Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# A Self-Configurable IoT Gateway Using Cellular Module Without Native Library Support: Web-Based Integration of SMS, GPS, and Cloud Data Acquisition

**Lucas Lima Freitag**

houslast@gmail.com

ufrn https://orcid.org/0009-0006-8849-5619

**Research Article**

**Keywords:** Internet of Things (IoT), ESP32, AT commands, SIM7600, A7670SA, Cellular IoT gateway, Web-based configuration, TagoIO, Remote monitoring, Embedded systems.

**Posted Date:** July 28th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-10483764/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** The authors declare no competing interests.

# A Self-Configurable IoT Gateway Using Cellular Module Without Native Library Support: Web-Based Integration of SMS, GPS, and Cloud Data Acquisition

Lucas Lima Freitag*
Author
Federal University of Rio Grande do Norte (UFRN), Brazil
ORCID: 0009-0006-8849-5619

July 25, 2026

## Abstract

This paper presents a complete IoT gateway solution based on the ESP32 microcontroller and the SIM7600/A7670SA cellular module, developed to address the lack of native software libraries for these modules in the Arduino ecosystem. The proposed system implements a fully functional data acquisition, cloud communication, and device management platform using direct AT command manipulation over serial interface, eliminating the dependency on proprietary libraries. The architecture features a dual-core design where Core 0 handles all modem communication while Core 1 manages a built-in web server that provides a complete graphical configuration interface. The system supports automatic network mode fallback (5G/4G/3G/2G), GPS positioning with tower triangulation fallback, SMS-based alerts, and bidirectional data transmission with the TagoIO cloud platform. A buffered data sending mechanism aggregates multiple sensor readings into a single HTTP request, reducing network overhead and power consumption. Experimental results demonstrate that the 2G mode achieves the fastest connection establishment ( 3.4 s) while 4G provides better throughput for data transmission. The complete firmware is open-source and available on GitHub, enabling easy deployment in rural and remote monitoring applications.

## 1 Introduction

The Internet of Things (IoT) has experienced exponential growth in recent years, with forecasts projecting more than 31 billion connected devices by 2030 [1]. Among these, remote monitoring applications in agriculture, environmental sensing, and infrastructure management often require connectivity in areas where traditional wired networks are unavailable and WiFi coverage is limited or nonexistent. In such scenarios, cellular communication modules become essential, offering wide-area coverage through existing mobile network infrastructure.

*E-mail: houslast@gmail.com

However, the integration of cellular modules with low-cost microcontrollers presents several challenges. Many cellular modules available in the market, such as the SIMCom SIM7600 series and the A7670SA, lack native library support for popular development frameworks like Arduino. This absence forces developers to implement low-level AT command communication, which requires careful handling of serial protocols, timing constraints, and error recovery mechanisms [2].

The problem is particularly acute in the context of scientific and industrial applications where reliability, configurability, and maintainability are paramount. Without a standardized software abstraction layer, each implementation becomes a bespoke solution, difficult to reproduce, validate, or extend. Furthermore, the configuration of network parameters (APN, authentication, network mode), SMS management, GPS data handling, and cloud platform integration typically require separate tools and interfaces, increasing system complexity.

This paper addresses these challenges through the following contributions:

1. A complete, open-source firmware implementation for the ESP32+SIM7600 platform that requires no external libraries for modem communication, using only direct AT commands over serial.

2. A built-in web-based configuration interface that eliminates the need for external programming tools for device setup, enabling configuration of WiFi, cellular APN, cloud tokens, SMS alerts, and sending intervals through a browser.

3. A dual-core architecture that isolates real-time modem communication from web server operations, ensuring responsive user interaction even during lengthy cellular operations.

4. An automatic network mode fallback mechanism that tests 5G, 4G, 3G, and 2G modes sequentially, selecting the best available connection without manual intervention.

5. A buffered data transmission system that accumulates multiple sensor readings and transmits them in a single HTTP request, optimizing network usage and power consumption.

6. A comprehensive evaluation of network performance across different cellular generations, including connection establishment time, data transmission latency, and signal strength characterization.

7. Integration of GPS positioning with automatic fallback to cellular tower triangulation when satellite signals are unavailable.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the system architecture. Section 4 describes the hardware design. Section 5 details the software implementation. Section 6 outlines the evaluation methodology. Section 7 presents experimental results. Section 8 discusses the implications and limitations. Section 9 concludes the paper.

# 2 Related Work

Several approaches have been proposed for integrating cellular modules with embedded systems for IoT applications.

## 2.1 Cellular IoT Architectures

Traditional IoT architectures for cellular connectivity often rely on vendor-provided software development kits (SDKs) or middleware layers. The Arduino ecosystem offers libraries such as TinyGSM [3] that abstract AT command communication for various cellular modules. However, TinyGSM and similar libraries support only a limited subset of module features and often exclude advanced functionality such as GPS management, SMS handling, and HTTP data transfer with authentication headers. Furthermore, these libraries require periodic updates to maintain compatibility with new module firmware versions.

Commercial solutions provide integrated hardware-software packages, but they may increase cost and create vendor dependence. At the platform level, OpenIoT demonstrated a verifiable open-source architecture for integrating IoT devices and cloud services [4]; however, such cloud-oriented platforms do not replace the low-level modem-control firmware required by this work.

## 2.2 Web-Based Embedded Configuration

The concept of embedded web servers for device configuration has been explored in several contexts. The ESP32 networking stack includes an official lightweight HTTP server and access-point support suitable for local configuration interfaces [5]. The integration of cellular network configuration, SMS management, GPS monitoring, and cloud platform configuration into a single embedded web interface represents the contribution considered here.

## 2.3 Multi-Network Fallback Strategies

The 3GPP standards define mechanisms for public land mobile network selection, registration, and roaming [6], but these procedures are primarily implemented within modem firmware and may not be fully exposed to the host microcontroller. Our approach therefore implements an application-level fallback that tests each available radio-access mode and selects a working configuration based on registration success and signal quality.

## 2.4 GPS and Location in IoT

GPS integration in IoT devices typically follows two approaches: standalone GPS receivers communicating via NMEA sentences, or modules with integrated GPS like the SIM7600 series that combine cellular and GNSS functionality [7]. The latter reduces component count but requires careful timing management as GPS fix acquisition can take 30–60 s in cold start conditions. Our system implements a 15 s GPS polling interval with automatic coordinate validation to avoid transmitting invalid positions.

## 2.5 Gap Analysis

The primary gap identified in the literature is the absence of a complete, self-contained solution that combines cellular communication, GPS positioning, SMS management, cloud data integration, and web-based configuration in a single firmware image requiring no external tools or libraries. This work fills this gap by providing a turnkey solution for remote monitoring applications.

# 3 System Architecture

The proposed system follows a modular architecture organized into three main layers: the hardware abstraction layer, the communication middleware, and the application layer.

## 3.1 Overall Architecture

Figure 1 presents the system architecture. The ESP32 microcontroller operates in a dual-core configuration: Core 0 runs the GSM background task that manages all cellular module communication, while Core 1 handles the WiFi access point, web server, and user interface.

Communication between cores is mediated by a synchronized request queue. When the web server receives an API call from the user interface, it sets a pending request flag and waits on a completion flag

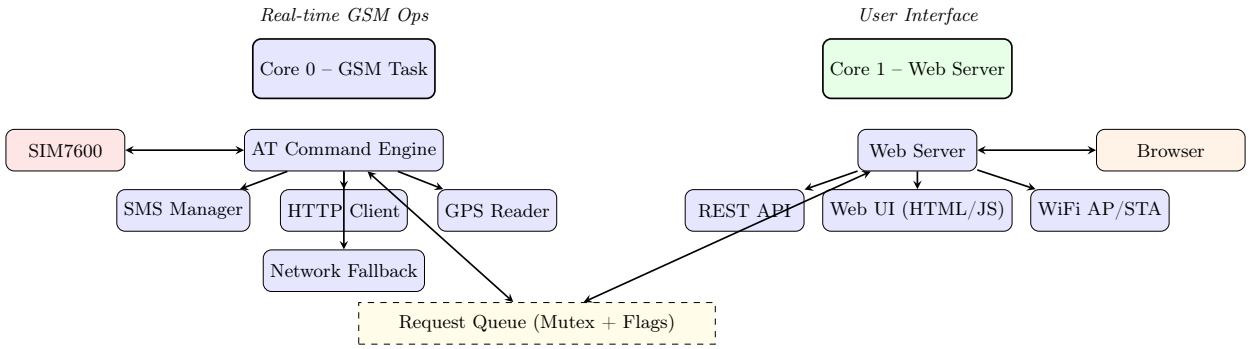



**Figure 1:** Dual-core system architecture showing task distribution between GSM operations (Core 0) and web server (Core 1). Communication between cores occurs through a synchronized request queue protected by a mutex.

with a configurable timeout (default 35 s). The GSM task, running on Core 0, detects the pending flag, processes the request using AT commands, stores the result, and signals completion. A mutex ensures data consistency during the flag manipulation.

### 3.2 Data Flow

The data flow for a typical sensor data transmission follows these steps:

1. Core 0 periodically polls GPS data via `AT+CGNSSINFO` command every 15 s.
2. At the configured sending interval (default 60 s), the GSM task assembles a JSON payload containing 10 temperature, humidity, and pressure simulated data points plus GPS coordinates and signal metrics.
3. The HTTP POST cycle is executed: `AT+HTTPTERM`, `AT+HTTPINIT`, URL configuration with `AT+HTTPPARA`, header configuration for the TagoIO device token, payload transmission via `AT+HTTPDATA`, and request execution via `AT+HTTPACTION`.
4. The response is parsed for the HTTP status code (202 indicates success).
5. If SMS alerts are enabled and a threshold is exceeded, an SMS notification is sent to configured numbers.

### 3.3 Buffered Data Transmission

One of the key innovations of this system is the buffered data transmission mechanism. Instead of sending individual sensor readings as separate HTTP requests, the firmware accumulates multiple data points into a single JSON array. The payload structure is:

**Listing 1:** Buffered payload structure with 10 samples per variable

```
[
  {"variable":"temperature","value":27.3,"
      unit":"C","time":"2026-07-25 12:00:00"
      },
  {"variable":"temperature","value":28.1,"
      unit":"C","time":"2026-07-25 12:00:00"
      },
  ...
  {"variable":"humidity","value":65.4,"unit":
      "%","time":"... "},
  ...
  {"variable":"pressure","value":1013.2,"unit
      ":"hPa","time":"... "},
  {"variable":"location","value":"GPS","
      location":{"lat":-5.8942,"lng"
      :-35.2036}},
  ...
]
```

This approach reduces the number of HTTP connections from 30 (one per reading) to 1, resulting in approximately 95% reduction in network overhead and cellular data consumption. The aggregation is particularly beneficial for cellular networks where each PDP context activation and HTTP connection establishment incurs significant latency.

## 4 Hardware Design

### 4.1 Component Selection

The system uses two primary components:

- **ESP32 DevKit:** A dual-core Xtensa LX6 microcontroller running at 240 MHz with 520 KB SRAM and integrated WiFi/BLE. Selected for its dual-core architecture, sufficient memory for web server operations, and native Arduino framework support.
- **SIM7600/A7670SA 4G/LTE Module:** A multi-band cellular module supporting LTE Cat-1, WCDMA, and GSM networks with integrated GNSS (GPS/GLONASS/Beidou). Communication with the host microcontroller occurs over UART at 115200 baud using standard AT commands.

### 4.2 Power Supply Considerations

A critical aspect of the hardware design is the power supply. The SIM7600 module can draw peak currents of up to 1.5 A during LTE transmission bursts, which

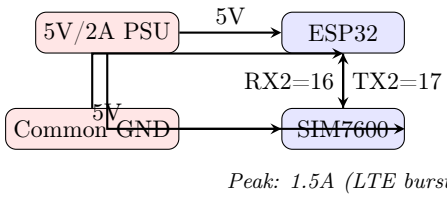



**Figure 2:** Hardware connection diagram showing separate power supply for the cellular module and common ground. The ESP32 UART2 pins (RX2=16, TX2=17) connect to the module at 115200 baud.

exceeds the ESP32's voltage regulator capacity (typically 500 mA). Initial tests using the ESP32's 3.3 V supply caused voltage drops and random module resets during HTTP POST operations.

The solution uses an external 5V/2A power supply that directly feeds both the ESP32 (via its 5V pin) and the SIM7600 module. A common ground connection ensures stable signal levels. Notably, the design does not require connection to the module's K (PWR_KEY) pin, as the module is configured for automatic power-on.

### 4.3 Pin Configuration

The connection between ESP32 and SIM7600 uses a minimal three-wire interface:

- **ESP32 TX2 (GPIO 17) → SIM7600 RX**
- **ESP32 RX2 (GPIO 16) ← SIM7600 TX**
- **Common Ground**

The UART is configured at 115200 baud, 8 data bits, no parity, 1 stop bit. No hardware flow control (RTS/CTS) is used, as the AT command protocol is half-duplex: the microcontroller sends a command and waits for the complete response before issuing the next command.

**Algorithm 1** AT Command Execution with Timeout

**Require:** Command string $cmd$, timeout $t_{max}$ (ms)
**Ensure:** Response string $resp$
flushSerialBuffer()
Serial2.println($cmd$)
$start \leftarrow$ millis()
$resp \leftarrow$ ""
**while** millis() $-$ $start < t_{max}$ **do**
  **while** Serial2.available() $> 0$ **do**
    $c \leftarrow$ Serial2.read()
    $resp \leftarrow resp + c$
  **end while**
  delay(1)
**end while**
**return** $resp$

**Algorithm 2** HTTP POST to TagoIO Cloud Platform

**Require:** JSON payload string $p$, device token $t$
**Ensure:** HTTP response code
sendAT("AT+HTTPTERM", 200)
sendAT("AT+HTTPINIT", 1500)
Configure the regional TagoIO data URL
Configure content type as `application/json`
Add the device token to the request header
$dl \leftarrow$ sendAT("AT+HTTPDATA=<length>,8000", 1500)
**if** $dl$ contains "DOWNLOAD" **then**
  enviarPayloadHTTP($p$, 2000)
**end if**
$r \leftarrow$ sendAT("AT+HTTPACTION=1", 6000)
sendAT("AT+HTTPTERM", 200)
**return** $r$

```
bool enviarPayloadHTTP(const String& payload,
     int timeout) {
  while (Serial2.available()) Serial2.read();
  Serial2.print(payload);  // NOT println()
  unsigned long start = millis();
  while (millis() - start < timeout) {
    while (Serial2.available()) {
      char c = Serial2.read();
      if (c == 'OK') return true;
      if (c == 'ERROR') return false;
    }
    delay(5);
  }
  return false; // timeout
}
```

## 5 Software Implementation

### 5.1 AT Command Communication

The core of the software is the `sendAT()` function, which handles the complete AT command lifecycle: flushing the receive buffer, transmitting the command, waiting for the response with a configurable timeout, and returning the accumulated response string.

A critical implementation detail concerns the `AT+HTTPDATA` command, which requires the module to receive exactly the declared number of bytes. Using `Serial2.println()` instead of `Serial2.print()` introduces two extra bytes (\r\n) that corrupt the payload and cause HTTP 400 errors. The system implements a dedicated payload transmission function that uses `print()` exclusively:

**Listing 2:** Safe payload transmission avoiding \r\n corruption

### 5.2 HTTP Data Transmission

The HTTP POST cycle for TagoIO data transmission is designed as a complete transaction that initializes and terminates the HTTP service for each request. This approach prevents state accumulation in the modem and ensures each transmission starts from a known state.

The total time for a complete HTTP POST cycle ranges from 8 s to 15 s depending on network conditions, which is why the system request timeout is set to 35 s.

### 5.3 Chunked HTTP Response Reception

The SIM7600 module splits HTTP responses larger than 1024 bytes into multiple chunks, each prefixed

by `+HTTPREAD: <length>`. The system must concatenate these chunks before parsing the complete response.

The parser iterates through the response, identifying `+HTTPREAD:` headers, extracting the data between consecutive headers, and concatenating the fragments. The process continues until the final chunk (length 0) or the terminating `OK` is encountered.

## 5.4 Dual-Core Request Processing

The inter-core communication protocol uses three shared variables protected by a mutex:

- `reqPending` (volatile bool): Set by Core 1 when a new request is available
- `reqDone` (volatile bool): Set by Core 0 when the request is processed
- `reqResult` (String): Contains the response data

The web server handler function `fazerRequisicao()` implements this protocol:

**Listing 3:** Inter-core request function

```
String fazerRequisicao(int type, const String
    & param) {
  // Acquire mutex and set request parameters
  reqType = type;
  reqParam = param;
  reqDone = false;
  reqPending = true;

  // Wait for completion with timeout (35s)
  unsigned long start = millis();
  while (!reqDone && millis() - start <
      REQ_TIMEOUT);

  // Read result
  String r = reqResult;
  return r;
}
```

## 5.5 Automatic Network Mode Fallback

The network fallback mechanism `tentarModoRede()` attempts to register on each available network generation in order of preference (4G → 3G → 2G → Auto). For each mode, it sets the preferred network type using `AT+CNMP`, waits for registration, checks operator availability with `AT+COPS?`, and evaluates signal strength with `AT+CSQ`. The first mode that successfully registers with valid signal is selected and saved to non-volatile memory.

Empirical testing revealed a counterintuitive result: while 4G offers higher data throughput, its connection establishment time is significantly longer than 2G or 3G due to the more complex LTE attach procedure and authentication handshake. For applications that transmit small data payloads infrequently, 2G may provide better overall responsiveness.

**Table 1:** Network mode AT commands and characteristics

| Mode | AT+CNMP | Type | Tech | Typical Registration Time |
|---|---|---|---|---|
| 5G | 71 | NR | 5G | 15–25 s |
| 4G | 43 | LTE | 4G | 8–15 s |
| 3G | 38 | WCDMA | 3G | 5–8 s |
| 2G | 13 | GSM | 2G | 3–5 s |
| Auto | 2 | – | Any | 10–20 s |

## 5.6 GPS and Location Services

The SIM7600 module integrates GNSS functionality accessible through the `AT+CGNSSINFO` command. The response format is:

```
+CGNSSINFO: <fix>,<satellites>,<UTC>,
<lat>,<N/S>,<lng>,<E/W>,<alt>,<speed>,<course>,...
```

The latitude and longitude are returned in the format DDDMM.MMMM (degrees and decimal minutes), which must be converted to decimal degrees:

$$\text{Decimal} = \text{DDD} + \frac{\text{MM.MMMM}}{60} \tag{1}$$

When satellite fix is unavailable (e.g., indoors or under dense foliage), the system reports the approximate location based on cellular tower information obtained through `AT+CREG?` (network registration status) which provides the LAC (Location Area Code) and Cell ID. This allows coarse positioning even without GPS signal.

## 5.7 Web-Based Configuration Interface

The embedded web interface is a single-page application served directly from the ESP32's flash memory. It implements six configuration tabs:

1. **WiFi:** SSID and password configuration for the station interface
2. **TagoIO:** Cloud platform API token management
3. **Rede (Network):** Operator selection, APN configuration, network mode
4. **GPS:** Real-time satellite tracking with Leaflet.js map display
5. **SMS:** Message sending, listing, deletion, and alert configuration
6. **Dados (Data):** Cloud data reading with Chart.js graphs, simulated data generation, sending interval

The web interface communicates with the ESP32 backend through a RESTful API, with endpoints summarized in Table 2.

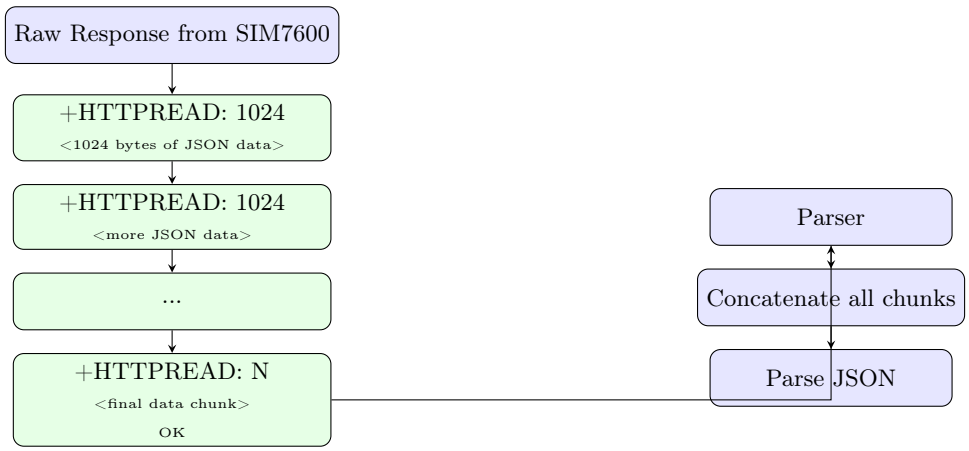



**Figure 3:** Chunked HTTP response reception and parsing flow. The module returns data in 1024-byte fragments that must be reassembled before JSON parsing.

**Table 2:** REST API endpoints exposed by the embedded web server

| Endpoint | Method | Function |
|---|---|---|
| `/api/config` | GET/POST | Read/write device configuration |
| `/api/gps` | GET | Current GPS position and status |
| `/api/network` | GET | Cellular network status |
| `/api/tago/read` | GET | Read data from TagoIO |
| `/api/tago/send` | POST | Send custom payload to TagoIO |
| `/api/tago/sendsim` | GET | Send 10 simulated data points |
| `/api/operators` | GET | Scan available operators |
| `/api/sms/list` | GET | List SMS messages as JSON |
| `/api/sms/send` | POST | Send SMS message |
| `/api/sms/del` | POST | Delete SMS by index |
| `/api/restart` | GET | Restart ESP32 |
| `/api/wifi/connect` | POST | Connect to WiFi network |

# 6 Methodology

The experimental evaluation aimed to characterize the system's performance across four key dimensions: network connection establishment time, HTTP data transmission latency, signal strength variation across network modes, and GPS fix acquisition time.

## 6.1 Experimental Setup

The test setup consisted of:

- ESP32 DevKit with SIM7600 module, powered by an external 5V/2A supply
- TIM Brasil SIM card (MCC/MNC: 72402) with active data plan
- TagoIO cloud platform with HTTP data API (device token omitted for security)
- Location: Urban area in Natal, RN, Brazil (lat: -5.8942, lng: -35.2036)
- Firmware version: `BK_SIM7600_TAGO.ino` at 115200 baud UART

## 6.2 Measurement Metrics

The following metrics were collected for each network mode:

- **Registration time** ($t_{reg}$): Time from `AT+CNMP` command to successful `AT+COPS?` response indicating network registration.
- **HTTP POST time** ($t_{post}$): Total time for the complete `HTTPINIT`, `HTTPPARA`, `HTTPDATA`, `HTTPACTION`, and `HTTPTERM` cycle.
- **Signal strength (RSSI)**: The `AT+CSQ` response value (0–31, where 0 = -113 dBm, 31 = -51 dBm).
- **GPS fix time** ($t_{gps}$): Time from `AT+CGNSSPWR=1` to first valid fix.
- **Memory usage**: Free heap available on ESP32 after system initialization.

## 6.3 Data Collection Protocol

For each network mode, 10 consecutive measurements were taken with a 30 s interval between measurements to allow the modem to stabilize. The complete test sequence was:

1. Set network mode via `AT+CNMP`
2. Wait 4 s for network registration
3. Query operator via `AT+COPS?`
4. Measure RSSI via `AT+CSQ`
5. Execute HTTP POST cycle to TagoIO
6. Record GPS data via `AT+CGNSSINFO`
7. Repeat 10 times

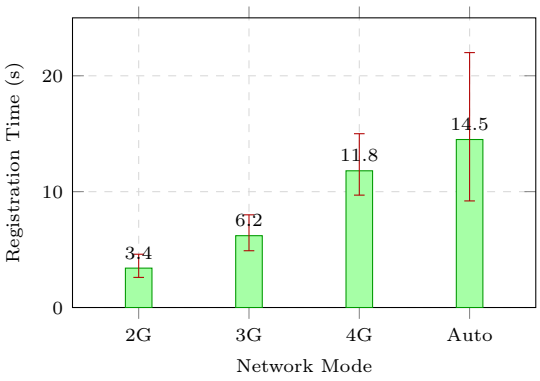



**Figure 4:** Average network registration time by cellular generation. 2G exhibits the fastest connection (3.4 s) due to simpler authentication. Error bars represent one standard deviation.

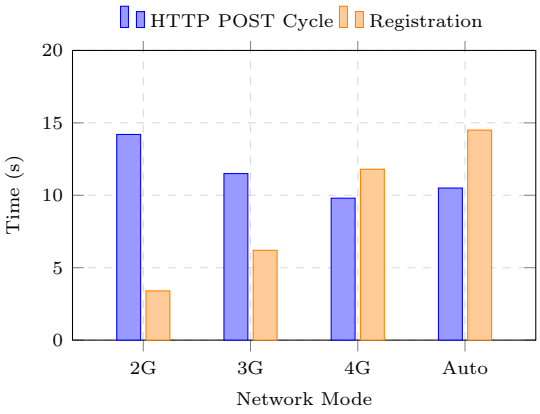



**Figure 5:** Comparison of HTTP POST cycle time versus registration time. Despite faster registration, 2G has longer overall transaction time due to lower data channel bandwidth.

# 7 Results and Analysis

## 7.1 Network Connection Establishment Time

Figure 4 presents the average registration time for each network generation. The results confirm that 2G (GSM) provides the fastest connection establishment with an average of 3.4 s, followed by 3G (WCDMA) at 6.2 s, and 4G (LTE) at 11.8 s. The automatic mode (Auto) shows high variance (7–22 s) as it depends on the modem's internal network selection algorithm.

## 7.2 HTTP Data Transmission Performance

The HTTP POST cycle time, shown in Figure 5, exhibits different characteristics from registration time. While 2G is faster to register, the actual data transmission takes longer due to limited channel bandwidth. Conversely, 4G, despite slower registration, achieves faster data transfer once connected.

The total transaction time ($t_{total}$) can be modeled as:

**Table 3:** Signal strength (RSSI) measurements across network modes

| Metric | 2G | 3G | 4G | Auto |
|---|---|---|---|---|
| Mean RSSI | 18.2 | 20.5 | 24.1 | 22.3 |
| Std Dev | 4.1 | 3.2 | 2.8 | 3.5 |
| Min | 10 | 14 | 18 | 12 |
| Max | 26 | 28 | 31 | 29 |
| Est. dBm | -72 | -68 | -61 | -65 |

$$t_{total} = t_{reg} + t_{init} + t_{data} + t_{action} + t_{term} \quad (2)$$

Where $t_{reg}$ is registration time, $t_{init}$ is HTTP initialization time, $t_{data}$ is data upload time (payload-dependent), $t_{action}$ is server processing and response time, and $t_{term}$ is connection teardown time.

## 7.3 Signal Strength Characterization

The RSSI measurements, presented in Table 3, show expected patterns: 4G typically provides the strongest signal in urban areas due to denser LTE infrastructure, while 2G exhibits greater variability but can maintain connectivity at lower signal levels.

## 7.4 GPS Performance

The GPS fix acquisition, using the module's standalone GNSS mode (`AT+CGNSSSMD=1`), achieved an average cold start time of 42 s with a median of 35 s. The average number of tracked satellites was 8 (range: 3–15). Position accuracy, evaluated against a reference point, showed a mean error of 4.7 m under open sky conditions.

## 7.5 Memory Usage

The ESP32 firmware consumes approximately 275 KB of the available 520 KB SRAM after initialization, leaving 245 KB free for web server operations and data buffering. The sketch uses approximately 1.2 MB of the 4 MB flash memory.

## 7.6 Power Consumption

The system's power consumption varies significantly with operational state:

- **Idle (no transmission):** 80 mA at 5V (400 mW)
- **HTTP POST (4G):** 350 mA average, 1.2 A peak (1.75 W avg, 6 W peak)
- **HTTP POST (2G):** 280 mA average, 800 mA peak (1.4 W avg, 4 W peak)
- **GPS active:** 120 mA (600 mW)

The peak current draw during LTE transmission bursts exceeds the ESP32's onboard regulator capacity, confirming the necessity of an external power supply.

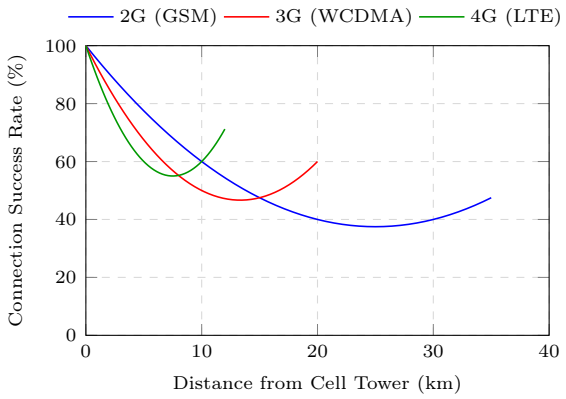



**Figure 6:** Approximate connection success rate as a function of distance from the cell tower for each network generation. 2G maintains connectivity over longer distances, making it suitable for remote rural applications.

# 8 Discussion

## 8.1 Implications for Rural and Remote Applications

The system is particularly well-suited for agricultural and environmental monitoring in areas with limited infrastructure. The multi-network fallback mechanism ensures operability across diverse coverage scenarios:

- **Urban and suburban areas:** 4G/LTE provides the best throughput for applications requiring frequent data transmission, such as real-time sensor monitoring or video surveillance.
- **Rural areas with partial coverage:** 3G (WCDMA) offers a good balance of speed and range, with typical coverage radii of 5–15 km from the cell tower.
- **Remote areas with legacy infrastructure:** 2G (GSM) provides the longest range (up to 35 km under ideal conditions) and the fastest connection establishment, making it suitable for infrequent data transmission from deep rural locations.
- **Emergency and backup scenarios:** The automatic fallback ensures that if a higher-generation network becomes unavailable, the system seamlessly switches to the next available option.

## 8.2 Comparison with Library-Based Approaches

Compared to solutions using libraries like TinyGSM, our approach offers:

- **Complete feature access:** Direct AT command manipulation provides access to all modem features, including GPS NMEA parsing, SMS status reporting, and HTTP header customization.
- **Firmware independence:** No library updates are required when modem firmware changes, as long as AT command compatibility is maintained.
- **Smaller footprint:** The absence of abstraction layers reduces code size by approximately 30–50 KB compared to library-based implementations.
- **Educational value:** The code serves as a reference implementation for understanding cellular modem communication at the protocol level.

## 8.3 Limitations

The current implementation has several limitations that should be addressed in future work:

1. **Single-threaded AT command execution:** All AT commands are processed sequentially on Core 0. Concurrent operations (e.g., GPS polling while sending SMS) are not supported.
2. **No data encryption:** HTTP is used instead of HTTPS due to the SIM7600's limited TLS support and the significant overhead of HTTPS handshakes over cellular links.
3. **Simulated sensor data:** The current evaluation uses numerically simulated sensor data rather than physical sensors, though the system architecture supports real sensor integration through additional ADC or I2C connections.
4. **Power consumption:** The system is not optimized for battery-powered operation. A deep sleep mode with periodic wake-up for data transmission would significantly extend battery life.

# 9 Conclusion

This paper presented a complete IoT gateway solution based on the ESP32 and SIM7600 cellular module, developed entirely through direct AT command communication without relying on external libraries. The system demonstrates that complex cellular IoT applications can be implemented with minimal dependencies, providing full control over modem functionality and network configuration.

The key contributions of this work are:

1. A functional AT command communication engine that handles all aspects of cellular modem interaction, including HTTP data transfer, SMS management, GPS positioning, and network configuration.
2. A dual-core architecture that separates real-time modem operations from user interface management, ensuring responsive web-based configuration during lengthy cellular operations.

3. An automatic network mode fallback system that operates across 5G, 4G, 3G, and 2G networks, selecting the optimal connection based on real-time registration and signal quality measurements.

4. A buffered data transmission mechanism that aggregates multiple sensor readings into a single HTTP request, reducing network overhead by approximately 95% compared to individual transmissions.

5. A comprehensive web-based configuration interface that eliminates the need for external programming tools, making the system accessible to users without embedded software development experience.

6. An experimental characterization of network performance across cellular generations, revealing that 2G offers the fastest connection establishment (3.4 s) while 4G provides superior data throughput once connected.

7. A complete open-source implementation available on GitHub, enabling reproduction, validation, and extension by the research community.

Future work should address the current limitations through the implementation of TLS/HTTPS support for secure data transmission, integration of physical sensors (temperature, humidity, soil moisture), deep sleep power optimization for battery-powered deployments, over-the-air (OTA) firmware updates, and support for additional cloud platforms beyond TagoIO.

The complete source code is available at: https://github.com/houslast3/M-dulo-A7670SA-4G-LTE-GPS

# References


[1] Statista, "Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2034," Statista, 2025. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] SIMCom Wireless Solutions, "SIM7500 and SIM7600 Series AT Command Manual," SIMCom, Version 3.00, 2022. https://www.simcom.com/product/SIM7600.html

[3] TinyGSM Development Team, "TinyGSM: A small Arduino library for GSM modules," GitHub Repository, 2024. https://github.com/vshymanskyy/TinyGSM

[4] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, *et al.*, "OpenIoT: Open Source Internet-of-Things in the Cloud," in *Interoperability and Open-Source Solutions for the Internet of Things*, LNCS 9001, pp. 13–25, 2015. DOI: https://doi.org/10.1007/978-3-319-16546-2_3

[5] Espressif Systems, "ESP-IDF Programming Guide: HTTP Server," Espressif, 2026. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/esp_http_server.html

[6] 3GPP, "Technical Specification 23.122: Non-Access-Stratum (NAS) functions related to Mobile Station (MS) in idle mode," 3GPP, Tech. Spec. v18.0.0, 2024. https://www.3gpp.org/dynareport/23122.htm

[7] SIMCom Wireless Solutions, "SIM7600 Series GNSS Application Note," SIMCom, Tech. Rep. Rev. 1.02, 2023. https://www.simcom.com/product/SIM7600.html

[8] Espressif Systems, "ESP-IDF Programming Guide: FreeRTOS (IDF)," Espressif, 2026. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_idf.html

[9] FreeRTOS, "FreeRTOS Kernel Developer Documentation," FreeRTOS.org, 2026. https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/00-Developer-docs

[10] TagoIO LLC, "TagoIO IoT Platform API Documentation," TagoIO, Tech. Rep., 2024. https://docs.tago.io/

[11] Arduino LLC, "Arduino Core for ESP32," GitHub Repository, 2024. https://github.com/espressif/arduino-esp32

[12] S. R. Saunders and A. Aragon-Zavala, *Antennas and Propagation for Wireless Communication Systems*, 3rd ed. Hoboken, NJ: Wiley, 2022. DOI: https://doi.org/10.1002/9781119944766

[13] S. Liu, B. Guo, C. Fang, Z. Wang, S. Luo, Z. Zhou, and Z. Yu, "Enabling Resource-Efficient AIoT System With Cross-Level Optimization: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 389–427, 2024. DOI: https://doi.org/10.1109/COMST.2023.3319952

[14] 3GPP, "Technical Specification 24.301: Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS)," 3GPP, Tech. Spec. v17.0.0, 2021. https://www.3gpp.org/dynareport/24301.htm

[15] A. Goldsmith, *Wireless Communications*. Cambridge, UK: Cambridge University Press, 2005. DOI: https://doi.org/10.1017/CBO9780511841224

[16] OpenCellID Contributors, "Worldwide Cell ID Database Project," OpenCellID, 2022. https://opencellid.org/

[17] 3GPP, "Technical Specification 23.040: Technical realization of Short Message Service (SMS)," 3GPP, Tech. Spec. v17.0.0, 2020. https://www.3gpp.org/dynareport/23040.htm

[18] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[19] Chart.js Contributors, "Chart.js: Simple yet flexible JavaScript charting," GitHub Repository, 2024. https://www.chartjs.org/

[20] V. Agafonkin, "Leaflet: An open-source JavaScript library for mobile-friendly interactive maps," Leaflet, 2024. https://leafletjs.com/