Implementation of Multiplexing and Demultiplexing

Using Statistical TDM

CSE 2213 : Data and Telecommunication Lab

Batch : 29 / 2nd Year 2nd Semester 2024

Date

23 May, 2025

Submitted by

Atiya Fahmida (Roll-49)
Jubair Ahammad Akter (Roll-59)

Course Instructors

Dr. Md. Mustafizur Rahman (Professor)
Mr. Palash Roy (Lecturer)

Department of Computer Science and Engineering
University of Dhaka

Table of Contents

1 Introduction

1.1 What is MUX & DEMUX ?

Multiplexing (MUX) is the process of combining multiple signals into one for transmission over a shared medium. Meanwhile **Demultiplexing (DEMUX)** is the process of separating the combined signal back into the original individual signals.

1.2 The Necessity of Multiplexing and Demultiplexing

- Efficient utilization of bandwidth by allowing multiple data streams to share a single channel.
- Reduces the cost and complexity of the communication infrastructure.
- Minimizes the number of physical communication lines needed.
- Supports scalability for growing numbers of users or devices.
- Enables simultaneous transmission of multiple signals like audio, video, and data.
- DEMUX ensures accurate delivery of each stream to the appropriate destination.
- Real-world examples include :
  - Multiple phone calls on a single fiber optic cable.
  - Multiple video/audio streams over a single internet connection.

1.3 Types of Multiplexing

Different multiplexing techniques are used depending on the application and transmission medium :

- **Time Division Multiplexing (TDM) :** Assigns fixed time slots to each signal in a round-robin fashion. If a sender has no data to send, the slot is wasted (synchronous TDM) or reassigned (Statistical TDM).
- **Frequency Division Multiplexing (FDM) :** Each signal is transmitted over a different frequency band within the same channel. Common in radio, television broadcasting, and cable TV.
- **Statistical Time Division Multiplexing (Stat-TDM) :** Dynamically allocates time slots based on demand, improving channel utilization.
- **Wavelength Division Multiplexing (WDM) :** Used in fiber optics where different data streams are sent over different wavelengths of light.

1.4 Introduction to TDM Techniques

- **Synchronous TDM (Sync-TDM)** assigns fixed time slots to each sender in a round-robin fashion, regardless of whether the sender has data to send or not. This can lead to bandwidth being wasted if some sources are idle during their assigned time slots.

  Example : Suppose three devices (A, B, and C) are connected via Sync-TDM. Each device gets one time slot in every frame. If device C has no data, its time slot remains empty, wasting bandwidth.

- **Statistical TDM (Stat-TDM)** dynamically allocates time slots only to those sources that have data to transmit. This method is more efficient in scenarios with bursty or irregular traffic since time slots are not reserved unnecessarily.

  Example : In the same three-device setup (A, B, and C), if only A and B have data to send, the TDM frame in Stat-TDM will include slots only for A and B, omitting C, thereby improving channel utilization. Stat-TDM is widely used in modern communication systems, such as packet-switched networks, where bandwidth efficiency and flexibility are critical.

2 Objectives

- Implement Multiplexing (MUX) and Demultiplexing (DEMUX) in data communication and understand the basic concept.
- To simulate statistical TDM using Java.
- To analyze and compare different types of multiplexing techniques, especially Sync-TDM and Stat-TDM.
- To demonstrate how demultiplexing helps identify and deliver data to the correct application.

3 Algorithms/Pseudocode

3.1 Client Side (Multiplexing)

Code 1 : Open N input files. java

```
//I can give in the terminal command as many textfiles as I want, just
↪  have to store those files in the location folder.
// java L4_49_59_Client_Stat_TDM <IP> <Port> <InputFile1> <InputFile2> ...
↪  <InputFileN>
// java L4_49_59_Client_Stat_TDM 127.0.0.1 6000 Input1.txt Input2.txt
↪  Input3txt Input4.txt (For me)
```

Code 2 : Initialize an empty frame. java

```
byte[] frame = frameStream.toByteArray();
```

Code 3 : For each source : If it has data available :

- Read 1 byte.
- Create a data unit : Stream ID, Data.
- Add to frame.

java

```
if (!allEOF) {
    byte[] frame = frameStream.toByteArray();
    dos.writeShort(frame.length);
    dos.write(frame);
    dos.flush();

    System.out.println("Frame #" + frameNumber);
    for (int i = 0; i < frame.length; i += 2) {
        int streamID = frame[i] & 0xFF;
        char dataChar = (char) frame[i + 1];
        System.out.println(" Stream ID: " + streamID + " Data: " +
        ↪  dataChar);
    }
    System.out.println();
    frameNumber++;
}
```

Code 4 : Store and send frame to server. java

```
Socket socket = new Socket(serverIP, port);
DataOutputStream dos = new DataOutputStream (socket.getOutputStream());
...
for (FileInputStream fis : inputs) fis.close();
dos.close();
socket.close();
System.out.println("All data sent.");
```

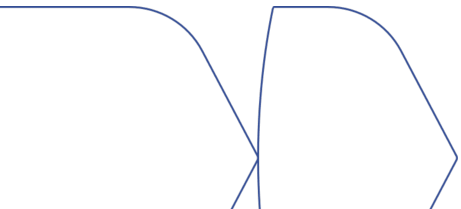
3.2 Server Side (Demultiplexing)

Code 5 : Read each frame line-by-line and parse Stream ID and Data. java

```
byte[] frame = new byte[frameLength];
dis.readFully(frame);
System.out.println("Frame #" + frameNumber);
for (int i = 0; i < frameLength; i += 2) {
    int streamID = frame[i] & 0xFF;
    byte data = frame[i + 1];
    char dataChar = (char) data;
    System.out.println(" Stream ID: " + streamID + " Data: " + dataChar);
    if (!outputFiles.containsKey(streamID)) {
        String outFileName = "Output" + streamID + ".txt";
        outputFiles.put(streamID, new FileOutputStream(outFileName));
        System.out.println("Created output file: " + outFileName);
    }
    outputFiles.get(streamID).write(data);
}
```

Code 6 : Write data to the corresponding output file using Stream ID. java

```
if (!outputFiles.containsKey(streamID)) {
    String outFileName = "Output" + streamID +".txt";
    outputFiles.put(streamID, new FileOutputStream(outFileName));
    System.out.println("Created output file: " + outFileName);
}
```



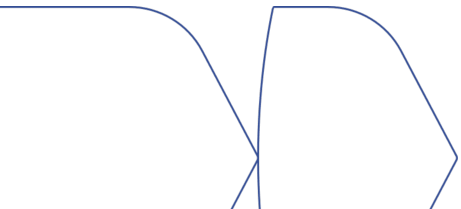
4 Implementation

4.1 Client Side (Java)

Code 7 : Stat-TDM Client - Multiplexing java

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

public class L4_49_59_Client_Stat_TDM {
    public static void main(String[] args) {
        if (args.length < 3) {
            System.out.println("Usage: java L4_49_59_Client_Stat_TDM <IP>
            ↪  <Port> <InputFile1> [<InputFile2> ... <InputFileN>]");
            System.exit(1);
        }

        String serverIP = args[0];
        int port = Integer.parseInt(args[1]);
        int numFiles = args.length - 2;

        List<FileInputStream> inputs = new ArrayList<>();

        try {
            // Print each input file content before sending
            for (int i = 2; i < args.length; i++) {
                String filename = args[i];
                String content = readFileAsString(filename);
                System.out.println(filename + ": " + content);
                inputs.add(new FileInputStream(filename));
            }

            Socket socket = new Socket(serverIP, port);
            DataOutputStream dos = new
            ↪  DataOutputStream(socket.getOutputStream());

            boolean[] eofFlags = new boolean[numFiles];
            boolean allEOF = false;

            int frameNumber = 1;
```

```
            while (!allEOF) {
                allEOF = true;
                ByteArrayOutputStream frameStream = new
                ↪  ByteArrayOutputStream();

                for (int i = 0; i < numFiles; i++) {
                    if (!eofFlags[i]) {
                        int data = inputs.get(i).read();
                        if (data == -1) {
                            eofFlags[i] = true;
                        } else {
                            allEOF = false;
                            frameStream.write(i + 1);
                            frameStream.write(data);
                        }
                    }
                }

                if (!allEOF) {
                    byte[] frame = frameStream.toByteArray();
                    dos.writeShort(frame.length);
                    dos.write(frame);
                    dos.flush();

                    System.out.println("Frame #" + frameNumber);
                    for (int i = 0; i < frame.length; i += 2) {
                        int streamID = frame[i] & 0xFF;
                        char dataChar = (char) frame[i + 1];
                        System.out.println(" Stream ID: " + streamID + "
                        ↪  Data: " + dataChar);
                    }
                    System.out.println();
                    frameNumber++;
                }
            }

            for (FileInputStream fis : inputs) fis.close();
            dos.close();
            socket.close();

            System.out.println("All data sent.");

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



```
        }
    }

    // Utility method to read entire file content as String
    private static String readFileAsString(String filename) throws
    ↪ IOException {
        StringBuilder sb = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new
        ↪ FileReader(filename))) {
            int ch;
            while ((ch = br.read()) != -1) {
                sb.append((char) ch);
            }
        }
        return sb.toString();
    }
}
```

4.2 Server Side (Java)

Code 8 : Stat-TDM Server - Demultiplexing java

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

public class L4_49_59_Server_Stat_TDM {
    public static void main(String[] args) {
        int port = 6000;

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server listening on port " + port);

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected.");

            DataInputStream dis = new
            ↪ DataInputStream(clientSocket.getInputStream());

            Map<Integer, FileOutputStream> outputFiles = new HashMap<>();
            boolean running = true;
```

```
int frameNumber = 1;

while (running) {
    try {
        int frameLength = dis.readUnsignedShort();
        if (frameLength <= 0) break;

        byte[] frame = new byte[frameLength];
        dis.readFully(frame);

        System.out.println("Frame #" + frameNumber);
        for (int i = 0; i < frameLength; i += 2) {
            int streamID = frame[i] & 0xFF;
            byte data = frame[i + 1];
            char dataChar = (char) data;

            System.out.println(" Stream ID: " + streamID + "
            ↪  Data: " + dataChar);

            if (!outputFiles.containsKey(streamID)) {
                String outFileName = "Output" + streamID +
                ↪  ".txt";
                outputFiles.put(streamID, new
                ↪  FileOutputStream(outFileName));
                System.out.println("Created output file: " +
                ↪  outFileName);
            }

            outputFiles.get(streamID).write(data);
        }
        System.out.println();
        frameNumber++;
    } catch (EOFException eof) {
        running = false;
    }
}

// Close all output streams
for (FileOutputStream fos : outputFiles.values()) {
    fos.close();
}

dis.close();
clientSocket.close();
```

```
            // After closing, print contents of each output file
            for (int streamID : outputFiles.keySet()) {
                String outFileName = "Output" + streamID + ".txt";
                String content = readFileAsString(outFileName);
                System.out.println(outFileName + ": " + content);
            }

            System.out.println("Server terminated.");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Utility method to read entire file content as String
    private static String readFileAsString(String filename) throws
    ↪ IOException {
        StringBuilder sb = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new
        ↪ FileReader(filename))) {
            int ch;
            while ((ch = br.read()) != -1) {
                sb.append((char) ch);
            }
        }
        return sb.toString();
    }
}
```

5 Result Analysis

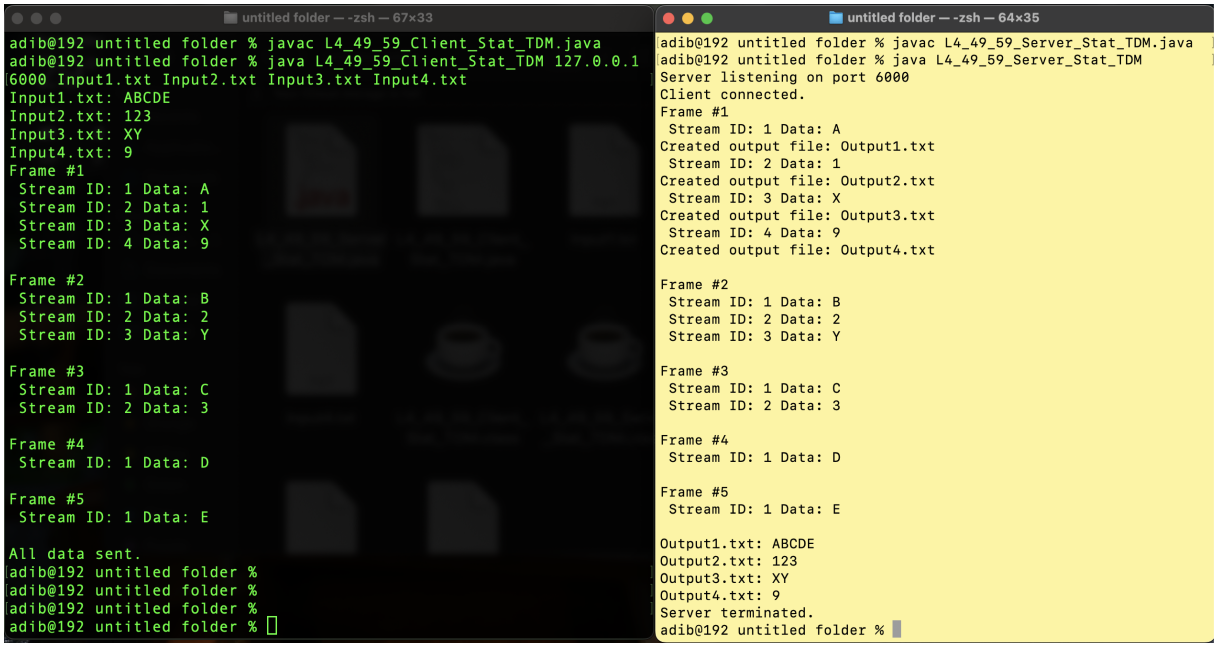


FIGURE 1 – Output of Client and Server on terminal

- **Frame Output (frame.txt) :**

Frame	Stream ID	Data
1	1	A
	2	1
	3	X
	4	9
2	1	B
	2	2
	3	Y
3	1	C
	2	3
4	1	D
5	1	E

Input Files :	Output Files :
Input1.txt : ABCDE	Output1.txt : ABCDE
Input2.txt : 123	Output2.txt : 123
Input3.txt : XY	Output3.txt : XY
Input4.txt : 9	Output4.txt : 9

6 Discussion

6.1 Sync-TDM vs Stat-TDM

- **Sync-TDM :** Fixed time slots for each stream regardless of activity. Waste of bandwidth if some sources are idle.
- **Stat-TDM :** Dynamically allocates time slots to active sources. More efficient in real-time applications.

6.2 Implementation Comparison

- Stat-TDM requires parsing stream IDs and managing variable frame size.
- Sync-TDM would require synchronization even for idle sources, making code more rigid.

7 Learning and Difficulties

- Learned dynamic handling of data streams using Java.
- Understood the efficiency advantage of statistical-TDM.
- Faced issues with file read/write buffering which were resolved with try-catch and proper closing of resources.

8 Conclusion

This experiment successfully demonstrated the implementation of multiplexing and demultiplexing using statistical TDM. The dynamic allocation of time slots based on data availability improved efficiency, especially in scenarios with sporadic data generation across sources. This exercise reinforced both theoretical understanding and practical skills in file handling and stream management in Java.