University of Dhaka

Department of Computer Science and Engineering

CSE 3111 – Computer Networking Lab
Collaborative Music Streamer

Lab Group A

Nafiul Alim Adeeb (23)
Sharmila Surovi (39)

Submitted To:

Dr. Ismat Rahman
Associate Professor, Dept. of CSE, University of Dhaka

Palash Roy
Lecturer, Dept. of CSE, University of Dhaka

Submission Date: 26/06/2025

Contents

1 PROJECT OVERVIEW **1**

2 PROBLEM STATEMENT & MOTIVATION **1**
2.1 Technical Challenges 1
2.2 Application Domain Motivation 1

3 SYSTEM OBJECTIVES **1**
3.1 Primary Objectives 1
3.2 Performance Goals 2

4 SYSTEM ARCHITECTURE **2**
4.1 High-Level Architecture 2
4.2 Component Specifications 3
4.2.1 Client Components 3
4.2.2 Server Components 4

5 PROTOCOL DESIGN **4**
5.1 Custom UDP Streaming Protocol 4
5.2 Control Protocol (Using JavaSockets) 4

6 IMPLEMENTATION WORKFLOW **5**

7 TECHNOLOGY STACK **5**
7.1 Backend Technologies 5
7.2 Frontend Technologies 5

8 CONCLUSION **5**

1 PROJECT OVERVIEW

Project Title: Collaborative Music Streamer

Abstract: This project presents the design and implementation of a real-time collaborative music streaming platform that enables multiple users to participate in synchronized audio playback sessions. The system demonstrates practical networking protocol implementations by utilizing TCP for reliable control plane communications and a custom UDP-based protocol for efficient media streaming. Users can collaboratively build playlists by submitting YouTube URLs, which are processed server-side and broadcast to all connected clients with timestamp-based synchronization.

2 PROBLEM STATEMENT & MOTIVATION

The development of this collaborative music streaming platform addresses several technical and social challenges:

2.1 Technical Challenges

- **Protocol Selection and Optimization:** Real-time media streaming requires careful consideration of network protocol characteristics. This project explores the trade-offs between TCP's reliability and latency versus UDP's low overhead and unreliability, necessitating a custom protocol design for the data plane.
- **Distributed Synchronization:** Maintaining synchronized playback across geographically distributed clients with varying network conditions (jitter, packet loss) presents significant timing challenges. This will be addressed through timestamping, adaptive buffering, and packet loss concealment strategies.
- **Media Processing Pipeline:** Efficiently handling user-submitted content requires a robust, asynchronous media processing pipeline to prevent blocking of the main server loop.

2.2 Application Domain Motivation

- **Social Connectivity:** Existing music platforms primarily cater to individual consumption. This project aims to create shared listening experiences that foster community engagement across physical distances.
- **Educational Value:** The system serves as a practical demonstration of core network programming concepts, including socket programming, protocol design, concurrency, and distributed systems challenges commonly encountered in production streaming services.

3 SYSTEM OBJECTIVES

3.1 Primary Objectives

1. **Reliable Control Plane Implementation:** Develop a robust TCP-based communication channel (via Javasockets) for playlist management, user commands, and

system state messages, ensuring reliable, in-order delivery.

2. **Custom Streaming Protocol Development:** Design and implement a lightweight, RTP-inspired protocol over UDP featuring packet sequencing, timestamping for synchronization, and flags for stream control (e.g., end-of-stream).

3. **Synchronized Playback Architecture:** Create client-side synchronization logic including an adaptive jitter buffer to mitigate network timing variations and a simple packet loss concealment (PLC) mechanism to handle lost UDP packets gracefully.

4. **Scalable Media Processing:** Implement an efficient, non-blocking server-side media processing pipeline to handle concurrent downloads and format conversions without impacting network performance.

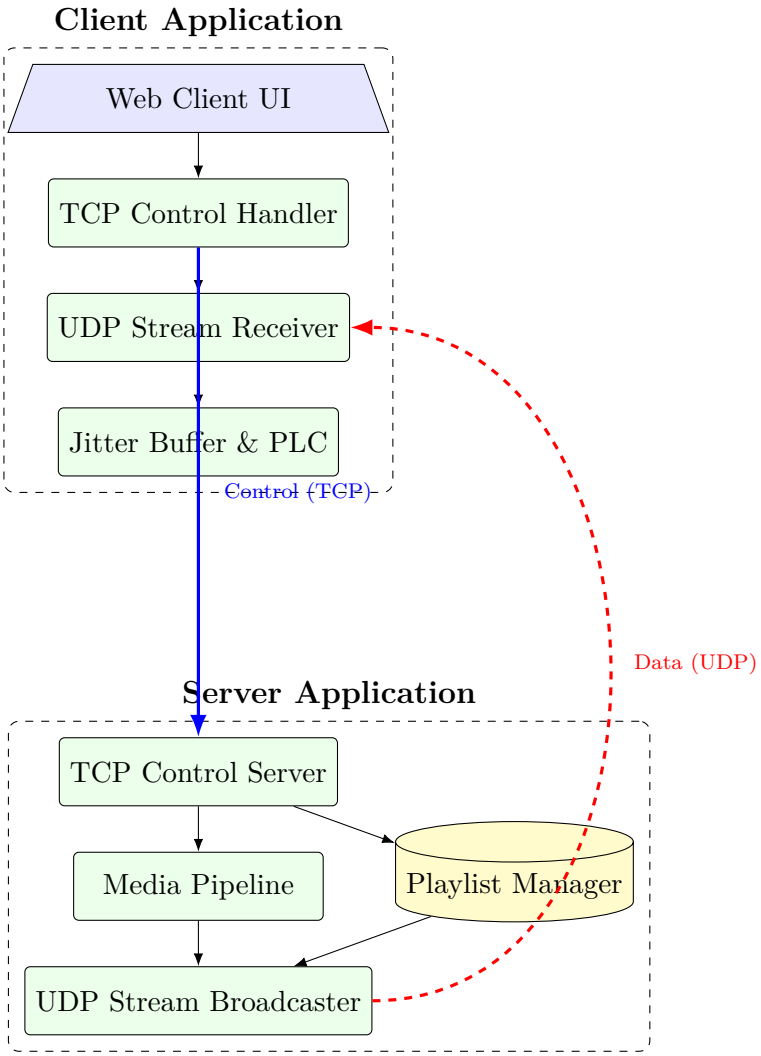
3.2 Performance Goals

- **Control Latency:** Achieve low latency for control commands over TCP to ensure a responsive user experience.
- **Synchronization:** Implement timestamp-based synchronization with adaptive client-side buffering to minimize playback drift between clients.
- **Throughput:** Support a baseline of 10+ concurrent clients streaming 128kbps audio.
- **Reliability:** Demonstrate graceful handling of packet loss on the UDP data plane.

4 SYSTEM ARCHITECTURE

4.1 High-Level Architecture

The system follows a centralized client-server model, cleanly separating the control plane (TCP) for reliability and the data plane (UDP) for low-latency streaming.



4.2 Component Specifications

4.2.1 Client Components

- **Web Client (React UI):** Browser-based interface for playlist interaction and playback control.
- **TCP Control Handler:** Manages the Javasocket connection for sending commands and receiving state updates from the server.
- **UDP Stream Receiver:** Listens for incoming UDP packets and passes them to the jitter buffer.
- **Jitter Buffer & PLC Engine:** Temporarily stores incoming audio packets to smooth out arrival time variations (jitter). Implements simple packet loss concealment (e.g., repeating the last good packet).
- **Audio Player:** Uses the Web Audio API to schedule and play audio chunks with precision, driven by the jitter buffer.

4.2.2 Server Components

- **TCP Control Server:** An asynchronous, multi-client server that processes commands, manages client state, and orchestrates other backend components.
- **Playlist Manager:** A thread-safe queue that manages the list of upcoming songs.
- **Media Pipeline:** An asynchronous worker that downloads content (using `yt-dlp`) and converts it to a standard PCM format (using FFmpeg).
- **UDP Stream Broadcaster:** Reads processed audio data, packetizes it according to the custom protocol, and broadcasts it to all connected clients.

5 PROTOCOL DESIGN

5.1 Custom UDP Streaming Protocol

The audio streaming protocol is designed to be lightweight and provides essential features for synchronization and stream management.

Field	Size (bytes)	Description
Sequence Number	4	Increasing packet counter for loss detection and reordering.
Timestamp	8	timestamp indicating the sampling instant of the first audio sample in the packet. Used for synchronization and jitter calculation.
Flags	1	stream control signal. E.g., '0x01' for Start of Stream (SOS), '0x02' for End of Stream (EOS).
Payload Length	2	Length of the audio data in bytes.
Audio Data	Variable	Raw audio samples.

Table 1: Custom UDP Packet Format

5.2 Control Protocol (Using JavaSockets)

Listing 1: Control-plane message examples over TCP

```
# Client → Server: subscribe to the live audio stream
{"type":"STREAM_SUBSCRIBE","payload":{"port":54321}}

# Server → Client: broadcast current playlist state
{"type":"PLAYLIST_UPDATE",
 "payload":{
   "queue":[{"title":"Song A"},{"title":"Song B"}],
   "now_playing":{"title":"Song A","duration":240.5}
 }}
```

6 IMPLEMENTATION WORKFLOW

The system's operation follows a clear sequence of events, coordinating actions between the client and server over TCP and UDP.

7 TECHNOLOGY STACK

7.1 Backend Technologies

- **Python 3:** Server media pipeline implementation.
- **Java:** Core Server Implementation.
- **Supporting Libraries:**
  - `yt-dlp` - For robust extraction of media from YouTube.
  - **FFmpeg (CLI):** Invoked as a subprocess for audio conversion and normalization.

7.2 Frontend Technologies

- **JAVAFX:** Will be used to build the interface for client.

8 CONCLUSION

This Collaborative Music Streamer project provides a robust framework for exploring and implementing core network programming principles. By architecting separate, fit-for-purpose control (TCP) and data (UDP) planes, the project offers a practical demonstration of the trade-offs inherent in network protocol design. The development of a custom streaming protocol, complete with mechanisms for synchronization and error handling, will provide deep, hands-on experience with the fundamental challenges of real-time media delivery.