CSC3935 Final Project

Audio Streaming and Library Management

Sean Kelley, Moses Mwaura, Aidan Varano

May 8th, 2024

The Project:

The goal of this project was to create a Music Library System implemented in Java. The system should allow for a user to interact with a command line interface (CLI) to manage, add songs and albums to, view the library's contents, and start a server for media streaming, which will then be accessed by a client. The client should be able to take a song from said music library system, and be able to play it back.

Key Features:

Music Library Management

Here, users can add songs and albums to the library. The system supports adding an album cover to the library for a song. These images are compressed and stored in a Base64 encoded format, in order to optimize and lower the required resources for storage.

CLI for Library Interaction

Here, users can Actually interact with the library, users can add songs, add albums, print the library, start the server, view the library, and exit the CLI. The specific commands for use in the CLI regarding library interaction are listed below.

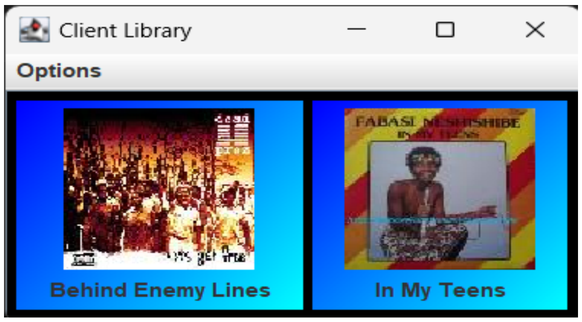
- AddSong <song_name>: Adds a new song to the library. You will be prompted to select an MP3 file and an image for the album cover.
- AddAlbum <album_name>: Adds a new album. You will select a directory that contains the songs.
- PrintLibrary: Displays the current contents of the library.
- StartServer: Initializes the server for media streaming.
- ViewLibrary: Opens the library view where you can see all songs and albums.
- Exit: Closes the CLI.

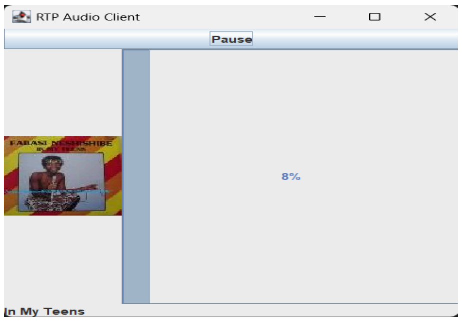
Client GUI

This GUI is for use when users wish to interact with specific songs within the management library. Actions that a user can perform within the GUI include:

- Setup Connection: Establishes a connection to the server.
- Play: Plays the selected song.
- Pause: Pauses the current song.
- Close Connection: Closes the streaming connection.
- Describe Stream: Provides details about the current stream.

Below are some examples of both the library view, and the media player GUI:





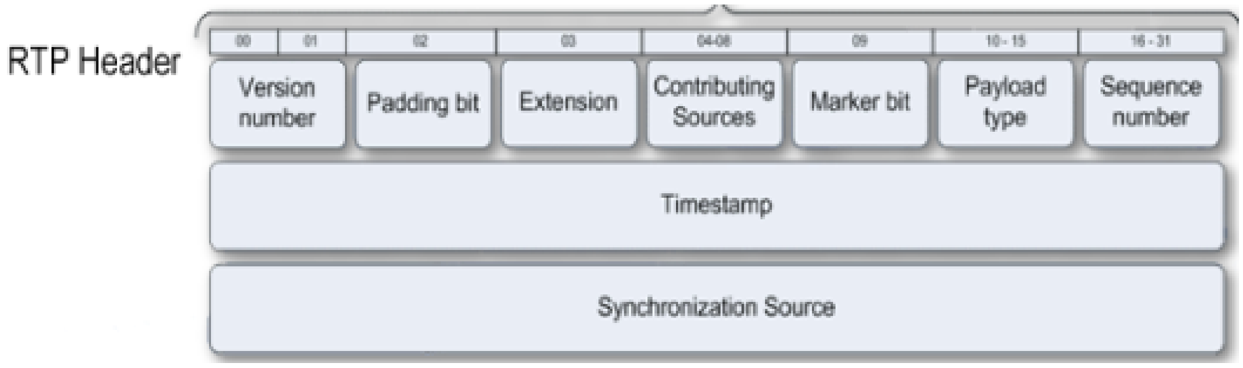
Real Time Protocol (RTP):

What is the Real Time Protocol?

The Real-Time Protocol (RTP) plays a crucial role in the realm of digital communications by specializing in the efficient delivery of audio, and video content over networks. This protocol operates over and on top of the User Datagram Protocol (UDP), taking advantage of its lightweight and connectionless nature to ensure timely transmission of media data. RTP adds some important functionalities such as sequence numbering, timestamps, and payload type identification, enabling receivers to reconstruct the transmitted multimedia streams in the correct order, even given the unreliability of UDP. By using the Real Time Protocol, applications can achieve real-time synchronization, error detection, and seamless playback experiences, making it an important protocol for usage in streaming and communication services and applications.

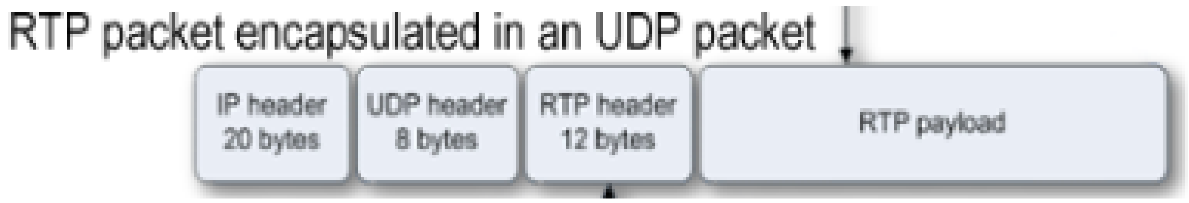
RTP Protocol Packets

RTP Packets are built to include both a header and a payload, with the header carrying details essential for the receiver's reconstruction of the media stream. Within the header, crucial information such as the Payload Type, which specifies the type of media being transmitted, the Sequence Number, used in organizing out-of-order packets, the Timestamp used for synchronization purposes, and the Synchronization Source Identifier (SSRC) uniquely identifying the stream, are all included. A diagram depicting the header, and it’s related fields can be found below:



In this diagram, the Payload Type, Sequence Number, Timestamp, and SSRC identifier / source are all clearly labeled. The number of bits each field takes is also shown using the numbers at the top of the diagram.

Additionally, the payload section of an RTP Packet contains the actual media data being sent by the sender, completing the comprehensive package necessary for real-time audio and video communication over IP networks. Below is an example of what the entire packet looks like:

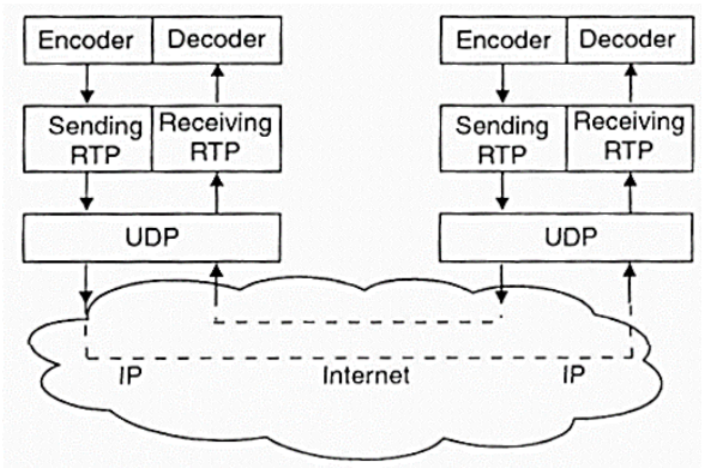


Real Time Protocol Flow:

The communication flow of the Real Time Protocol follows the order of Session Initiation, Packaging (encoding), transmission, Media Reconstruction (decoding), and Session Termination. Each step is described in detail below.

- Session Initiation
  - Before RTP packets are exchanged, a session needs to be initiated. This is typically done using the Session Initiation Protocol (SIP), which sets up any parameters needed (IP address and port numbers).
- Packaging
  - Once a connection is made, the sender starts packing media data into RTP Packets
- Transmission
  - RTP packets are sent over UDP. This supports time-sensitive delivery, but does not guarantee that packets will reach their destination.
- Media Reconstruction
  - RTP Packets are processed as they arrive. The receiver uses sequence numbers to reorder them if they arrive in the incorrect sequence, using timestamps to correct timing due to network delays.
- Session Termination
  - The session can be terminated using the control protocols (like SIP) that initiated the session.

The protocol flow can be seen below:



Session Management:

As mentioned above, RTP does not provide a mechanism for opening, maintaining, or closing sessions. These functions are typically handled by other protocols such as Session Initiation Protocol (SIP) or Session Description Protocol (SDP)

Payload Formats:

RTP can carry a wide variety of media payloads and the payload format is defined by the payload type filed in its header. The different types of data that RTP can send need to be agreed upon separately by the communicating devices before they start exchanging data.

Quality of Service (QoS):

Quality of service is crucial in real-time media applications. RTP helps manage QoS by organizing the data into a specific format and adding sequence numbers and timestamps. These features help reduce delays and rearrange any data that arrives out of order. However, RTP doesn’t handle other Qos tasks such as prioritizing certain types of data or reserving network resources. These tasks are managed by different protocols and settings in the network.

Security Considerations:

Security in RTP is bolstered by the Secure Real-Time Transport Protocol (SRTP) which provides encryption, message authentication, and integrity verification.

Conclusion:

Using the Real Time Protocol, we have achieved the primary goal of reliable and efficient audio streaming. Our project successfully facilitated the transmission of audio data between a server and a client in a manner that prioritizes real-time processing and minimal latency.

RTP's integration into the project was supported by data packets over UDP which is crucial for streaming media due to its low overhead. The use of RTP enhanced the system's ability to handle packet reordering and synchronization issues that are inherently associated with UDP's connectionless nature. This was achieved through RTP's sequence numbering and timestamp features. We ensured that audio streams are played back in the correct order and timing, despite network volatility.

Additionally, the project's implementation of RTP was complimented by a robust session management from initiation to termination. This system design supports user interactions through a command line interface and a client GUI. This not only improves user experience by providing a responsive streaming service, but also simulates the effectiveness of RTP in real-world applications.

Successfully developing the RTP within this project was important in highlighting its crucial role in contemporary multimedia communication systems especially where uninterrupted delivery and synchronization of streaming media are essential. RTP continues to be a foundational technology in networking and empowers creation of real-time communication platforms.

References:

RFC 3550