Proceedings of CONF-MLA 2025 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2025.LD27215

Programming Design of an STM32-Based Positioning Information Acquisition and Upload System

Wang Sen

Department of Mechanical Engineering, Tianjin University, Tianjin, China
3182539801@qq.com

Abstract. We design a low-power STM32F103-based GNSS terminal that prioritizes NB-IoT uplink and falls back to a LoRa-to-Ethernet gateway when cellular attach or TCP setup fails. An RTC-driven wake → acquire → package → upload → sleep loop with compact, newline-delimited JSON payloads reduces airtime and energy. Layered design and power gating extend lifetime, while the LLCC68/W5500 gateway transparently bridges bytes to a TCP server. Results show robust uploads under heterogeneous connectivity with minimal terminal complexity.

Keywords: STM32F103, NB-IoT, LoRa, AT6558R, LLCC68, W5500, JSON Lines, RTC, low-power, TCP

1. Introduction

With the rapid expansion of the IoT, field terminals must localize and report data reliably while operating on tight energy budgets and under intermittent connectivity. NB-IoT offers wide-area reach but can exhibit higher per-uplink energy and coverage gaps in practice [1]. LoRa provides very low-power links but requires a nearby gateway and careful payload sizing [2,3]. To reconcile these constraints, we adopt a dual-uplink architecture: a GNSS-enabled terminal that prioritizes NB-IoT for direct cloud upload and transparently falls back to a LoRa-to-Ethernet gateway when cellular service is unavailable. The design emphasizes RTC-scheduled duty cycling, compact JSON Lines messages for streaming [4], and power-domain control to enable long-life, unattended positioning at low cost.

DOI: 10.54254/2755-2721/2025.LD27215

2. System overview

Table 1. Overall architecture and topology

Subsystem / Item	Components / Steps	Purpose / Notes
Hardware system	Terminal + Gateway	Two-tier topology: a battery-powered terminal reports to a gateway for aggregation and backhaul.
Software workflow	1) Exit low-power → 2) Acquire GPS → 3) Upload data → 4) Enter low-power	Duty-cycled loop that minimizes energy use while ensuring periodic positioning and uplink.
Power-management system	(1) Ultra-long standby (2) MCU low-power modes	Long-life operation via aggressive sleep states and coordinated wake/sleep scheduling.

Table 2. Hardware architecture and key designs

Terminal	(1) Main MCU: STM32F103C8T6(2) GPS: AT6558R (5N32 package)(3) NB-IoT modem: QS100(4) LoRa transceiver: LLCC68	MCU orchestrates sensing, positioning, and communications; GPS provides position fix; NB-IoT offers cellular uplink; LoRa enables low-power local relay when needed.
Gateway	LLCC68 LoRa radio + W5500 Ethernet controller (W5500 is gateway-side only; the gateway transparently bridges bytes to the TCP server without parsing application semantics.)	Aggregates LoRa frames from terminals and forwards them to the server via Ethernet backhaul.

2.1. Main controller (STM32F103C8T6)

The STM32F103C8T6 (72 MHz; 64 KB Flash; 20 KB RAM) orchestrates GNSS, NB-IoT, and LoRa via UART/SPI/I²C. Designed for battery-powered deployments, the MCU spends most of its time in sleep and wakes on an RTC alarm or external interrupt to acquire a fix, package a record, upload, and return to sleep. Ethernet (W5500) is implemented on the gateway only; the terminal never interfaces with W5500. This separation simplifies the terminal's hardware, tightens its power budget, and clarifies responsibilities between terminal and gateway.

2.2. Software architecture and task orchestration

Application layer:Schedules a duty-cycled loop (wake–acquire–package–upload–sleep), selects NB-IoT with automatic LoRa fallback, and handles timeouts/retries/resume.Communication layer: Exposes two paths—UART→QS100→TCP (NB-IoT) and SPI→LLCC68→gateway→W5500→TCP (LoRa→Ethernet)—and provides NB-IoT attach & sockets, LoRa RF setup & framing, and gateway socket & forwarding.Driver layer: Initializes and power-sequences GNSS/LoRa/NB-IoT/W5500/RTC, with interrupts and buffer/DMA support.

Common services: Logging and timestamps, ID/UUID, integrity checks, parameter loading, and staging buffers.

DOI: 10.54254/2755-2721/2025.LD27215

2.3. Data model and communication protocols

2.3.1. Application-layer data model (JSON packaging)

To balance readability and backend parsing, a minimal JSON schema is used—only essential fields (latitude/longitude, hemisphere, timestamp, etc.) are transmitted. Field minimization reduces air-interface load and serial congestion. Each record is emitted as a JSON Line (newline-delimited JSON), with uuid + dateTime used for de-duplication.

"uuid": "device-unique-identifier",
"lat": 0.000000,
"lon": 0.000000,
"latDir": "N|S",
"lonDir": "E|W",
"dateTime": "YYYY-MM-DD HH:MM:SS" // UTC

2.4. Low-power strategy and energy budget

Strategies: peripheral power-gating on demand; MCU clock gating / frequency scaling; immediate sleep after task completion; GPS warm/hot start to shorten time-to-fix and reduce active duration.

3. Hardware implementation and software integration

3.1. System-level collaboration (terminal — gateway — server)

The terminal operates a duty-cycled loop—wake → acquire → package → upload → sleep—driven by the RTC. The state machine is illustrated in Figure 2. Payloads are emitted as newline-delimited JSON objects (JSON Lines) so that the server can stream-parse records regardless of LoRa frame or TCP segment boundaries [4]. The gateway performs byte-transparent forwarding from LoRa to a persistent TCP client without interpreting application semantics. This collaboration proved stable in field tests and keeps terminal hardware and firmware minimal.

3.2. Hardware-to-software mapping and timing

GPS (AT6558R):The terminal drives the AT6558R through a dedicated UART, powers it only during the acquisition window, and leverages hot/warm start to shorten time-to-fix. In low-power mode the GNSS rail is fully gated by the MCU; on wake, the firmware allows a short stabilization interval before reading NMEA frames into a staging buffer and packaging them as a JSON record. The start/backup behaviors and timing choices follow the vendor datasheet recommendations [5].

NB-IoT (QS100). Session setup follows the AT sequence summarized in Figure 3: check attachment (AT+CGATT? → +CGATT:1), create a TCP stream socket (AT+NSOCR=STREAM,6,0,0), then connect (AT+NSOCO=<id,ip,port>). On failure, apply retry-with-exponential-backoff; after the retry budget is exhausted, automatically switch to the LoRa path. Data-plane send/acknowledgment uses AT+NSOSD with a sequence ID and polling via AT+SEQUENCE.

DOI: 10.54254/2755-2721/2025.LD27215

```
// P01/interface/Int_QS100.c (excerpt)
Int_QS100_Send_Cmd("AT+CGATT?\r\n");
// Query attachment (+CGATT:1 means attached)

Int_QS100_Send_Cmd("AT+NSOCR=STREAM,6,0,0\r\n");
// Create TCP socket

sprintf((char*)cmd, "AT+NSOCO=%d,%s,%d\r\n", socket, ip, port);
// Connect

sprintf((char*)cmd, "AT+NSOSD=%d,%d,%s,0x200,%d\r\n",
    socket, len, hex_data, SEQUENCE);
// Send HEX (JSON payload)

sprintf((char*)cmd, "AT+NSOCL=%d\r\n", socket);
// Close socket
```

Figure 1. NB-IoT client setup with minimal AT commands

LoRa (LLCC68): The terminal connects an LLCC68-class transceiver (MS21SF1) over SPI and uses it only as a fallback path when the NB-IoT uplink is unavailable. LoRa PHY/MAC settings (channel plan, data rate, RX/TX windows) follow the LoRaWAN L2 1.0.4 specification, while the driver abstracts IRQ-driven reception into a simple "copy-to-app-buffer and signal-ready" primitive [6,7].

4. Software implementation (specific collaboration among terminal, gateway, and cloud)

4.1. Main workflow

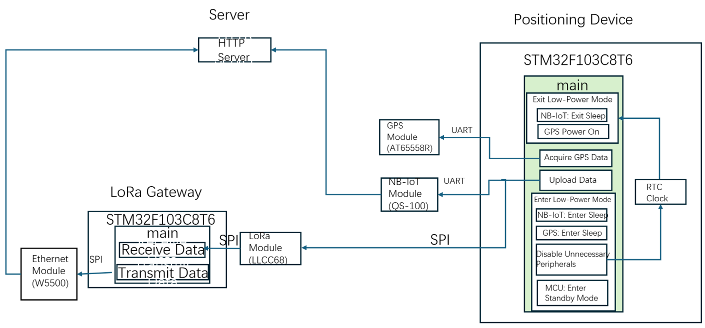

Figure 2. Terminal–LoRa gateway–server topology with NB-IoT primary, LoRa fallback

4.2. NB-IoT connection establishment and session initialization

We employ a blocking AT helper that writes a command, aggregates UART responses via HAL_UARTEx_ReceiveToIdle, and returns on OK/ERROR/timeout. The deployed QS100 firmware exposes the NSO* socket family (AT+NSOCR/NSOCO/NSOSD/NSORF/NSOCL). Client setup proceeds as: (1) verify attachment, (2) open stream socket, (3) connect. Treat only OK as success; otherwise close with AT+NSOCL to avoid half-open sockets. On repeated failures, back off and then hand over to the LoRa fallback. The resulting flow is depicted in Figure 3.

DOI: 10.54254/2755-2721/2025.LD27215

```
// Attach / Get IP check
CommonStatus Int_QS100_GetIP(void)
{
    uint8_t* cmd = "AT+CGATT?\r\n";          // Query attachment
    Int_QS100_Send_Cmd(cmd);
    if (strstr((char*)iot_full_buff, "+CGATT:1") != NULL) return COMMON_OK;
    return COMMON_ERROR;
}

// Create TCP socket, connect to server
CommonStatus Int_QS100_Create_Client(uint8_t* socket)
{
    uint8_t cmd[32];
    sprintf((char*)cmd, "AT+NSOCR=STREAM,6,0,0\r\n");   // Create TCP stream socket
    Int_QS100_Send_Cmd(cmd);

    // Parse response "+NSOCR:<id>"
    char* p = strstr((char*)iot_full_buff, "+NSOCR:");
    if (p == NULL) return COMMON_ERROR;
    *socket = (uint8_t)atoi(p + strlen("+NSOCR:"));
    return COMMON_OK;
}

CommonStatus Int_QS100_Connect_Server(uint8_t socket, uint8_t* ip, uint16_t port)
{
    uint8_t cmd[64];
    sprintf((char*)cmd, "AT+NSOCO=%d,%d.%d.%d.%d,%d\r\n",
        socket, ip[0], ip[1], ip[2], ip[3], port);
    Int_QS100_Send_Cmd(cmd);
    return (strstr((char*)iot_full_buff, "OK") ? COMMON_OK : COMMON_ERROR);
}
```

Figure 3. Blocking AT helper and response-parsing implementation

4.3. LoRa channel coordination: terminal fragmentation, gateway reception, ethernet forwarding

4.3.1. Terminal: 255-B fragments

Terminal fragmentation. Each record is sliced into ≤255-byte frames (leaving headroom for headers/CRC at the chosen modulation). The gateway forwards raw bytes immediately on reception. On the server, a record is reconstructed by concatenating bytes until a newline (\n) is observed, then parsing one JSON object—independent of LoRa frame or TCP segment boundaries [4]. This streaming-safe packaging is shown in Figure 4.

```
while (upload_data.dataLen > 255) {
    Int_LoRa_Send_Data(&(upload_data.data[tmp * 255]), 255);
    tmp++;
    upload_data.dataLen -= 255;
}
Int_LoRa_Send_Data(&(upload_data.data[tmp * 255]), upload_data.dataLen);
```

Figure 4. LoRa TX fragmentation: send payload in 255-byte chunks, then the remainder

4.3.2. Gateway: forward LoRa to Ethernet

Gateway and Ethernet forwarding. The LoRa gateway maintains a persistent TCP client and transmits only in the ESTABLISHED state. Ethernet is provided by a W5500 hardwired TCP/IP offload device; the firmware opens a socket, reconnects on drop, and forwards the application buffer verbatim without interpreting JSON or fragment semantics [8].

DOI: 10.54254/2755-2721/2025.LD27215

```c
uint8_t Int_LoRa_Receive_Data(uint8_t* pData, uint16_t* Size)
{
    if (llcc68_irq_handler(&gs_handle) != 0) { return 1; }   // Handle receive interrupt

    if (gs_handle.receive_len > 0)
    {
        memcpy(pData, gs_handle.receive_buf, gs_handle.receive_len);
        *Size = gs_handle.receive_len;
        pData[*Size] = '\0';
        // Later forwarded to Ethernet by App_GW.c
    }
}
```

Figure 5. Interrupt-driven LoRa RX: service IRQ, copy to app buffer, set size, null-terminate, forward

5. Experimental results

5.1. Data transmission via NB-IoT

We first established the TCP client and verified reachability of the server endpoint. After connecting, the terminal transmitted the payload and we confirmed reception on the TCP monitoring page.

5.2. Transmission via LoRa gateway

When the NB-IoT module failed to create or connect the client, the gateway fallback path was used. We re-validated end-to-end connectivity and confirmed that the payloads were forwarded to the server.

```
Log (English) – Socket Close & IoT Send Result
2
OK
----------------------------------------
[Int_QS100.c:62] AT+SEQUENCE=0,1
1
OK
----------------------------------------
[Int_QS100.c:62] AT+NSOCL=0      # close socket
*NSOCL:0
OK
----------------------------------------
[App_Location.c:144] IoT data transmission failed
[llcc68] irq tx done.
```

```
Log (English) – GNSS Fix & JSON Payload
$GNGGA,0,00,25.5,,,,,,*64
$GNGLL,,N,,,,,V*7A
$GNGSA,A,3,25,5,25,5,,,,,,1.01
$GNGSA,A,3,25,5,25,5,,,,,,1.01
$GPGSV,3,1,09,02,25,5,404
$GPGSV,3,2,09,05,25,5,404
$GPGSV,3,3,09,07,25,5,404
QS100, GT_01_01, ANTENNA OPEN×25
[App_Location.c:92] Detected valid positioning data
[App_Location.c:51] time: 203050, lat: 31.11136, latDir: N, lon: 121.225492, lonDir: E, date: 310125
[Com_Tool.c:51] [UTC time]: 1733584250
[Com_Tool.c:60] datetime: 2025-02-01 04:30:50
[App_Location.c:90] step: 0
[App_Location.c:90] json.str:
{"gwId":"670245456e4bfa906e08fff5","lat":"31.1113163333333","lon":"121.22549233333333","latDir":"N","lon
Dir":"E","dateTime":"2025-02-01 04:30:50","stepCount":0}
```

Figure 6. The result of system

6. Conclusions and outlook

The proposed dual-mode, RTC-duty-cycled design achieves a clear separation of concerns and robust, energy-efficient uploads across heterogeneous links. Future work will consolidate lightweight protocols, refine power domains, harden security/FOTA, and extend multi-GNSS positioning at fleet scale

References

[1] D. Yang et al., “Understanding Power Consumption of NB-IoT in the Wild, ” Proc. MobiCom, 2020.
[2] S. Ould et al., “Energy Performance Analysis and Modelling of LoRa, ” Sensors, 2021.
[3] L. Casals et al., “Modeling the Energy Performance of LoRa, ” Sensors, 2017.
[4] JSON Lines, “JSON Lines — Newline-Delimited JSON, ” n.d.
[5] Hangzhou Zhongke Microelectronics Co., Ltd., “AT6558R BDS/GNSS Satellite Positioning SoC Chip Datasheet, ” n.d.

DOI: 10.54254/2755-2721/2025.LD27215

[6] MinewSemi, “MS21SF1 (LLCC68/SX1262) LoRa Module Datasheet, ” 2024-06-06.
[7] LoRa Alliance, “LoRa® L2 1.0.4 Specification (TS001-1.0.4), ” 2020/2023.
[8] WIZnet, “W5500 Datasheet, ” v1.0.9, 2025-08-14.