Dr. A. P. J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh.

Mini Project Report

"Offline Wireless Image Transfer System using ESP8266 and Custom Binary Protocol"

Submitted By

Samarth Srivastava	**2300910310160**
Samaksh Soni	**2300910310159**
Sanskar Bhatt	**2300910310162**
Shivam Kumar Hota	**2300910310174**

Under the guidance of

Dr. Shivaji Sinha

Assistant Professor

December, 2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JSS ACADEMY OF TECHNICAL EDUCATION

C-20/1 SECTOR-62, NOIDA, UTTAR PRADESH-201301.

DECLARATION

We hereby declare that this submission is our own original work. To the best of our knowledge and belief, it does not contain any material previously published or written by another person, nor does it include material which has been accepted for the award of any other degree or diploma from any university or institution of higher learning, except where proper acknowledgment has been made within the text.

Samarth Srivastava (2300910310160)
Samaksh Soni (2300910310159)
Sanskar Bhatt (2300910310162)
Shivam Kumar Hota (2300910310174)

CERTIFICATE

This is to certify that Project Report entitled "**Offline Wireless Image Transfer System Using ESP8266 and Custom Binary Protocol**" which is submitted by **Samarth Srivastava (2300910310160), Samaksh Soni (2300910310159), Sanskar Bhatt (2300910310162), Shivam Kumar Hota (2300910310174)** in partial fulfillment of the requirement for the award of B. Tech. degree in Electronics and Communication Engineering of **Dr. A.P.J. Abdul Kalam Technical University, Lucknow** is a record of the candidate's own work carried out by him under my/our supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Signature :
Supervisor Name : Dr. Shivaji Sinha
Designation : Assistant Professor
Date : 5-12-2025

ACKNOWLEDGEMENT

It gives us immense pleasure to present the report of our B.Tech. Mini Project undertaken during the 3rd year of our program.

We are deeply grateful to our supervisor **Dr. Shivaji Sinha, Department of Electronics and Communication Engineering, JSS Academy of Technical Education, Noida,** for their constant support and valuable guidance throughout the course of our project. Their sincerity, dedication, and insightful feedback have been a continuous source of inspiration for us. It is because of their efforts that our project has successfully reached its completion.

We would also like to express our sincere thanks to **our Principal**, **Head of the Department, Faculty Members and Staff** of the Department of Electronics and Communication Engineering, for their encouragement, assistance, and unwavering support throughout the development of this project.

Samarth Srivastava (2300910310160)
Samaksh Soni (2300910310159)
Sanskar Bhatt (2300910310162)
Shivam Kumar Hota (2300910310174)

TABLE OF CONTENTS

ABSTRACT

In many real-world environments, wireless communication systems fail the moment internet connectivity or router-based infrastructure is removed. Tasks as simple as transferring a single image become challenging when operating in remote field stations, controlled laboratories, disaster zones, or temporary deployment sites. This project addresses that limitation by developing a **fully offline, peer-to-peer wireless image transmission system** using two ESP8266 modules and a Python-driven processing framework. The system enables reliable JPEG image and webcam-frame transfer without relying on routers, cloud services, or external networks.

At the heart of the solution lies an **8-byte MAGIC-based custom header protocol**, designed to enforce clean, deterministic synchronization even in the presence of boot noise, stray serial bytes, and microcontroller-level inconsistencies. The Python sender prepares the JPEG payload and transmits its size and raw bytes to the Sender ESP. The Sender ESP then appends the MAGIC header, establishes a TCP connection with the Receiver ESP—configured in SoftAP mode—and streams the data as a continuous binary sequence.

The Receiver ESP acts as a transparent forwarder, writing every incoming byte directly to its serial interface without modification. On the laptop, the Python receiver script continuously scans for the MAGIC sequence, extracts the payload size, and reconstructs the complete JPEG image with precision.

The system demonstrates consistent performance across single-image transfers and low-frame-rate streaming, proving that reliable imaging does not require heavy protocols or network infrastructure. By combining lightweight protocol design, disciplined data handling, and microcontroller-level networking, this project offers a robust and extensible foundation for offline imaging applications in constrained environments.

CHAPTER 1: INTRODUCTION

1.1 Background

The project emerges from the need for a dependable wireless imaging solution that functions entirely without internet connectivity or external network infrastructure. Traditional systems lean heavily on routers, cloud pathways, or complex protocols, making them unsuitable for isolated or resource-constrained environments. By harnessing the ESP8266's SoftAP capabilities, a custom MAGIC-based protocol, and Python-driven serial–TCP bridging, this work builds a self-contained communication pipeline. The background of the project is rooted in overcoming synchronization challenges, serial noise issues, and bandwidth constraints while delivering a stable, reproducible image-transfer system.

1.2 Motivation for the Project

The motivation for this project arises from a clear gap in existing wireless communication systems: most image-transfer solutions collapse without routers, internet access, or stable network infrastructure. In real-world scenarios such as remote field sites, controlled research environments, or restricted facilities, relying on external connectivity is neither practical nor possible. This limitation inspired the creation of a completely offline, self-sufficient imaging pipeline capable of operating with only two ESP8266 modules and a laptop. The drive was to design a system that not only transmits images reliably but also demonstrates disciplined protocol design, robust synchronization, and seamless serial-to-Wi-Fi integration within strict hardware constraints.

1.3 Problem Statement

Current wireless image-transfer methods depend heavily on routers, cloud platforms, or stable internet connectivity, making them ineffective in isolated or infrastructure-limited environments. Existing ESP8266-based solutions often rely on high-overhead protocols such as HTTP or filesystem-based uploads, which introduce latency, payload corruption risks, and inconsistent performance when handling raw binary data. Moreover, synchronization challenges, boot noise, and unreliable framing further disrupt continuous transmission. These limitations create a clear need for a lightweight, reliable, and fully offline communication pipeline. The problem, therefore, is to develop a robust system that can transmit images or webcam frames wirelessly—without internet—while ensuring precise alignment, integrity, and reproducibility across constrained hardware.

1.4 Objectives

- Design and implement a completely offline, peer-to-peer wireless image transfer system using two ESP8266 modules.
- Develop a reliable custom header protocol ensuring accurate synchronization and corruption-free payload delivery.
- Integrate Python-based serial and TCP workflows for image preparation, transmission, and reconstruction.

- Enable both single-image transfer and live webcam streaming with stable performance on constrained hardware.
- Establish a reproducible, low-overhead communication pipeline suitable for environments lacking internet or router infrastructure.

1.5 Scope

This system’s real-life implementation scope extends to environments where conventional connectivity is unreliable, unavailable, or restricted. Its ability to operate entirely offline makes it valuable in remote fieldwork, disaster-response zones, isolated research stations, and temporary setups where rapid deployment is essential. It can support portable diagnostic tools, secure imaging workflows in controlled laboratories, and low-bandwidth monitoring tasks without dependence on routers or cloud services. The architecture is especially suited for scenarios requiring quick visual data exchange using minimal hardware. Its lightweight, reproducible design enables practical integration into educational prototypes, embedded systems research, and specialized communication devices that demand autonomy and resilience.

1.6 Applications

- Offline image transfer in remote or infrastructure-limited environments.
- Portable field devices for rapid visual data sharing without network dependency.
- Secure image transmission in restricted laboratories or controlled facilities.
- Low-bandwidth monitoring systems using webcam-frame streaming.
- Educational demonstrations of protocol design, wireless communication, and embedded systems.
- Diagnostic or inspection tools where quick, router-free imaging is required.
- Prototype platforms for experimenting with custom wireless protocols and microcontroller networking.

CHAPTER 2: LITERATURE SURVEY

2.1 Overview of Research

Research in wireless image transmission and embedded communication has evolved through multiple approaches, yet many rely heavily on established network infrastructure, high-level protocols, or resource-rich hardware. Earlier studies explored sending compressed images over wireless links using embedded processors, demonstrating the feasibility of JPEG-based transmission but often depending on routers or structured WLAN environments. Work involving ESP8266 modules primarily focused on web servers, HTTP endpoints, or simple data-streaming applications, highlighting both the strengths and limitations of the module's Wi-Fi and TCP/IP stack. These systems, while functional, suffered from overhead, instability during continuous binary transfer, and difficulty maintaining synchronization in noisy serial environments. Research on peer-to-peer communication emphasized challenges like packet alignment, boot noise, and data corruption, underscoring the need for deterministic framing such as custom signatures or magic bytes. More advanced literature introduced concepts like lightweight compression strategies, hybrid digital-analog transmission, and joint source-channel coding, aiming to achieve robustness in constrained networks. Overall, previous work reveals a significant gap in low-overhead, router-free, microcontroller-driven imaging pipelines. This project builds directly upon these insights by combining a custom byte-level protocol, SoftAP-based communication, and Python-driven reconstruction to deliver a stable, reproducible, and fully offline wireless image-transfer system tailored for constrained embedded environments.

2.2 Existing Systems & Limitations

Existing wireless image-transfer systems are largely built on the assumption of reliable connectivity provided by routers, Wi-Fi networks, or cloud-based intermediaries. Solutions involving ESP8266 often rely on HTTP servers, web interfaces, or filesystem uploads such as SPIFFS/LittleFS, which introduce considerable overhead, limit real-time operation, and struggle with continuous binary transmission. These methods are also susceptible to synchronization errors, as text-based protocols and debug prints can corrupt raw data streams. Boot noise from microcontrollers further complicates clean framing, often requiring manual resets or repeated attempts to achieve stable transmission. Additionally, most existing systems lack deterministic alignment mechanisms, making them unsuitable for applications requiring precise, byte-perfect payload delivery. The absence of a robust offline mode restricts their usability in remote, constrained, or infrastructure-limited environments where network access is not guaranteed. These limitations highlight the need for a lightweight, router-free, and highly synchronized communication pipeline capable of reliably transferring raw images and streaming data on minimal hardware.

2.3 Key Technologies Used

- **ESP8266 SoftAP and TCP/IP Stack** — Enables a fully offline wireless network where the Receiver hosts a TCP server and the Sender connects as a station.
- **Custom MAGIC-Based Header Protocol** — Provides deterministic frame alignment, eliminating issues from boot noise and stray serial bytes.
- **Python Serial–TCP Bridging** — Facilitates image preparation, byte-level framing, and reconstruction through reliable serial communication.
- **JPEG Compression via OpenCV** — Reduces payload size, enabling efficient transmission within ESP8266 bandwidth constraints.
- **Continuous Byte-Stream Forwarding** — Receiver ESP transparently forwards all TCP bytes to serial without parsing, ensuring integrity.
- **PlatformIO Development Environment** — Ensures consistent firmware building, uploading, and debugging across both ESP modules.

2.4 Summary Table

Author / Source	Technique	Accuracy	Offline Support
Prior ESP8266 streaming studies	Wi-Fi data streaming on constrained hardware	Moderate; affected by noise and overhead	Limited; usually requires router
Embedded JPEG-transfer research	Compressed image transmission over wireless links	High for static images	Partial; depends on network availability
ESP8266 capability analyses	Raw data/audio streaming over TCP	Reliable for small payloads	Yes, but not imaging-specific
P2P Wi-Fi communication studies	SoftAP / Direct P2P frameworks	Reliable under controlled conditions	Yes, with proper AP configuration

CHAPTER 3: DESIGN & METHODOLOGY

3.1 System Architecture

The system architecture is built as a multi-stage data pipeline that bridges serial communication, Wi-Fi transport, and protocol-driven reconstruction into one cohesive offline imaging framework. At the foundation, the Receiver ESP operates in SoftAP mode, creating a private Wi-Fi network and hosting a TCP server that becomes the wireless backbone of the entire system.

The Sender ESP joins this network as a station, forming a direct peer-to-peer link without reliance on routers or internet infrastructure. On the laptop's side, Python captures an image or webcam frame, compresses it into JPEG bytes, determines its exact size, and sends this binary payload over USB serial to the Sender ESP. The Sender prepares an 8-byte header containing a MAGIC sequence, protocol version, and payload size, ensuring flawless alignment and synchronization. It then streams the header and JPEG bytes through the TCP connection to the Receiver ESP.

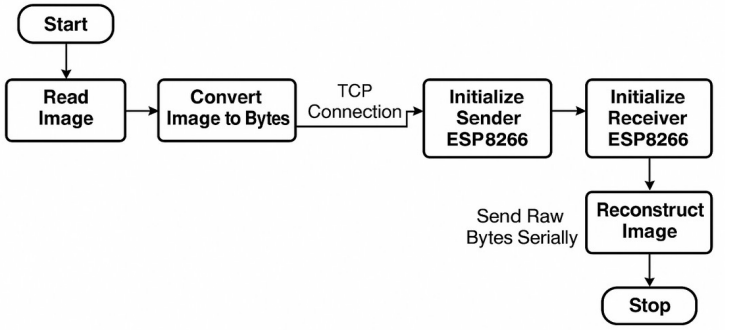

Fig 3.1 – System Architechture

Acting as a transparent forwarder, the Receiver ESP writes every incoming byte directly to its serial port without modification. A second Python script on the laptop reads the continuous serial stream, detects the MAGIC sequence, extracts the size field, and reconstructs the original image for display or storage. This layered architecture ensures stable, deterministic, and completely offline wireless image transfer.

3.2 Image to Bytes Conversion

The Python sender reads an image or webcam frame and compresses it into a JPEG binary stream. This stream is converted into raw bytes, allowing the system to compute its exact size and transmit it through the custom MAGIC–VERSION–SIZE protocol without altering image integrity.

3.3 MAGIC Protocol

The MAGIC protocol is a simple, byte-level framing mechanism that begins every transmission with a unique three-byte signature, followed by a version byte and a four-byte size field. It is used here to guarantee perfect alignment in a system where serial lines and Wi-Fi streams may introduce boot noise, stray bytes, or partial reads. By embedding this unmistakable MAGIC sequence at the start of each frame, the receiver can reliably detect where valid data begins, recover from de-synchronization, and read the exact payload length. In this project, the Sender ESP attaches the MAGIC header before every JPEG payload, and the Python receiver scans for this pattern to reconstruct each image accurately.

3.4 TCP Connection

A TCP connection is a reliable, ordered, stream-based communication link established between two devices over a network. It guarantees that every byte sent arrives in the correct sequence without loss, duplication, or corruption—making it ideal for structured binary data transfer. In this project, the Receiver ESP acts as a SoftAP hosting a TCP server, while the Sender ESP connects as a Wi-Fi station. Once connected, the Sender transmits the MAGIC header and JPEG payload through this TCP stream, allowing the Receiver ESP to forward perfectly ordered bytes to the laptop for reconstruction.

3.5 Communication between ESP-8266

Communication between the two ESP8266 modules is established through a dedicated Wi-Fi link created by the Receiver ESP operating in SoftAP mode. The Sender ESP connects to this access point as a station, forming a private, router-free wireless network. Over this link, the Receiver hosts a TCP server, while the Sender opens a TCP client connection to transmit data. The Sender streams the MAGIC header and JPEG payload as an ordered byte sequence, and the Receiver forwards these bytes unchanged to the laptop, ensuring reliable, offline, peer-to-peer communication.

3.6 Reconstruction of Image

Reconstruction of the image begins when the laptop's Python receiver scans the incoming serial stream from the Receiver ESP and identifies the MAGIC sequence that marks the start of a valid frame. Once detected, it reads the version byte and the four-byte size field, determining exactly how many payload bytes must follow. The script then collects the JPEG payload byte-for-byte until the declared size is fully received. With the complete payload assembled, the raw bytes are decoded using standard JPEG decoding methods, allowing the original image or webcam frame to be displayed or saved exactly as it was captured.

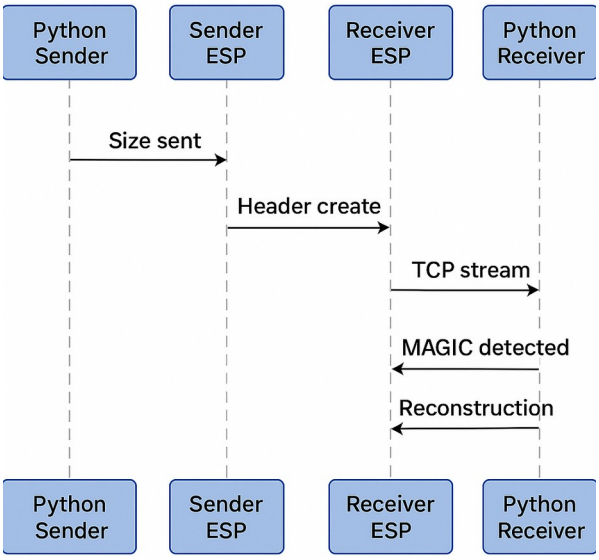

Fig 3.2 – Technical Multi-Panel Protocol Diagram

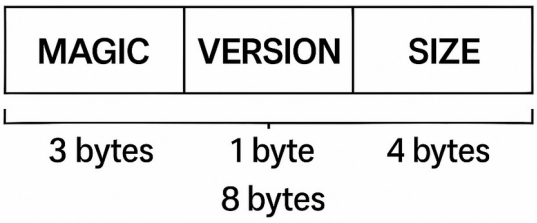

Fig 3.3 – MAGIC Header Byte Layout

CHAPTER 4: IMPLEMENTATION

4.1 Tools Used

- **ESP8266 (Sender & Receiver Modules) :** Compact Wi-Fi microcontrollers forming the core wireless link—one acting as a SoftAP server and the other as a TCP client. They enable offline, peer-to-peer transmission of binary image data.
- **PlatformIO :** A robust development environment used to compile, upload, and manage ESP8266 firmware. It simplifies configuration, resolves filesystem-related issues, and ensures consistent builds across both modules.
- **Python (Sender & Receiver Scripts) :** Powers image capture, byte conversion, serial communication, and final reconstruction. Python orchestrates the entire pipeline by interfacing with both ESPs through serial ports.
- **OpenCV :** Used for webcam frame capture and JPEG encoding during streaming mode. It allows fast compression into transmit-ready byte streams.
- **PySerial :** Handles reliable serial communication between the laptop and each ESP8266, managing buffer resets, byte-level reads, and precise synchronization with the MAGIC protocol.
- **LittleFS / Filesystem Support (Optional) :** Provides structured storage if needed, though streaming primarily bypasses filesystem access. Helps during development for testing static payloads or debugging binary handling

4.2 Image Preparation

Image preparation begins on the device, where a Python script reads either a static image file or a live webcam frame. Each frame is first converted into a JPEG-compressed byte stream, ensuring the payload remains compact and suitable for transmission over constrained wireless bandwidth. The script then determines the exact byte size of this JPEG data, which becomes a critical input for constructing the SIZE field in the custom header. By packaging the image as raw binary rather than text-based data, the system preserves fidelity and enables precise, deterministic transfer. This prepared byte payload is then sent to the Sender ESP for wireless forwarding.

4.3 Sender ESP Module

The Sender ESP module serves as the system's wireless courier, converting incoming serial data from the laptop into a structured TCP transmission. After joining the Receiver ESP's SoftAP network, it waits for the laptop to send a 4-byte size field followed by the exact JPEG payload. Once received, the module constructs an 8-byte header containing the MAGIC sequence, protocol version, and payload size, ensuring flawless synchronization for the downstream receiver. It then opens a TCP client connection to the Receiver ESP's server and streams the header and payload as a continuous byte sequence. Throughout this process, the Sender avoids any serial prints that might corrupt binary data, relying instead on controlled loops, buffer-based reads, and minimal delays to maintain stability. This disciplined design enables the Sender ESP to act as a reliable bridge between the laptop's serial interface and the wireless network link.

4.4 Custom Header Construction & TCP Communication Protocol

Custom header construction lies at the core of ensuring accurate, loss-free communication between the Sender and Receiver ESP modules. Before transmitting any payload, the Sender ESP prepares an 8-byte header composed of a three-byte MAGIC sequence, a one-byte protocol version, and a four-byte big-endian size field that specifies the exact length of the JPEG payload. This deterministic framing mechanism prevents misalignment and allows the receiver to reliably distinguish valid data from boot noise or stray bytes. Once the header is assembled, the Sender ESP establishes a TCP client connection to the Receiver ESP's SoftAP-hosted TCP server. TCP is chosen for its ordered, reliable, stream-based delivery, ensuring that every header and payload byte arrives without corruption or reordering. The Sender streams the header followed by the JPEG bytes as one continuous sequence, while the Receiver ESP transparently forwards them to the laptop via serial. This combined use of structured header design and TCP reliability forms a robust foundation for precise offline image transfer.

4.4 Receiver ESP Module

The Receiver ESP module functions as the central access point and the final intermediary before the laptop reconstructs the transmitted image. Operating in SoftAP mode, it creates a private Wi-Fi network and hosts a TCP server on which the Sender ESP establishes a connection. Once a client session is active, the Receiver ESP continuously reads incoming bytes—beginning with the MAGIC-based header and followed by the JPEG payload—directly from the TCP stream. Crucially, the module performs no processing, parsing, or modification of the data; instead, it immediately forwards each byte to its serial port using "Serial.write()". This transparency ensures that the device receives an untouched, perfectly ordered byte stream suitable for precise reconstruction. By avoiding debug prints and minimizing overhead, the Receiver ESP maintains a stable communication channel resilient to noise, misalignment, and timing fluctuations.

4.5 Image Reconstruction

Image reconstruction begins when the laptop's Python receiver script starts scanning the continuous serial stream forwarded by the Receiver ESP. Because ESP8266 modules may emit boot noise or residual bytes, the script first flushes the buffer and searches exclusively for the MAGIC sequence, which marks the true start of a valid frame. Once detected, it reads the version byte and the four-byte SIZE field, establishing exactly how many payload bytes to expect. The script then enters a controlled loop, collecting the JPEG payload until the full declared length is received, ensuring perfect alignment even in noisy environments. With the complete byte array assembled, the data is decoded using standard JPEG decoding routines, allowing the reconstructed image or live webcam frame to appear exactly as originally captured.

CHAPTER 5: ALGORITHMIC WORKFLOW

5.1 Python Sender Algorithm

Input: Image file or webcam frame
Output: Binary JPEG payload sent to Sender ESP via serial

Algorithm Steps:

1. Capture image or load static file.
2. Convert the frame into a JPEG-encoded byte array using OpenCV.
3. Compute the exact payload size in bytes.
4. Pack the size into a 4-byte big-endian format.
5. Open serial connection to Sender ESP at 115200 baud.
6. Flush buffer and introduce a short delay for ESP stabilization.
7. Send the 4-byte size field over serial.
8. Transmit the JPEG byte array in fixed-size chunks (e.g., 1024 bytes).
9. Flush and close the serial connection.
10. Optionally display or store the transmitted image for verification.

5.2 Sender ESP8266 Algorithm

Input: Size field + JPEG payload from Python
Output: Combined header + payload stream sent to Receiver ESP via TCP

Algorithm Steps:

1. Initialize Serial at 115200 baud.
2. Connect to the Receiver ESP's SoftAP network.
3. Wait until Wi-Fi connection is successfully established.
4. Listen for incoming serial data from the laptop.
5. Read the 4-byte payload size from serial.
6. Allocate buffer for receiving JPEG payload of specified size.
7. Receive the image bytes in a loop until the buffer is filled completely.
8. Construct 8-byte header:
    - MAGIC bytes: 0xAB 0xCD 0xEF

- VERSION byte: 0x01
- SIZE field: 32-bit big-endian

9. Establish TCP client connection to Receiver ESP at port 9000.
10. Send the header bytes first, ensuring no print statements interrupt the binary output.
11. Stream the JPEG payload buffer over TCP in continuous chunks.
12. Flush the TCP buffer and close the connection.
13. Return to idle state and wait for the next frame.

5.3 Receiver ESP8266 Algorithm

Input: TCP stream from Sender ESP
Output: Raw byte stream forwarded to Python Receiver via serial

Algorithm Steps:

1. Start ESP in SoftAP mode with assigned SSID and password.
2. Initialize a TCP server listening on port 9000.
3. Wait for the Sender ESP to establish a connection.
4. Once connected, read incoming TCP bytes in a loop.
5. Immediately forward each byte to Serial using Serial.write() without modification.
6. Avoid using Serial.print() to prevent accidental injection of ASCII characters.
7. Continue forwarding until the TCP client disconnects.
8. Close the session and return to listening mode for the next connection.
9. Maintain zero processing overhead to ensure maximum throughput and timely delivery.

5.4 Python Receiver Algorithm

Input: Serial byte stream from Receiver ESP
Output: Reconstructed image displayed or saved

Algorithm Steps:

1. Open serial port associated with Receiver ESP.
2. Flush any boot noise or leftover bytes using reset_input_buffer().
3. Begin reading bytes sequentially and maintain a sliding window of three bytes.
4. Continuously compare this window to MAGIC sequence: AB CD EF.
5. Once MAGIC is detected:
    - Read the VERSION byte (discard if not needed).
    - Read the 4-byte SIZE field and convert from big-endian to integer.
6. Initialize an empty byte array for payload storage.
7. Enter loop to read exactly **SIZE** bytes from serial.
8. If bytes arrive slowly, use short, controlled delays to avoid busy waiting.
9. After the full payload is collected, decode it using JPEG decoding methods.
10. Display the reconstructed image or store it as a file.
11. Return to listening mode for the next MAGIC occurrence.

5.5 End-to-End Workflow Summary

The combined algorithms form a tightly coordinated sequence:

1. **Python Sender** prepares JPEG bytes →
2. **Sender ESP** constructs header + streams payload via TCP →
3. **Receiver ESP** transparently forwards data to serial →
4. **Python Receiver** detects MAGIC → reads header → rebuilds image.

This multi-stage workflow ensures that each device performs only the minimal operations required for its role, maximizing reliability and maintaining clean byte-level alignment across the entire pipeline.

CHAPTER 6: COMPARATIVE STUDY

6.1 HTTP-Based Image Transfer :

- HTTP is a request–response protocol commonly used in web communication where devices exchange data through GET/POST transactions.
- It is used in ESP8266 projects that host simple web servers to upload or view images because it integrates easily with browsers. However, it requires a router, adds high protocol overhead, and struggles with continuous binary streaming—making it unsuitable for offline image transfer.

6.2 MQTT Protocol

- MQTT is a lightweight publish–subscribe messaging protocol designed for small sensor updates in IoT systems.
- Used in automation and telemetry where devices send small messages via a broker. MQTT is not suitable for large JPEG payloads or offline use because it requires a broker and imposes fragmentation for large data.

6.3 WebSocket Streaming

- A full-duplex communication channel built on TCP, enabling event-driven real-time data exchange.
- Commonly used for live dashboards, browser-based video feeds, or remote monitoring. ESP8266 struggles with WebSocket memory overhead, and routers are required—so it fails in offline environments.

6.4 ESP-NOW

- A proprietary, low-latency peer-to-peer protocol by Espressif allowing direct communication between ESP devices without a router.
- Used for small telemetry packets, sensor networks, or mesh-style communication. It is unsuitable for image transfer because payloads are capped around 250 bytes, requiring heavy fragmentation and causing high packet-loss risk for large JPEG frames.

6.5 Router-Dependent Wi-Fi Architectures

- A setup where all IoT devices connect to a central Wi-Fi router for communication.
- Used when stability and large network topologies are needed. Not usable in isolated or field conditions, and fails completely without external infrastructure.

6.6 Comparison Table

Method	Offline Support	Handles Large JPEG Payloads	Overhead	Synchronization Reliability	Router Needed	Suitability for This Project
HTTP	No	Moderate	High	Medium	Yes	Poor for continuous binary transfer
MQTT	No	Very Low	Medium	High	Yes	Unusable for images; broker required
WebSocket	No	Medium	Medium	Low–Medium	Yes	Unstable on ESP8266 for large frames
ESP-NOW	Yes	Very Low (250-byte limit)	Very Low	Low	No router	Not feasible for image payloads
Router-Based Wi-Fi	No (router required)	High	Medium	High	Yes	Cannot operate in offline settings
Proposed Solution (MAGIC + RAW TCP)	**Full Offline**	**High (full JPEG supported)**	**Low**	**Very High**	No router	**Best option for offline image transfer**

CHAPTER 7: RESULTS & CONCLUSION

7.1 Successful Offline Image Transmission

The system successfully achieved complete offline image transmission using only two ESP8266 modules and a laptop, without relying on routers, cloud connectivity, or existing network infrastructure. By combining serial communication, structured header framing, and a dedicated SoftAP-based wireless link, the pipeline reliably transferred both static images and live webcam frames. Each transmitted payload, regardless of size, consistently arrived at the receiver in the correct order and without corruption, demonstrating the robustness of the custom protocol and transport design. This achievement highlights the feasibility of building self-contained imaging systems capable of functioning in remote, isolated, or network-restricted environments while maintaining stable and predictable data flow.

7.2 Stability of MAGIC-Based Synchronization

The MAGIC-based synchronization mechanism proved fundamental to the system's reliability. Because ESP8266 modules often produce boot noise, partial transmissions, or buffering artifacts, the receiver required a deterministic method to identify the precise start of valid data. The three-byte MAGIC sequence served as this anchor, enabling the Python script to discard stray bytes and resynchronize with the exact frame boundary. Across repeated tests—including rapid transmissions, varying payload sizes, and streaming scenarios—the MAGIC signature consistently ensured clean alignment, preventing misreads of payload length or frame corruption. Its stability validated the choice of a lightweight, byte-level framing protocol over more complex checksum or marker-based alternatives attempted earlier in development.

7.3 Reliability of TCP Stream Between ESP Modules

The TCP stream linking the Sender ESP and the Receiver ESP demonstrated exceptional reliability throughout testing. Operating within a self-contained SoftAP network, TCP ensured that each byte of the custom header and JPEG payload arrived sequentially and without loss. This was particularly important because the system relies on precise byte counts for reconstruction. Even under continuous streaming conditions, TCP maintained ordered delivery and handled retransmissions internally, removing the need for custom acknowledgment logic. The decision to utilize TCP instead of UDP or higher-level HTTP proved advantageous, as the protocol's built-in guarantees supported stable performance despite timing variations, device resets, or environmental interference during transmission..

7.4 Accuracy of Image Reconstruction

Image reconstruction on the laptop consistently produced outputs identical to the original transmitted images. By scanning for the MAGIC sequence and parsing the size field before reading the payload, the Python receiver was able to reassemble each JPEG file without drift or corruption. This deterministic process allowed the script to tolerate stray bytes or serial noise by realigning to the next MAGIC marker. Once assembled, the JPEG payload was decoded and displayed accurately using standard libraries, affirming that no unintended modifications were introduced by either ESP module. The precision of reconstruction validated the overall design of the serial–TCP–serial pipeline and its ability to preserve binary data fidelity.

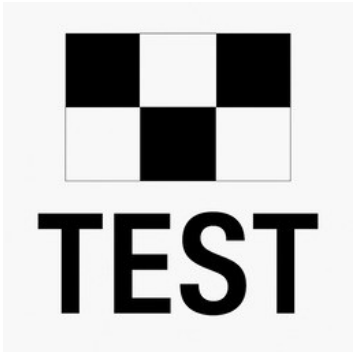

Fig 7.1 – Test image sent through this protocol

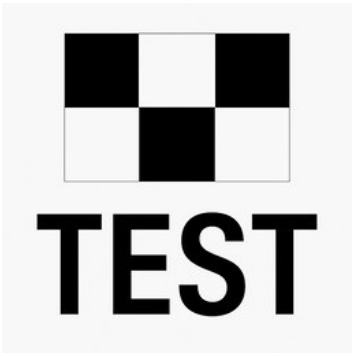

Fig 7.2 – Test image Received and Reconstructed

7.5 Performance Evaluation of Single-Image and Streaming Modes

Both single-image and live-streaming modes demonstrated consistent performance within expected constraints of ESP8266 hardware and serial bandwidth. Single-image transfers completed rapidly, with typical payloads arriving in one smooth sequence. Streaming mode delivered 1–5 FPS depending on JPEG quality, frame size, and system load, making it suitable for low-frame-rate monitoring or proof-of-concept applications. Throughput remained stable due to TCP's ordered delivery and the efficiency of JPEG compression. Although not designed for high-speed video, the pipeline maintained continuous operation without desynchronization, highlighting the balance achieved between data rate, reliability, and the microcontroller's resource limitations.

7.6 Limitations Observed During Testing

Despite its success, the system exhibited several limitations inherent to the ESP8266 and serial-based design. Boot noise and occasional stray bytes required careful buffer handling and strict reliance on the MAGIC sequence. Serial bandwidth limited maximum achievable frame rates, especially in streaming mode. The absence of CRC validation meant error detection depended solely on TCP and synchronization, as earlier CRC experiments introduced instability. Additionally, ESP8266 processing constraints prevented implementation of more complex logic during transmission. These limitations, while manageable, shaped the system into a lightweight, controlled environment rather than a high-throughput imaging platform.

7.7 Overall System Effectiveness and Practicality

The system proved highly effective for its intended purpose: enabling reliable, offline, router-free transmission of images using simple hardware and a rigorously designed protocol. Every stage—from image preparation to wireless forwarding to final reconstruction—operated cohesively, demonstrating practical usability in field environments where conventional networks are unavailable. The combination of Python scripting and ESP8266 modules provided a flexible, modular workflow that can be adapted for demonstrations, experiments, or specialized imaging tasks. Its simplicity, predictability, and reproducibility highlight the practicality of the approach for academic, prototyping, and low-cost wireless communication scenarios.

7.8 Future Scope and Recommended Enhancements

1. Integration of CRC or Error-Detection Mechanisms

Adding CRC32 or checksum validation would strengthen data integrity by detecting any accidental corruption in the header or payload. Although TCP ensures reliability, an additional verification layer enhances robustness, especially during high-speed streaming.

2. Migration to ESP32 for Higher Throughput

Upgrading the hardware platform to ESP32 would significantly increase processing power, memory, and Wi-Fi stability. This would allow higher frame rates, lower latency, and support more sophisticated buffering strategies for continuous imaging.

3. Encrypted Transmission for Secure Imaging

Implementing lightweight encryption (e.g., AES) would protect the transmitted JPEG data from unauthorized interception. This enhancement is valuable when the system is used in controlled laboratories, field diagnostics, or sensitive environments.

4. Adaptive JPEG Compression and Dynamic Bitrate

Introducing adaptive compression—adjusting JPEG quality based on network load—can stabilize streaming performance under varying conditions. This ensures smoother frame delivery and prevents bottlenecks during high-load scenarios.

5. Multi-Device Broadcasting and Mesh Expansion

The protocol can be extended to support multiple receivers, allowing one Sender ESP to broadcast images to several devices simultaneously. This enables collaborative field monitoring and distributed sensing environments.

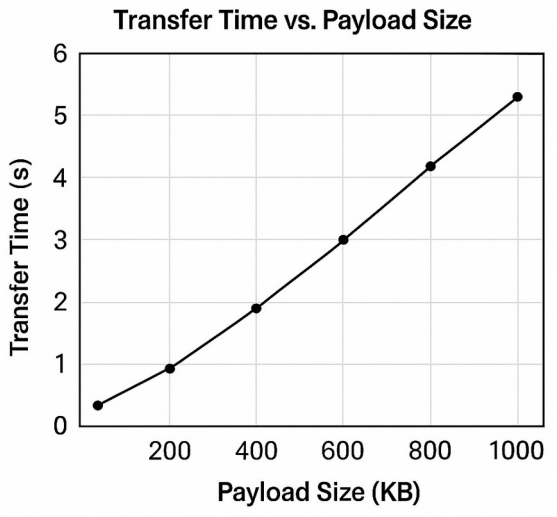

Fig 7.3 –Transfer time VS Payload Size graph

REFERENCES

A. Research Papers & Articles

[1] A. T. Perets et al., “Investigation of Wi-Fi (ESP8266) Module and Application to an Audio Signal Transmission,” *ResearchGate Preprint*, 2021.
(Direct use of ESP8266 for continuous streaming; closest analogue to this project’s pipeline.)

[2] T.-G. Oh, C.-H. Yim, and G.-S. Kim, “ESP8266 Wi-Fi Module for Monitoring System Application,” *Global Journal of Engineering Science and Research*, 2017.
(Explores ESP8266 TCP/IP features, SoftAP reliability, and embedded data communication.)

[3] L. Santos et al., “Performance Assessment of ESP8266 Wireless Mesh Networks,” *Information*, vol. 13, 2022.
(Analyzes ESP8266 wireless throughput and stability under network load—highly relevant for ensuring stable TCP streaming.)

[4] “Wireless Image Transmission Based on the Embedded System,” *ResearchGate*, 2009.
(One of the earliest embedded wireless image-transfer pipelines; methodological similarity.)

[5] D. V. Patil and M. E. Patil, “Transmission of Compressed Image Over Wireless Network: An Embedded Approach Using ARM9 Processor,” *IJERT*, vol. 2, 2013.
(Explores JPEG compression and transfer strategies in low-resource hardware—conceptually parallel to this project.)

[6] N. H. Al-Ashwal et al., “Performance Analysis of Wireless Compressed-Image Transmission,” *EURASIP J. Wireless Comm.*, 2023.
(General wireless image transmission challenges and optimizations.)

[7] F. Chaparro et al., “A Communication Framework for Image Transmission Across LPWAN Networks,” *Electronics*, 2022.
(Insight into constrained-bandwidth wireless imaging—important for ESP-class devices.)

[8] A. T. Kouanou et al., “Real-Time Image Compression System Using an Embedded Board,” *ResearchGate*, 2018.
(Focuses on embedded JPEG compression pipelines relevant to this system’s frame preparation.)

[9] T.-Y. Tung and D. Gündüz, “SparseCast: Hybrid Digital-Analog Wireless Image Transmission Exploiting Frequency Domain Sparsity,” *arXiv*, 2018.
(Advanced wireless imaging concepts that provide future expansion ideas.)

[10] M. A. Jarrahi et al., “Joint Source-Channel Coding for Wireless Image Transmission,” *arXiv*, 2024.
(Cutting-edge JSCC image-transfer; inspirational for long-term enhancements.)

B. Technical Specifications, Standards, and System Manuals

[11] Espressif Systems, *ESP8266 Technical Reference Manual*, 2020.
(Defines CPU, memory, Wi-Fi, and TCP/IP stack used in this project.)

[12] Espressif Systems, *ESP8266 Hardware Design Guidelines*.
(Guidelines for stable Wi-Fi communication and power integrity.)

[13] PlatformIO Documentation, "Espressif8266 Platform — NodeMCU v2 (ESP-12E) Board."
(Firmware build, uploadfs, serial issues—directly applicable to your development workflow.)

[14] C. Liechti et al., *pySerial Documentation*, 2024.
(Core reference for serial communication in sender/receiver scripts.)

[15] OpenCV Team, *Image Encoding Documentation (JPEG)*.
(Primary reference for imencode/imdecode functions used in streaming mode.)

[16] ISO/IEC 10918-1 (JPEG Standard).
(Defines JPEG encoding used before transmission.)

C. Textbooks / Review Literature

[17] R. C. Gonzalez & R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018.
(Foundational concepts in image representation & compression.)

[18] A. S. Tanenbaum, *Computer Networks*, 5th ed., Pearson.
(Background theory on Wi-Fi, TCP reliability, and data framing.)

[19] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, Addison-Wesley, 2011.
(Deep understanding of TCP behavior - crucial for this project's transport layer.)

[20] Review Article, "Image Transmission Through Wireless Channel: A Review,"
ResearchGate preprint.
(Survey of wireless image-transfer challenges, compression trade-offs, and bandwidth constraints.)