Department of Computer Science and Engineering

University of Dhaka

Implementation of File Transfer Using Socket Programming and HTTP GET/POST Requests

---

CSE 3111: Computer Networking Lab

3rd Year 1st Semester 2025

Submitted By:

Sara Faria Sundra(58)
Asuma Bhuiyan(85)

Submitted To:

Dr. Sabbir Ahmed
Dr. Ismat Rahman
Mr. Palash Roy (PR)

Submitted: 24 September 2025

Introduction

File transfer is one of the most fundamental aspects of computer networking and plays a crucial role in data communication between systems. There are multiple ways to implement file transfer, among which socket programming and HTTP requests (GET/POST) are widely used. Socket programming enables low-level control over data transfer using the TCP/IP stack, making it useful for custom applications. On the other hand, HTTP provides a higher-level abstraction, allowing file transfer in a standardized and widely accepted way across web applications.

This lab experiment focuses on implementing file transfer using socket programming and HTTP GET/POST methods. By doing so, we aim to understand the underlying mechanisms, compare performance, and analyze the advantages and challenges of both approaches.

Objectives

- To implement file transfer using HTTP GET and POST methods.
- To analyze and compare file transfer mechanisms between socket programming and HTTP.
- To gain practical knowledge of client-server communication in networking.

Design Details

The implementation consists of two main approaches:

1. **Socket Programming:** A custom server and client communicate directly using TCP sockets to transfer files.
2. **HTTP GET/POST:** A server provides files through HTTP requests, where the client retrieves or uploads files using GET/POST methods.

Flow of File Transfer Using HTTP GET/POST:

1. The client establishes a connection with the server via HTTP.
2. For downloading: the client sends a GET request to the server.
3. For uploading: the client sends a POST request along with the file.
4. The server processes the request and responds accordingly.

Implementation

Below is the updated Java code for HTTP file transfer (GET and POST with file listing and error handling):

Server Code

```
import com.sun.net.httpserver.*;
import java.io.*;
import java.net.InetSocketAddress;
import java.nio.file.Files;

public class HttpFileServer {

    public static void main(String[] args) throws IOException {
        int port = 8000;
        HttpServer server = HttpServer.create(new
            ↪ InetSocketAddress(port), 0);


        server.createContext("/list", new HttpHandler() {
            @Override
            public void handle(HttpExchange exchange) throws
                ↪ IOException {
                if ("GET".equals(exchange.getRequestMethod())) {
                    File dir = new File("server_files");
                    if (!dir.exists()) dir.mkdir();

                    File[] files = dir.listFiles();
                    StringBuilder response = new StringBuilder();

                    if (files != null && files.length > 0) {
                        for (File f : files) {
                            if (f.isFile()) {
```

```
                        response.append(f.getName()).
                            ↪ append("\n");
                    }
                }
                System.out.println("␣Sent␣file␣list␣to␣
                    ↪ client");
            } else {
                response.append("No␣files␣found.");
                System.out.println("No␣files␣found␣in␣
                    ↪ server_files");
            }

            byte[] respBytes = response.toString().
                ↪ getBytes();
            exchange.sendResponseHeaders(200, respBytes.
                ↪ length);
            OutputStream os = exchange.getResponseBody();
            os.write(respBytes);
            os.close();
        }
    }
});


server.createContext("/download", exchange -> {
    if ("GET".equals(exchange.getRequestMethod())) {
        String query = exchange.getRequestURI().getQuery
            ↪ ();
        String fileName = query != null && query.
            ↪ startsWith("file=") ? query.substring(5) :
            ↪ "";
        File file = new File("server_files/" + fileName);

        if (file.exists()) {
            exchange.sendResponseHeaders(200, file.length
                ↪ ());
            OutputStream os = exchange.getResponseBody();
            Files.copy(file.toPath(), os);
            os.close();
            System.out.println("File␣sent:␣" + fileName);
        } else {
```

```
                String response = "File not found: " +
                    ↪ fileName;
                exchange.sendResponseHeaders(404, response.
                    ↪ length());
                OutputStream os = exchange.getResponseBody();
                os.write(response.getBytes());
                os.close();
                System.out.println("File requested but not
                    ↪ found: " + fileName);
            }
        }
    });


    server.createContext("/upload", exchange -> {
        if ("POST".equals(exchange.getRequestMethod())) {
            String fileName = exchange.getRequestHeaders().
                ↪ getFirst("File-Name");
            if (fileName == null) fileName = "uploaded_file";

            File uploadDir = new File("server_files");
            if (!uploadDir.exists()) uploadDir.mkdir();

            File file = new File(uploadDir, fileName);
            InputStream is = exchange.getRequestBody();
            Files.copy(is, file.toPath(), java.nio.file.
                ↪ StandardCopyOption.REPLACE_EXISTING);
            is.close();

            String response = "File uploaded: " + fileName;
            exchange.sendResponseHeaders(200, response.length
                ↪ ());
            OutputStream os = exchange.getResponseBody();
            os.write(response.getBytes());
            os.close();

            System.out.println(" File uploaded: " + fileName)
                ↪ ;
        }
    });
```

```
        server.setExecutor(null);
        server.start();
        System.out.println("␣HTTP␣File␣Server␣started␣at␣port␣" +
            ↪ port);
    }
}
```

Client Code

```
import java.io.*;
import java.net.*;
import java.nio.file.Files;
import java.util.Scanner;

public class HttpFileClient {

    private static final String SERVER_URL = "http://localhost
        ↪ :8000";

    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.print("Enter␣command␣(list,␣download␣<
                ↪ filename>,␣upload␣<filename>,␣quit):␣");
            String command = scanner.nextLine();

            if (command.equalsIgnoreCase("quit")) {
                System.out.println("Exiting␣client...");
                break;
            } else if (command.equalsIgnoreCase("list")) {
                listFiles();
            } else if (command.startsWith("download␣")) {
                String fileName = command.substring(9).trim();
                downloadFile(fileName);
            } else if (command.startsWith("upload␣")) {
                String fileName = command.substring(7).trim();
                uploadFile(fileName);
            } else {
                System.out.println("Invalid␣command.");
```

```
            }
        }
        scanner.close();
    }

    private static void listFiles() throws IOException {
        URL url = new URL(SERVER_URL + "/list");
        HttpURLConnection connection = (HttpURLConnection) url.
            ↪ openConnection();
        connection.setRequestMethod("GET");

        BufferedReader reader = new BufferedReader(new
            ↪ InputStreamReader(connection.getInputStream()));
        String line;
        System.out.println("Files␣on␣server:");
        while ((line = reader.readLine()) != null) {
            System.out.println("␣-␣" + line);
        }
        reader.close();
    }

    private static void downloadFile(String fileName) throws
        ↪ IOException {
        URL url = new URL(SERVER_URL + "/download?file=" +
            ↪ fileName);
        HttpURLConnection connection = (HttpURLConnection) url.
            ↪ openConnection();
        connection.setRequestMethod("GET");

        int responseCode = connection.getResponseCode();
        if (responseCode == 200) {
            File downloadDir = new File("client_files");
            if (!downloadDir.exists()) downloadDir.mkdir();

            InputStream is = connection.getInputStream();
            FileOutputStream fos = new FileOutputStream(new File(
                ↪ downloadDir, fileName));

            byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = is.read(buffer)) != -1) {
```

```
                fos.write(buffer, 0, bytesRead);
            }

            fos.close();
            is.close();
            System.out.println("File downloaded to client_files/"
               ↪ + fileName);
        } else {
            System.out.println("File not found on server.");
        }
    }

    private static void uploadFile(String fileName) throws
       ↪ IOException {
        File file = new File(fileName);
        if (!file.exists()) {
            System.out.println("File does not exist locally.");
            return;
        }

        URL url = new URL(SERVER_URL + "/upload");
        HttpURLConnection connection = (HttpURLConnection) url.
           ↪ openConnection();
        connection.setDoOutput(true);
        connection.setRequestMethod("POST");
        connection.setRequestProperty("File-Name", file.getName()
           ↪ );

        OutputStream os = connection.getOutputStream();
        Files.copy(file.toPath(), os);
        os.close();

        BufferedReader reader = new BufferedReader(new
           ↪ InputStreamReader(connection.getInputStream()));
        String response = reader.readLine();
        reader.close();
        System.out.println(" " + response);
    }
}
```

Result Analysis

Server Output

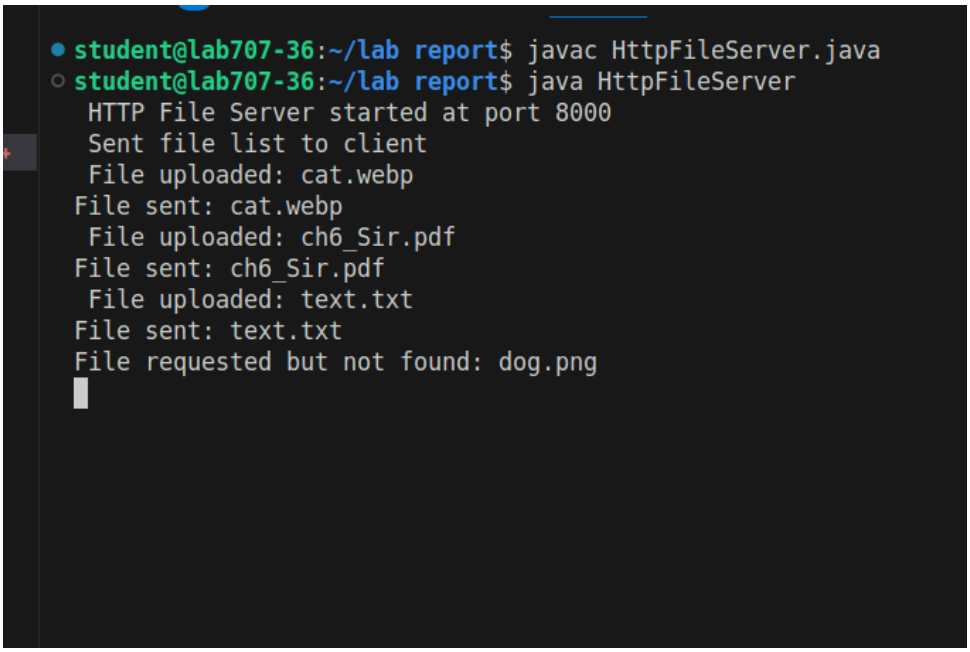

Figure 1: Server output showing file listing, upload, and download requests

Client Output

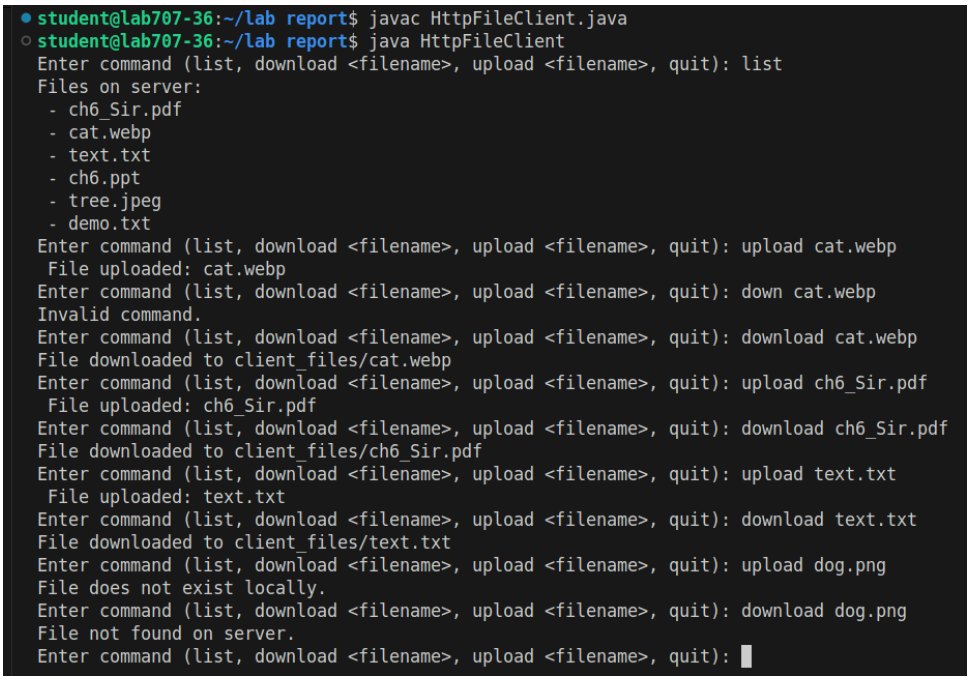

Figure 2: Client output after listing, downloading, and uploading files

Discussion

From the experiment, it is evident that both socket programming and HTTP methods enable successful file transfer, but with notable differences. Socket programming provides greater control and efficiency, as it communicates directly at the transport layer without extra overhead. However, it requires more complex implementation and lacks standardized mechanisms. HTTP GET/POST, on the other hand, offers simplicity, interoperability, and compatibility with web applications, but comes with additional protocol overhead.

In real-world scenarios, HTTP is more widely used due to its universality, ease of deployment, and support for various clients. This experiment helped us understand not only how file transfers work but also how protocol selection impacts performance, scalability, and complexity.

Conclusion

This lab successfully demonstrated file transfer using both socket programming and HTTP GET/POST methods. While socket programming provides fine-grained control over the communication process, HTTP offers a more user-friendly and standardized approach. We observed the practical differences between the two methods, gaining hands-on knowledge about client-server interaction, file handling, and protocol behavior. Overall, this experiment enhanced our understanding of networking fundamentals and their practical implementation in real-world systems.