Shri G.S. Institute of Technology & Science, Indore

Department of Computer Science & Engineering

Internet of Things
(CO34481)

Session- Jan23-Jun23

A Project Report on

"Wireless Announcement System"

Submitted To:

Prof. Priyanka Bamne

Prof. Ritambara Patidar

Submitted By:

Naman Hajela (0801CS201063)

Sudeep (0801CS201095)

Pushpendra Amoliya (0801CS201073)

Siddharth Swami Warke (0801CS201089)

Aryan Jain (0801CS193D02)

CERTIFICATE

This is to certify that Naman Hajela, Sudeep, Pushpendra Amoliya, Siddharth Swami Warke and Aryan Jain studying in B.Tech. 3nd year in session january2023 to june 2023 , have successfully completed the project “Wireless Announcement System” of the subject Internet of Things and have submitted a report having satisfactory details of it .

Date :-

25/04/23

Guided by :

Prof. Priyanka Bamne

Prof. Ritambara Patidar

ACKNOWLEDGEMENT

We would like to take this opportunity to express our gratitude to all those people who have helped us in our project, for their valuable guidance and experience which they poured on us in right direction. Their persistent encouragement, everlasting experience and valuable inspiration helped us a lot in molding the present shape of the project.

We would like to express gratitude to our staff members for their support too.

Table of contents -

1.INTRODUCTION

Traditional announcement systems in public spaces are often outdated, inefficient, and unreliable, as they rely on wired connections and central control rooms for managing announcements. Many existing systems also require expensive equipment and complicated installation processes, which limits their accessibility and practicality.

To address this gap in the market, we have designed a wireless announcement system that is affordable, easy to use, and customizable to meet the unique needs of different settings. The system uses Raspberry Pi connected to speakers to play announcements wirelessly, providing a more flexible and convenient solution compared to traditional wired systems.

Our system is designed to be scalable and adaptable, with the potential to integrate with other IoT devices and systems. The scope of the project includes the design, development, testing, and deployment of the system, with a focus on its reliability, scalability, and ease of use.

The goal of this project is to provide an innovative and effective solution for wireless playback of announcements, contributing to the evolution of IoT technology. In the following sections, we will discuss the technical details of our system and its implementation, as well as the results obtained. We will also explore the future potential of the system and provide recommendations for further development and research.

2.Problem Statement

Traditional announcement systems in public spaces, such as schools, hospitals, and government buildings, are often outdated, inefficient, and unreliable. These wired systems rely on complex wiring, central control rooms, and expensive equipment for managing announcements. Additionally, they can be difficult to install and maintain, limiting their accessibility and practicality.

Our goal is to address these challenges and provide a more reliable and user-friendly solution for announcements in public spaces. By utilizing wireless technology, we aim to create a system that is more flexible, scalable, and easier to install and maintain. This system will also be a better interface for announcement operators, as they don’t need to be in a control room to manage announcements. The system will be based on Raspberry Pi devices and a centralized server for managing announcements.

Design and Implementation:

Our wireless announcement system is based on the following components:

1. **Raspberry Pi:** serves as the audio playback controller for the project, responsible for receiving and playing back the announcements sent from the server.

2. **Speakers:** the output device, connected to the Raspberry Pi

3. **Mobile phone** (Telegram bot): used to record the announcements and send them to the server. (user friendly)

4. **Server:** receives the audio file from the mobile phone, processes it, and broadcasts it to the connected Raspberry Pi devices.

To implement our system, we first connected the Raspberry Pi devices to the speakers and installed the necessary software, including the Raspbian operating system and our MQTT client source code, along with a media player library. We also configured the Raspberry Pi devices to connect to the server over a Wi-Fi network.

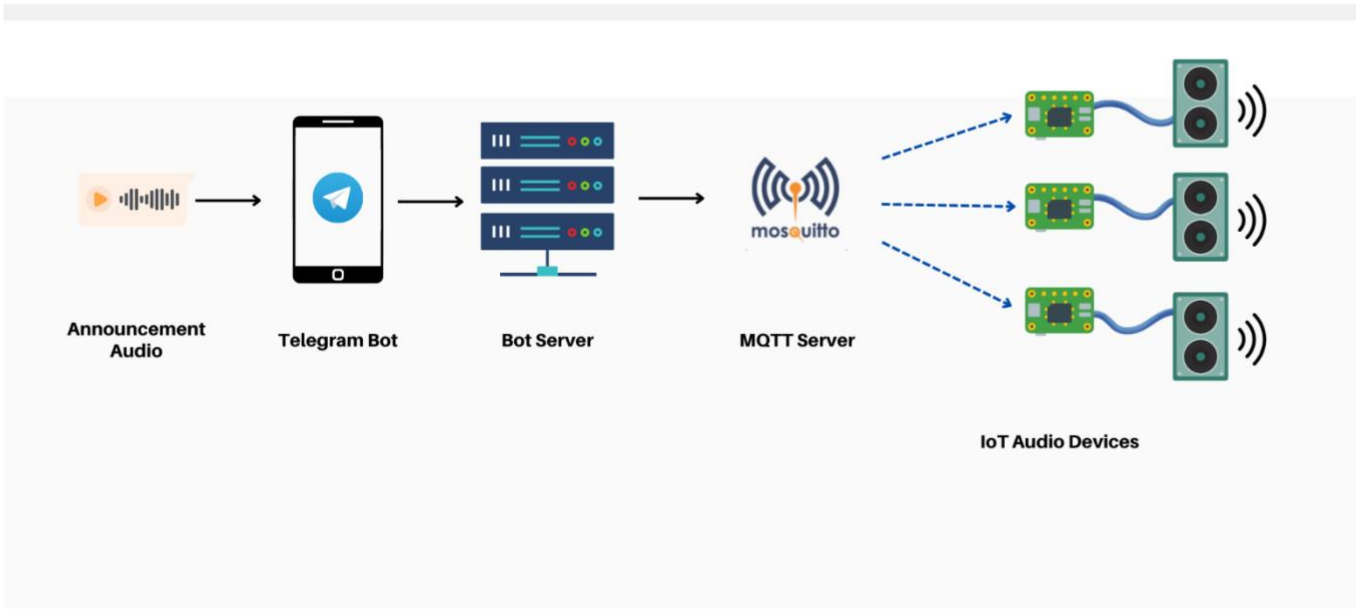
For the server component, we utilized Linode, to make our server platform that includes both the Telegram bot functionality for receiving audio files from users and MQTT functionality for broadcasting URLs to the Raspberry Pis.

When an announcement needs to be made, the user records the announcement using their mobile phone and sends it to the server using a Telegram bot. The server then processes the audio file and broadcasts it to all connected Raspberry Pi devices, which in turn play back the announcement on the attached speakers.

To ensure reliability and scalability, we designed the system to be modular and easily expandable. Additional Raspberry Pi or other IOT devices can be added as needed, and the system can be adapted to different network configurations and settings.

Overall, our wireless announcement system provides a more flexible and convenient solution compared to traditional wired systems. It is also cost-effective and easy to install and maintain, making it an ideal solution for various public spaces

System Diagram



4. COMPONENT USED –

	Name of component	**Specification**	**Quantity**	**Cost per piece (in Rs)**
1.	Raspberry Pi3 B	1 GB RAM 1.6 GHz	1	1500/-
2.	Micro Usb Cable		1	50/-
3.	Speakers		3	300/-
	Total Price			1850/-

5.COMPONENT DETAILS –

Raspberry Pi: The Raspberry Pi is a single-board computer that serves as the main processing unit for the wireless announcement system. We used a Raspberry Pi 3 Model B with 1GB RAM, which is more than sufficient for our use case. The Raspberry Pi runs a custom Python script that subscribes to the Adafruit IO MQTT broker and plays back audio received from the server. It is also responsible for managing the connection to the speakers and controlling the playback of the audio.

Speakers: We used simple USB-powered speakers for our prototype, but any speakers with a 3.5mm jack or USB connection can be used. The speakers are connected to the Raspberry Pi's audio output jack or USB port and are controlled by the custom Python script running on the Raspberry Pi.

Telegram Bot: The Telegram Bot API is a simple and powerful way to create user interfaces for messaging applications. We used the Telegram Bot API to implement the user interface for recording and sending audio announcements. The bot was hosted on a cloud server using a Python script that integrates with the Adafruit IO MQTT broker to send the audio to the Raspberry Pi. The bot accepts voice messages from users, downloads them from the Telegram server, and publishes the URL of the downloaded audio file to the MQTT broker.

Adafruit IO: Adafruit IO is a cloud service that provides an easy and secure way to connect IoT devices to the internet. We used Adafruit IO to store and manage the audio sent from the Telegram bot. The Raspberry Pi subscribes to the Adafruit IO MQTT broker to receive the audio URLs and play them back through the connected speakers. Adafruit IO provides an intuitive web interface for managing data feeds and monitoring device activity.

Code:

Raspberry Pi Code:

```
# Import standard python modules.
import sys
from vlcModule import play  # Import custom VLC media player module.

# Import Adafruit IO MQTT client.
from Adafruit_IO import MQTTClient

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = ''

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = ''

# Set to the ID of the feed to subscribe to for updates.
FEED_ID = 'audiourl'


# Define callback functions which will be called when certain events happen.

# Function to handle connection to Adafruit IO.
def connected(client):
    """
    Connected function will be called when the client is connected to Adafruit IO.
    This is a good place to subscribe to feed changes. The client parameter passed
    to this function is the Adafruit IO MQTT client so you can make calls against
    it easily.
    """
    print('Connected to Adafruit IO!  Listening for {0} changes...'.format(FEED_ID))
    # Subscribe to changes on a feed named 'audiourl'.
    client.subscribe(FEED_ID)


# Function to handle subscription to a new feed.
def subscribe(client, userdata, mid, granted_qos):
    """
    This method is called when the client subscribes to a new feed.
    """
    print('Subscribed to {0} with QoS {1}'.format(FEED_ID, granted_qos[0]))


# Function to handle disconnection from Adafruit IO.
def disconnected(client):
    """
    Disconnected function will be called when the client disconnects.
    """
    print('Disconnected from Adafruit IO!')
    sys.exit(1)


# Function to handle new message received from a subscribed feed.
def message(client, feed_id, payload):
    """
    Message function will be called when a subscribed feed has a new value.
    The feed_id parameter identifies the feed, and the payload parameter has
    the new value.
    """
    print('Feed {0} received new value: {1}'.format(feed_id, payload))
    # Play the audio file from the URL using the custom VLC media player module.
    player = play(payload)
    print('done')


# Create an MQTT client instance.
client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Setup the callback functions defined above.
client.on_connect    = connected
client.on_disconnect = disconnected
client.on_message    = message
client.on_subscribe  = subscribe

# Connect to the Adafruit IO server.
client.connect()

# Start a message loop that blocks forever waiting for MQTT messages to be
# received.  Note there are other options for running the event loop like doing
# so in a background thread--see the mqtt_client.py example to learn more.
client.loop_blocking()
```

Telegram Bot Code:

```
# Import required modules
import telebot
import requests
import os
import time
from dotenv import load_dotenv
from Adafruit_IO import Client, Data

# Load environment variables from .env file
load_dotenv()
TOKEN = os.getenv('TOKEN')
AIO_USERNAME = os.getenv('AIO_USERNAME')
```

```
AIO_KEY = os.getenv('AIO_KEY')
aio = Client(AIO_USERNAME, AIO_KEY)

# Create a Telegram bot instance
bot = telebot.TeleBot(TOKEN)

# Set the IP address of the server and the directory to store audio notes
SERVER_IP = ""
AUDIO_NOTES_DIR = "/home/main/iotScript/audio_notes"

# Define a handler for the 'announce' command
@bot.message_handler(commands=['announce'])
def announce_command_handler(message):
    """
    Handler function to respond to the 'announce' command.
    """
    bot.reply_to(message, "Please send an audio file/voice note.")

# Define a handler for voice messages
@bot.message_handler(content_types=['voice'])
def voice_message_handler(message):
    """
    Handler function to process incoming voice messages.
    """
    print('voice message received')
    # Get the file ID of the voice message
    voice_file_id = message.voice.file_id
    voice_file = bot.get_file(voice_file_id)
    voice_file_path = voice_file.file_path
    voice_url = f"https://api.telegram.org/file/bot{TOKEN}/{voice_file_path}"
    # Download the voice file
    response = requests.get(voice_url)
    # Save the voice file in the local folder with a human-readable filename
    file_name = f"{time.strftime('%Y-%m-%d_%H-%M-%S')}.ogg"
    file_path = os.path.join(AUDIO_NOTES_DIR, file_name)
    with open(file_path, 'wb') as f:
        f.write(response.content)
    print('file saved')
    # Publish the file URL to Adafruit IO after the file has finished writing and been closed
    while True:
        if os.path.exists(file_path):
            if not os.path.isfile(file_path):
                continue
            if not f.closed:
                f.close()
            # Construct the URL for the audio file and publish it to Adafruit IO
            file_url = f"http://{SERVER_IP}/audio_notes/{file_name}"
            data = Data(value=file_url)
            aio.create_data('audiourl', data)
            break
        else:
            time.sleep(1)
    # Keep only the last 3 audio notes
    files = os.listdir(AUDIO_NOTES_DIR)
    files.sort(key=lambda x: os.path.getctime(os.path.join(AUDIO_NOTES_DIR, x)))
    if len(files) > 3:
        oldest_file = os.path.join(AUDIO_NOTES_DIR, files[0])
        os.remove(oldest_file)

# Start the Telegram bot and listen for incoming messages
print('running')
bot.polling()
```

Future Work:

While our wireless announcement system provides a highly effective and reliable solution for announcements in public spaces, there is still room for further development and research. As a proof of concept, our system demonstrates the potential for wireless technology to provide a more flexible and convenient solution for announcements in various public settings.

One potential area for future work is the integration of other IoT devices and systems along with cheaper audio playback devices. The mode can be expanded to include other sensors and smart lighting devices, to create a more comprehensive and automated solution for managing public spaces. For example, the system could be integrated with occupancy sensors to automatically trigger announcements when a certain number of people are present in a room.

Another area for future work is the development of more sophisticated audio processing algorithms to improve the audio quality and reduce background noise in the announcements. This could involve the use of machine learning algorithms to automatically filter out noise and enhance the clarity of the announcements.

In addition, the system can be further optimized for power consumption to extend the battery life of the mobile devices and audio playback devices, enabling longer periods of use without the need for frequent recharging.

Finally, our system can be scaled up for larger public spaces and customized to meet the unique needs of different settings, including different languages and cultural preferences.

In conclusion, our wireless announcement system is a proof of concept that demonstrates the potential for wireless technology to provide a more flexible and convenient solution for announcements in public spaces. Similar systems can be developed and implemented at scale to provide effective and reliable solutions for various public settings.

Conclusion:

We have developed a wireless announcement system using Raspberry Pi devices that provides a more flexible and convenient solution for announcements in public spaces. The system is cost-effective, easy to install and maintain, and customizable to meet the unique needs of different settings. We have demonstrated that it is reliable, scalable, and provides decent audio quality. Our project is a proof of concept that demonstrates the potential for wireless technology to provide a more efficient and effective solution for announcements in public spaces.

With further development and research, similar systems can be developed and implemented at scale to provide effective and reliable solutions for various public settings. Overall, our wireless announcement system provides an innovative and effective solution for wireless playback of announcements, contributing to the evolution of IoT technology. We believe that our system has the potential to revolutionize the way announcements are made in public spaces and we look forward to seeing its continued development and adoption.