A Locally Controlled Smart Intercom System Using ESP32-CAM and Raspberry Pi 5

First AUTHOR[1,2], Nita Constantin Cosmin
[1]*Stefan cel Mare University of Suceava, 720229, Romania*
Constantin.nita2@student.usv.ro

***Abstract*—This paper presents the design and implementation of a smart intercom system that integrates live video streaming, local access control, and mobile maintenance capabilities. The system is based on a low-cost architecture combining an ESP32-CAM microcontroller and a Raspberry Pi 5 server. It eliminates the need for third-party cloud services by enabling all communication and processing locally. The developed system provides real-time video, authentication-based access, and a mobile application for remote maintenance. Results show that the system achieves low latency and high reliability for residential and small-scale commercial environments.**

Keywords: ESP32-CAM, Raspberry Pi, smart intercom, IoT, local control, video streaming

I. Introduction

Smart home systems have seen increasing adoption, with intercoms becoming a key point of integration for communication and security. Traditional intercoms are limited in functionality, while commercial smart solutions often rely on cloud platforms, raising concerns about privacy, dependency, and cost. This work proposes a hybrid intercom solution with **local intelligence and remote maintenance**, built using open-source tools and low-cost components.

The main contributions of this study are:

- Development of a real-time video intercom using ESP32-CAM;
- Implementation of a Raspberry Pi 5-based web server for access control;
- Integration of a mobile app for system diagnostics and control;
- Experimental validation of performance (latency, stability).

II. System Architecture

A The system comprises three major components:

1. **Video Capture Module:** Based on ESP32-CAM, responsible for real-time streaming.
2. **Control and Processing Unit:** A Raspberry Pi 5 running a Django-based web server.
3. **Maintenance App:** A mobile application (developed in Flutter) for diagnostics and firmware control.

The ESP32-CAM connects to the local Wi-Fi and streams MJPEG to the Pi, which serves the video feed to the web interface. A REST API allows the mobile app to interact with the Raspberry Pi for operations like rebooting the camera module or retrieving logs.

clearances.

III. Hardware and Software Design

3.1 ESP32-CAM Configuration

The ESP32-CAM is programmed using the Arduino IDE. Streaming is handled via `esp32-camera` and `WiFiClient`. The resolution is fixed to 320x240 to reduce latency.

#include "esp_camera.h"
#include <WiFi.h> minutes per page.

The stream is accessed via a public endpoint exposed to the local network.

3.2 Raspberry Pi Server

The Raspberry Pi runs a Django application with the following modules:

- `video_feed`: renders the MJPEG stream
- `user_auth`: handles login and access control
- `maintenance`: exposes diagnostics and control APIs

The web interface is accessible only from local IPs for security reasons.

3.3 Mobile App

The app communicates with the server through HTTPS. It can:
restart the ESP32 module
check logs from /var/log/intercom
send update commands (firmware via OTA)

This work was supported in part by the E.U. Commission under Grant EUV13/20160727/AE (**sponsor and financial support acknowledgment goes here - modify or delete this note if it is not the case**).

IV. Theoretical Model

To estimate the total latency L_t from camera capture to user display, we consider:

$$L_t = L_c + L_n + L_r$$

Where:

- L_c is the camera capture and encoding delay,
- L_n is the network transmission delay,
- L_r is the rendering delay on the web interface.

From empirical testing, we modeled L_n as a function of network congestion ρ using:

$$L_n(\rho) = \frac{B}{R(1-\rho)}$$

Where B is the stream bitrate (kbps), and R is the network throughput.

V. Results

Tests were performed in a local network (Wi-Fi 5, 100 Mbps nominal). Key observations include:

- Latency: Average end-to-end latency was 280 ms.
- Stability: The system sustained continuous operation for 72 hours without restart.
- Security: Local-only access prevented external probes; logs showed zero unauthorized access.

Metric	**Value**
Latency (avg)	280 ms
Max stream uptime	72 hours
Firmware updates	Supported OTA

VI. Discussion

Compared to commercial solutions (e.g., Ring, Nest Hello), our system provides:

- Full data control (no third-party cloud),
- Cost efficiency (~70 EUR total component cost),
- Customizability (open-source software stack).

However, limitations include:

- Limited camera resolution,
- Dependence on local network quality,
- Manual firewall configuration for VPN access.

VII. Conclusion

This paper demonstrated a cost-effective, private, and scalable smart intercom system. Future work includes AI-based face recognition for access control and integration with smart door locks. The open architecture allows adaptation in rural or off-grid areas where internet access is limited.

VIII. Conclusion

A conclusion section is not compulsory, but we recommend it. Although a conclusion may review the main points of the paper, do not replicate the abstract. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Try to emphasize your scientific contribution and the differences from previous works in the literature. The conclusion is a text only section - do not use equations, graphs or cite references in this section. Make sure that the whole text of your paper observes the textual arrangement on this page.

References

[1] Espressif Systems, "ESP32-CAM Technical Reference Manual", 2023.

[2] Raspberry Pi Foundation, "Raspberry Pi 5 Documentation", 2024.

[3] Django Software Foundation, "Django Framework Documentation", [Online]. Available: https://docs.djangoproject.com

[4] A. Tanenbaum, “Computer Networks”, 5th ed., Pearson, 2011.