTTP-based transfer using GET and POST methods. The aim of this lab is to implement an HTTP-based file server and client (supporting GET for downloads and POST for uploads) and compare it with a raw socket-based approach.

2 Objectives

1. Implement a simple HTTP file server that serves files over GET and accepts uploads over POST.
2. Implement a client capable of downloading (GET) and uploading (POST) files to/from the server.
3. Analyze and compare file transfer using raw socket programming vs HTTP-based transfer (advantages, limitations).

3 Design Details

This section describes step-by-step how the system was designed and implemented. A flowchart summarizing the HTTP file transfer process is provided.

3.1 High-level components

- **HTTP File Server:** Listens on a port (e.g. 8080). Two HTTP endpoints:
  - `/download?filename=<name>` — handles GET, returns file
  - `/upload` — handles POST, reads request body and writes to server storage.
- **HTTP File Client:** Provides a simple interactive menu to:
  - Upload local file to `/upload` (POST).
  - Download a file by name from `/download?filename=...` (GET).
- **Storage:** Server-side file directory (e.g. `./files/server`) and client-side save directory (e.g. `./files/client`).

3.2 Detailed step-by-step process

1. **Server initialization**
   (a) Configure server port (default 8080).
   (b) Create file storage directory if not present.
   (c) Instantiate HTTP server and register the two contexts (`/download` and `/upload`).
   (d) Start a thread pool / executor for concurrent handling.

2. **Download (GET) flow**

   (a) Client issues HTTP GET to `/download?filename=example.txt`.

   (b) Server validates method is GET; if not, returns 405 Method Not Allowed.

   (c) Server checks for file existence:

   - If not exists, respond 404 Not Found with message.
   - If exists, set headers: `Content-Type: application/octet-stream` and `Content-Disposition: attachment; filename="example.txt"`.
   - Send HTTP 200 with the file bytes streamed in chunks.

   (d) Client receives response, reads stream and writes to a local file.

3. **Upload (POST) flow**

   (a) Client opens a connection to `/upload` and sends POST request with file bytes in the body.

   (b) Server validates method is POST; if not, responds 405.

   (c) Server reads the request body stream in chunks and writes to a new file (e.g., `upload_YYYYMMDD_HHMMSS` or original name if provided).

   (d) Server replies with 200 OK and a confirmation message including saved filename.

3.3 Flowchart (HTTP GET/POST file transfer)

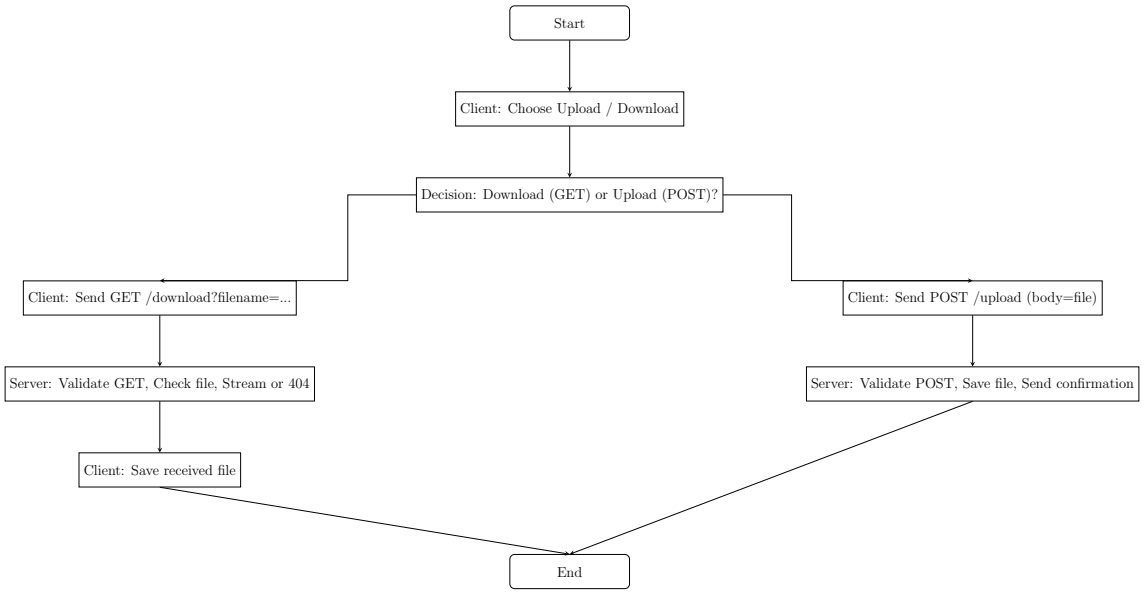

Figure 1: Flowchart for HTTP-based file transfer (GET download and POST upload).

4 Implementation

Below are the screenshots of server side and client side.

4.1 Server-side screenshots

```
package socketProgramming.httpServer;
import ...

public class Server {  new *
    public static final String FILES_PATH = "./files/server";  3 usages
    public static final int PORT = 8080;  2 usages

    public static void main(String[] args) throws IOException {  new *

        File dir = new File(FILES_PATH);
        if (!dir.exists()) dir.mkdirs();

        HttpServer server = HttpServer.create(new InetSocketAddress(PORT),  backlog: 0);
        System.out.println("HTTP File Server started at http://localhost:" + PORT);

        server.createContext( path: "/download", new DownloadHandler());

        server.createContext( path: "/upload", new UploadHandler());

        server.setExecutor(Executors.newFixedThreadPool( nThreads: 10));
        server.start();
    }

    static class DownloadHandler implements HttpHandler {  1 usage  new *
        @Override  new *
        public void handle(HttpExchange exchange) throws IOException {
            if (!exchange.getRequestMethod().equalsIgnoreCase( anotherString: "GET")) {
                exchange.sendResponseHeaders( rCode: 405,  responseLength: -1);
                return;
            }

            URI requestURI = exchange.getRequestURI();
            String query = requestURI.getQuery();
            String filename = null;
```

Figure 2: Server start log

```
public class Server {  new *
    static class DownloadHandler implements HttpHandler {  1 usage  new *
        public void handle(HttpExchange exchange) throws IOException {
            }

            File file = new File(FILES_PATH, filename);
            if (!file.exists()) {
                String response = "File Not Found";
                exchange.sendResponseHeaders( rCode: 404, response.length());
                exchange.getResponseBody().write(response.getBytes());
                exchange.close();
                return;
            }

            exchange.getResponseHeaders().add("Content-Type", "application/octet-stream
            exchange.getResponseHeaders().add("Content-Disposition", "attachment; filen

            exchange.sendResponseHeaders( rCode: 200, file.length());

            try (OutputStream os = exchange.getResponseBody();
                 FileInputStream fis = new FileInputStream(file)) {
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = fis.read(buffer)) != -1) {
                    os.write(buffer,  off: 0, bytesRead);
                }
            }
            exchange.close();
            System.out.println("File sent: " + file.getName());
        }
    }

    static class UploadHandler implements HttpHandler {  1 usage  new *
        @Override  new *
```

Figure 3: Download action log

```
    static class UploadHandler implements HttpHandler {  1 usage   new *
        @Override  new *
        public void handle(HttpExchange exchange) throws IOException {
            if (!exchange.getRequestMethod().equalsIgnoreCase( anotherString: "POST")) {
                exchange.sendResponseHeaders( rCode: 405,  responseLength: -1);
                return;
            }

            String timeStamp = new SimpleDateFormat( pattern: "yyyyMMdd_HHmmss").format(new Date());
            File outFile = new File(FILES_PATH,  child: "upload_" + timeStamp);

            try (InputStream is = exchange.getRequestBody();
                 FileOutputStream fos = new FileOutputStream(outFile)) {
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = is.read(buffer)) != -1) {
                    fos.write(buffer,  off: 0, bytesRead);
                }
            }

            String response = "File uploaded successfully as " + outFile.getName();
            exchange.sendResponseHeaders( rCode: 200, response.length());
            exchange.getResponseBody().write(response.getBytes());
            exchange.close();

            System.out.println("File uploaded: " + outFile.getName());
        }
    }
}
```

Figure 4: Upload action log

4.2 Client-side screenshots

```
public class Client {  new *
    private static final String SERVER_URL = "http://localhost:8080";  2

    public static void main(String[] args) {  new *
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.println("\n1. Upload File");
            System.out.println("2. Download File");
            System.out.println("3. Exit");
            System.out.print("Choice: ");
            int choice = sc.nextInt();
            sc.nextLine();

            try {
                if (choice == 1) {
                    System.out.print("Enter file path to upload: ");
                    String filePath = sc.nextLine();
                    uploadFile(filePath);
                } else if (choice == 2) {
                    System.out.print("Enter filename to download: ");
                    String filename = sc.nextLine();
                    downloadFile(filename);
                } else {
                    System.out.println("Exiting...");
                    break;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        sc.close();
    }
```

Figure 5: Client start log

```java
private static void uploadFile(String filePath) throws IOException {  1 usage   new *
    File file = new File(filePath);
    if (!file.exists()) {
        System.out.println("File not found!");
        return;
    }

    URL url = new URL( spec: SERVER_URL + "/upload");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");

    try (FileInputStream fis = new FileInputStream(file);
         OutputStream os = conn.getOutputStream()) {
        byte[] buffer = new byte[4096];
        int bytesRead;
        while ((bytesRead = fis.read(buffer)) != -1) {
            os.write(buffer,  off: 0, bytesRead);
        }
    }

    int responseCode = conn.getResponseCode();
    if (responseCode == 200) {
        try (BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()))) {
            System.out.println("Server: " + br.readLine());
        }
    } else {
        System.out.println("Upload failed, code: " + responseCode);
    }
}
```

Figure 6: Upload action log

```java
    private static void downloadFile(String filename) throws IOException {  1 usage   new *
        URL url = new URL( spec: SERVER_URL + "/download?filename=" + filename);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        int responseCode = conn.getResponseCode();
        if (responseCode == 200) {
            File saveDir = new File( pathname: "./files/client");
            if (!saveDir.exists()) saveDir.mkdirs();
            File outFile = new File(saveDir, filename);

            try (InputStream is = conn.getInputStream();
                 FileOutputStream fos = new FileOutputStream(outFile)) {
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = is.read(buffer)) != -1) {
                    fos.write(buffer,  off: 0, bytesRead);
                }
            }
            System.out.println("File downloaded successfully: " + outFile.getAbsolutePath());
        } else if (responseCode == 404) {
            System.out.println("File not found on server.");
        } else {
            System.out.println("Download failed, code: " + responseCode);
        }
    }
}
```

Figure 7: Download action log

5 Result Analysis

5.1 Observed outputs

- **Successful download:** The client issued GET to `/download?filename=test.txt` and received response code 200. The file was saved to `./files/client/test.txt`.

```
1. Upload File
2. Download File
3. Exit
Choice: 2
Enter filename to download: destination-Companion-main.zip
File downloaded successfully: /home/srabon/IntelliJProjects/datacomLab/./files/client/destination-Companion-main.zip

1. Upload File
2. Download File
3. Exit
Choice: 1
Enter file path to upload: /home/srabon/IntelliJProjects/datacomLab/files/client/kidshirt3.jpeg
Server: File uploaded successfully as upload_20250925_013635

1. Upload File
2. Download File
3. Exit
Choice:
```

- **File not found:** When requesting a non-existent filename the server returned 404 Not Found and the client displayed `"File not found on server"`.

```
2. Download File
3. Exit
Choice: 2
Enter filename to download: haaak
File not found on server.

1. Upload File
```

- **Successful upload:** The client uploaded a file using POST; server wrote the file (e.g. `upload_20250924_153000`) and returned confirmation.

```
/usr/lib/jvm/jdk-22-oracle-x64/bin/java -javaagent:/snap/intellij-idea-com
HTTP File Server started at http://localhost:8080
File sent: destination-Companion-main.zip
File uploaded: upload_20250925_013635
```

6 Discussion

6.1 Comparison: Socket Programming vs HTTP-based Transfer

- **Low-level Socket Programming**
  - **Pros:** Full control of protocol, minimal overhead, suitable for custom protocols and optimization.
  - **Cons:** Must implement framing, error handling, and concurrency manually; interoperability is lower; rebuilding features (headers, status codes) is time consuming.
- **HTTP (GET/POST) based Transfer**

- **Pros:** Uses standardized protocol; easier to interoperate with existing clients/tools (browsers, curl); built-in semantics (status codes, headers, content disposition); easier to implement using built-in libraries (e.g., `HttpServer`, `HttpURLConnection`); easier debugging using existing HTTP tooling.
- **Cons:** Some overhead (HTTP headers); less control on low-level performance tuning unless using advanced servers.

6.2 What I learned

- How to build a minimal HTTP server and client to transfer files using GET and POST.
- Importance of content headers (e.g., Content-Type, Content-Disposition) for correct file handling on the client.
- Concurrency management on the server side with a thread pool.

6.3 Difficulties faced

- Handling partial reads/writes and making sure to stream file bytes in chunks to avoid memory pressure for large files.
- Ensuring correct use of HTTP response codes and closing streams to avoid resource leaks.
- Choosing naming policy for uploaded files (unique names vs preserve original filename).

7 Conclusion

This lab demonstrates a practical comparison between raw socket file transfer and HTTP-based file transfer. HTTP simplifies interoperability and development effort and integrates well with web tooling, while low-level sockets can provide finer-grain performance control when required.