# External Project Report on
# Computer Networking (CSE3034)

## [Real-timeVoWi-FiAudio Streaming]
## (Using Socket Programing)

**Submitted by**

| | |
|---|---|
| **Name 01:** Sumit Saha | **Reg. No.:** 2141019013 |
| **Name 02:** Rishabh Kumar Agrahari | **Reg. No.:** 2141001083 |
| **Name 03:** Vs Prudhvi Rajeev Kumar | **Reg. No.:** 2141011145 |
| **Name 04:** Sibasish Padhy | **Reg. No.:** 2141016014 |
| **Name 05:** Talha Ayub | **Reg. No.:** 2141001071 |
| **Name 06:** Utkarsh Mishra | **Reg. No.:** 2141013143 |

**B. Tech. CSE 5th Semester (Section T)**

**INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH**
**(FACULTY OF ENGINEERING)**
**SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA**

## Declaration

We, the undersigned students of B. Tech. of **CSE** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled **“Real-timeVoWi- FiAudio Streaming (Using Socket Programing)”** submitted to **Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**.We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of othersfor using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible forthe same.

**(NAME1):** Sumit Saha
**Registration No.:** 2141019013

**(NAME2):** Rishabh Kumar Agrahari
**Registration No.:** 2141001083

**(NAME3):** Vs Prudhvi Rajeev Kumar
**Registration No.:** 2141011145

**(NAME4):** Sibasish Padhy
**Registration No.:** 2141016014

**(NAME5):** Talha Ayub
**Registration No.:** 2141001071

**(NAME6):** Utkarsh Mishra
**Registration No.:** 2141013143

**DATE: 13/01/2024**
**PLACE:BHUBANESHAR**

## Abstract

This   project introduces a real-time VoWi-Fi audio streaming solution through socket programming, enabling seamless communication between devices. Leveraging Java socket programming, the system establishes direct connections for low-latency audio transmission over Wi-Fi networks. The custom protocol ensures efficient packet exchange, allowing continuous streaming with minimal delay. Utilizing the Java Sound API for audio handling, the solution prioritizes compatibility and flexibility across platforms. Key features include peer-to-peer communication, reducing dependencies and enhancing system performance. This innovation contributes to real-time audio streaming, offering an accessible and efficient VoWi-Fi solution. Future developments may focus on security, multi-channel support, and mobile optimization, expanding the project's potential in diverse communication scenarios.

# Contents

# 1. Introduction

Voice over Wi-Fi (VoWi-Fi) has emerged as a crucial component in contemporary communication systems, leveraging the ubiquity of Wi-Fi networks for seamless voice transmission. In response to the escalating demand for efficient voice communication over wireless networks, this project endeavors to design and implement a fundamental VoWi-Fi system. The primary objective is to establish a reliable audio transmission mechanism within a single access point, achieved through the utilization of socket programming.

**Project Objective**:

The primary goal of this project is to create a VoWi-Fi system that captures audio input from a microphone on the client side, transmits it over a local network using socket programming, and facilitates playback on the server side. This comprehensive system aims to demonstrate the essential elements of real-time audio transmission and reception over Wi-Fi networks.

**Key Features:**

- Real-time audio capture from a client-side microphone.
- Socket-based transmission of audio data over a local network.
- Server-side reception and playback of transmitted audio.

# 2. Problem Statement

## I. Explanation of Problem:

The project addresses the challenge of implementing a fundamental Voice over Wi-Fi (VoWi-Fi) system within a single access point using socket programming. The primary problem involves the seamless flow of audio data from a client-side microphone to a server-side playback mechanism over a local network. The user interacts with the system by capturing audio elements through the microphone, which are then transmitted using socket programming. The desired result is the accurate playback of the audio on the server side. The project aims to establish a reliable and efficient solution for real-time audio communication over Wi-Fi networks.

## II. Highlighting Constraints:

The project operates within specific constraints to ensure a focused and practical implementation. These constraints include:

Single Access Point: The system is designed to function within the confines of a single access point, limiting the scope for broader network complexities.

Real-time Audio Transmission: The challenge involves achieving real-time audio transmission from the client side to the server side, necessitating efficient socket programming and network management.

Client-Side Microphone Integration: The system must seamlessly integrate with a client-side microphone, ensuring accurate capture of audio elements for transmission.

Server-Side Playback: The server must effectively process transmitted audio data and play it back in real-time, emphasizing the need for robust server-side handling.

Network Reliability: The success of the project depends on the reliability of the local network, as any disruptions may impact the quality of audio transmission.

.

## 3. Methodology(Algorithm/Pseudocode )

### VoWiFiClient :

#### Set Up Connection:
- Set the server's IP address and port.

#### Open Microphone:
- Define audio format (e.g., 16kHz, 16-bit, mono, signed, little-endian).
- Get the target data line for capturing audio.
- Open and start the target data line.

#### Create Socket Connection:
- Create a socket to connect to the server.
- Get the output stream from the socket.

#### Capture and Send Audio Loop:
- Create a buffer for audio data (e.g., byte array of 1024 bytes).
- Enter an infinite loop:
  - Read audio data from the target data line into the buffer.
  - Write the audio data to the output stream of the socket.

### VoWiFiServer :

#### Set Up Server:
- Set the server port.

#### Create Server Socket:
- Create a server socket to listen for incoming connections.

#### Accept Client Connection:
- Accept a client connection when a client connects.
- Print the client's IP address.

#### Open Speakers:
- Define audio format (e.g., 16kHz, 16-bit, mono, signed, little-endian).
- Get the source data line for playing audio.
- Open and start the source data line.

#### Create Buffer for Receiving Audio:
- Create a buffer for receiving audio data (e.g., byte array of 1024 bytes).

#### Receive and Play Audio Loop:
- Get the input stream from the socket.
- Enter an infinite loop:
  - Read audio data from the input stream into the buffer.
  - Write the audio data to the source data line for playback.

## 4. Implementation (Program)

### ClientSide:

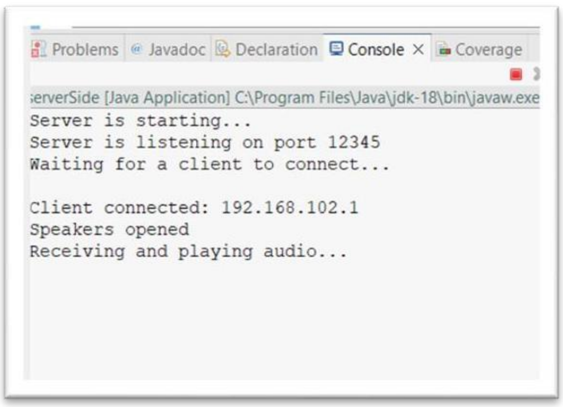
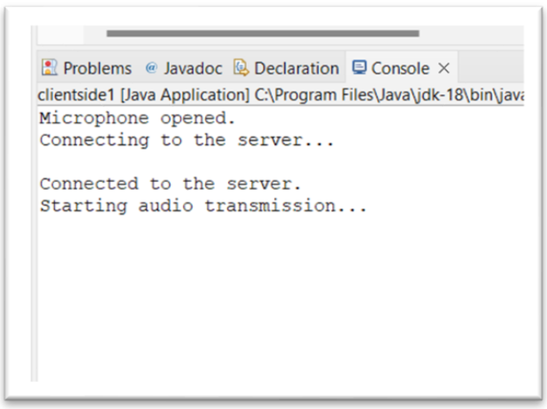
```
import javax.sound.sampled.*;
import java.io.*;
import java.net.Socket;

public class clientSide {

    public static void main(String[] args) {
        try {
                  String serverIp = "192.168.1.11";
                  int serverPort = 12345;

                  AudioFormat audioFormat = new AudioFormat(16000, 16, 1, true, true);
                  DataLine.Info info = new DataLine.Info(TargetDataLine.class, audioFormat);
                  TargetDataLine targetDataLine = (TargetDataLine) AudioSystem.getLine(info);
                  targetDataLine.open(audioFormat);
                  targetDataLine.start();
                  System.out.println("Microphone opened. \nConnecting to the server...");
                  System.out.println(" ");

                  Socket socket = new Socket(serverIp, serverPort);
                  OutputStream outputStream = socket.getOutputStream();

                  System.out.println("Connected to the server. \nStarting audio transmission...");
                  byte[] buffer = new byte[1024];

                  while (true) {
                          int bytesRead = targetDataLine.read(buffer, 0, buffer.length);
                          outputStream.write(buffer, 0, bytesRead);
                  }

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

## ServerSide:

```
import javax.sound.sampled.*;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class serverSide {

        public static void main(String[] args) {
                try {
                        int serverPort = 12345;
                        System.out.println("Server is starting...");

                        ServerSocket serverSocket = new ServerSocket(serverPort);
                        System.out.println("Server is listening on port " + serverPort);
                        System.out.println("Waiting for a client to connect...\n");

                        Socket socket = serverSocket.accept();`
                        System.out.println("Client connected: " + socket.getInetAddress().getHostAddress());

                        AudioFormat audioFormat = new AudioFormat(16000, 16, 1, true, true);
                        DataLine.Info info = new DataLine.Info(SourceDataLine.class, audioFormat);
                        SourceDataLine sourceDataLine = (SourceDataLine) AudioSystem.getLine(info);
                        sourceDataLine.open(audioFormat);
                        sourceDataLine.start();

                        System.out.println("Speakers opened \nReceiving and playing audio...");
                        byte[] buffer = new byte[1024];

                        InputStream inputStream = socket.getInputStream();

                       while (true) {
                                int bytesRead = inputStream.read(buffer, 0, buffer.length);
                                sourceDataLine.write(buffer, 0, bytesRead);
                       }

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

## 5. Results & Interpretation

Output screenshots with respect to inputs justifying the outcomes alongwith small explanations.

| Output | |
|---|---|
| VoWiFiServer | VoWiFiClient |
|  |  |

```
serverSide [Java Application] C:\Program Files\Java\jdk-18\bin\javaw.exe
Server is starting...
Server is listening on port 12345
Waiting for a client to connect...

Client connected: 192.168.102.1
Speakers opened
Receiving and playing audio...
```

```
clientside1 [Java Application] C:\Program Files\Java\jdk-18\bin\java
Microphone opened.
Connecting to the server...

Connected to the server.
Starting audio transmission...
```

Explanations:

### Continuous Audio Transmission:

The client continuously captures audio from the microphone and sends it to the server in chunks (buffer size of 1024 bytes). This results in a real-time audio transmission from the client to the server.

### Real-time Audio Playback:

On the server side, the received audio data is continuously played back through the speakers. This creates a real-time audio streaming effect where you can hear what the client is capturing through its microphone.

## 6. Conclusion

In this project, a basic Voice over Wi-Fi (VoWi-Fi) system was implemented using Java socket programming. The system successfully captures audio from a client's microphone, transmits it over a local network, and plays it back on the server side.

The following key points summarize the project outcomes:

### Client-Server Communication:

- The client successfully establishes a socket connection with the server, allowing for the transmission of audio data.

### Real-time Audio Streaming:

- The implemented system achieves real-time audio streaming, with audio captured from the client's microphone instantly played back on the server side.

### Code Execution Flow:

- The client-side code continuously captures audio from the microphone and sends it to the server in chunks. On the server side, the received audio is played back through the speakers in a continuous loop.

In conclusion, this project serves as a foundational implementation of a VoWi-Fi system usingJava socket programming. The system demonstrates the potential for real-time audio streaming over a local network

## References

Oracle Documentation for Java Sound API:

https://docs.oracle.com/javase/8/docs/technotes/guides/sound/

Java Tutorials - Networking Basics:

https://docs.oracle.com/javase/tutorial/networking/index.html

Java Socket Programming Tutorial:

https://www.javatpoint.com/socket-programming