Documentation for Lab 03 : MAC Layer

Omkar Shirpure (22B0910)
Suryansh Patidar (22B1036)
Krish Rakholiya (22B0927)
Parthiv Sen (22B1055)

1 Introduction

This document provides a detailed explanation of the scripts `sender.py` and `receiver.py`. These scripts are designed to simulate the MAC Layer Implementation using audio signals. The process involves encoding data and sending the data using a certain protocol to ensure data is transmitted or received between multiple nodes as intended.

2 Sender Script: `sender.py`

The `sender.py` script is responsible for encoding and transmitting bitstrings as audio signals. It uses predefined frequencies to transmit data in pairs of bits, simulating a MAC layer protocol where multiple devices communicate by taking turns.

2.1 Function Overview

Below is an overview of the main functions in the `sender.py` script:

- `get_timestamp()`: Returns the current timestamp up to milliseconds.
- `encode_message(bitstring, dest)`: Adds the frame header (length, source, destination) to the bitstring and ensures it is ready for transmission.
- `generate_combined_tone(frequencies, duration, sample_rate)`: Generates a sine wave for the given frequencies and combines them to create a tone.
- `transmit(bitstring)`: Transmits the bitstring as audio by mapping each 2-bit pair to a frequency.
- `countdown_to_start(delay_until_start)`: Provides a countdown to the start of the MAC Layer transmission.
- `is_valid_time(start_time)`: Determines if it is the correct time for the device to transmit based on the assigned time slots.
- `read_file()`: Reads the bitstring and destination ID from the input file.
- `engine()`: The main control function that orchestrates reading messages, scheduling transmissions, and managing the message queue.

2.2 Execution Flow

1. The script starts by initializing the audio system using PyAudio and reading the necessary parameters like device ID and total devices.
2. The function `engine()` starts, which waits for the next minute to begin the transmission process.
3. The `read_file()` function reads messages from the input file, and valid messages are appended to a queue.
4. During the transmission time slot for the current device, the `transmit()` function encodes the message using `encode_message()` and transmits it as audio signals using mapped frequencies.
5. The script continues running until all queued messages are transmitted.

2.3 Conclusion

The `sender.py` script successfully demonstrates the implementation of a MAC Layer protocol using audio signals. By encoding bitstrings and transmitting them in predefined time slots, it ensures that multiple devices can transmit data without collisions.

3 Receiver Script: `receiver.py`

The `receiver.py` script is responsible for detecting, filtering, and decoding audio signals received as bitstrings. It decodes the messages from other devices in the network based on the frequency of the transmitted tones. The process includes recording the incoming audio, detecting the presence of valid bitstrings using frequency detection, and filtering the signal to extract useful information.

3.1 Function Overview

Below is an overview of the main functions in the `receiver.py` script:

- `detect_frequency(data, sample_rate)`: Detects the frequency of the incoming audio chunk using FFT to identify the peak frequency in the audio signal.
- `process_audio_data(audio_data)`: Filters the recorded audio data for each frequency band, computes the average amplitude, and decodes the bitstring by identifying the highest signal for each time slot.
- `decode_bitstring(bitstring)`: Extracts the actual message and source device ID from the decoded bitstring.
- `decode_and_print(audio_data, timestamp, count)`: Processes the received audio data and prints the decoded message if valid.
- `record_bitstream(count)`: Records the audio during a time slot, processes the audio data, and decodes the bitstring.
- `is_valid_time(start_time)`: Checks if the current time falls within the device's assigned time slot for recording.
- `engine()`: Orchestrates the recording process by managing time slots and controlling when the device can record based on its assigned slot.

3.2 Execution Flow

1. The script starts by initializing the audio system using PyAudio and waiting for the next available time slot based on the device's assigned slot in the network.
2. During the device's time slot, the `record_bitstream()` function is triggered, recording audio for the specified duration.
3. The `process_audio_data()` function is responsible for filtering the recorded audio to identify the relevant frequencies, which correspond to the transmitted bitstrings.
4. The `decode_bitstring()` function decodes the bitstring by identifying the source and destination devices and extracting the message.
5. The decoded message is then printed, and the script waits for the next time slot.

3.3 Conclusion

The `receiver.py` script successfully implements the reception and decoding of MAC Layer transmissions via audio signals. By filtering audio data, detecting frequencies, and decoding the received bitstreams, the script demonstrates how multiple devices can communicate using predefined time slots.