

Internet of Things Engineering Practices

Course Report

Title : Intelligent Parking IOT Management System

Group number : 10

Name	Class	BUPT number
郭祎璇 GuoYiXuan	2022215115	2022213460
陈蔚 ChenWei	2022215116	2022213461
孙浩然 SunHaoRan	2022215116	2022213462

2025.12

1. Originality of Application System

1.1 Dual-Source Consistency Anti-Ticket-Evasion

By comparing real-time WSN occupancy statistics with RFID entry counts, the system auto-triggers an alarm whenever the two figures mismatch, ensuring the integrity of parking revenue.

1.2 RFID In-Card Data Loop

Key billing data “MAGIC + inMin + outMin + fee” are written into the card’s USER memory, cutting reliance on backend servers and keeping the exit workflow alive during weak or zero connectivity.

1.3 Three-State Parking-Slot Machine

Each slot is modeled as Free, Occupied, or Offline. A heartbeat-timeout instantly flags disconnected sensors, boosting maintainability and giving operators live visibility of every bay.

1.4 Sense-Decide-Act Edge Loop

Lux values captured by WSN nodes directly drive local light-control commands. When illuminance crosses the threshold, the edge gateway issues the lamp order without cloud involvement, achieving true edge control.

1.5 AI Travel Concierge

Fusing Gaode weather API and DeepSeek LLM, the module fetches departure-time forecasts and road-risk tips through a non-blocking async UI, upgrading the driver’s exit experience with zero wait.

2. Related technology

The intelligent parking IoT management system integrates multiple core IoT technologies to achieve full-link intelligence from data perception to service output. The key related technologies are elaborated as follows:

2.1 Wireless Sensor Network (WSN) Technology

WSN technology serves as the core of the system's perception layer, responsible for real-time collection of parking space status data. The system deploys WSN sensing nodes at each parking space, which periodically collect environmental illuminance (Lux) signals. Through threshold judgment and state machine logic, the nodes infer whether the parking space is occupied, idle, or offline. The key technical points include:

- **Low-power data transmission:** WSN nodes adopt low-power design and periodic

reporting mechanisms to ensure long-term stable operation without frequent battery replacement, adapting to the 24/7 working scenario of parking lots.

- **Frame structure design:** The data frame transmitted by nodes includes frame header, tail verification, node MAC address, illuminance value, and timestamp. Through strict frame verification, the reliability of data transmission is guaranteed, and interference from invalid data is avoided.
- **Heartbeat mechanism:** Nodes send heartbeat packets at fixed intervals. The management terminal judges whether the node is offline by detecting the timeout of heartbeat packets, realizing real-time monitoring of device status and improving the maintainability of the system.

2.2 Radio Frequency Identification (RFID) Technology

RFID technology undertakes the core functions of vehicle entry and exit identification, data storage, and closed-loop charging. The system uses UHF RFID readers/writers at entrances and exits to interact with RFID cards, and its technical application focuses on:

- **Dual-mode data interaction:** At the entrance, the reader/writer writes key data such as vehicle EPC code and entry time (inMin) into the USER memory area of the RFID card; at the exit, it reads the stored entry data, calculates the parking fee, and writes back the exit time (outMin) and fee amount (fee) to the card, forming an offline-verifiable data loop.
- **High-speed read/write response:** Optimize the communication protocol between the reader/writer and the card, ensuring that the RFID read/write response time is ≤ 500 milliseconds, meeting the requirement of fast passage at entrances and exits.
- **Anti-collision processing:** The RFID reader supports multi-tag anti-collision algorithms, which can accurately identify the target vehicle's RFID card even in scenarios with dense vehicles, avoiding misreading or missed reading.

2.3 Edge Computing Technology

The system adopts edge computing architecture, moving core business logic to the PC-side edge gateway for execution, which breaks the dependence on cloud servers and improves real-time performance and reliability:

- **Local decision-making:** The illuminance data collected by WSN nodes is directly processed by the edge gateway. When the illuminance crosses the set threshold, the gateway directly issues light control commands without cloud forwarding, realizing the "sense-decide-act" closed loop.
- **Offline operation support:** The key billing data is stored in both the RFID card and the local edge terminal. Even in weak network or offline environments, the system can complete fee calculation and barrier lifting based on the data in the card, ensuring the continuity of core services.
- **Lightweight resource scheduling:** The edge gateway reasonably allocates computing resources for tasks such as parking space state analysis, fee calculation, and equipment communication, avoiding resource competition and ensuring that the data processing delay is ≤ 2 seconds.

2.4 Serial Communication Technology

Serial communication is the key technology for data transmission between the system's management terminal and field devices, including WSN coordinators and RFID readers/writers:

- **Standardized communication parameters:** The system adopts a baud rate of 115200 bps, 8 data bits, 1 stop bit, and no parity check, ensuring compatibility with mainstream industrial serial devices.
- **Bidirectional data interaction:** The management terminal sends configuration commands (such as serial port connection, device initialization) to field devices through serial ports, and receives real-time data (such as WSN sensing data, RFID card swiping records) uploaded by devices, forming a complete data transmission link.
- **Exception handling mechanism:** The system monitors serial port connection status in real time. If a disconnection occurs, it automatically triggers reconnection and records error logs, ensuring the stability of device communication.

2.5 API Integration Technology

The system integrates third-party APIs to enrich intelligent service functions, mainly including:

- **Gaode Weather API:** Through HTTP GET requests, the system obtains real-time weather data (temperature, visibility, weather conditions) at the parking lot location, providing a data basis for AI travel recommendations.
- **DeepSeek LLM API:** Using HTTP POST requests, the system sends prompts combining weather, parking duration and other information to the LLM model, and obtains personalized travel suggestions (such as driving precautions, warmth tips), which improves the user's travel experience.
- **Asynchronous request processing:** To avoid blocking the UI interface, the system executes API request tasks in background threads, and updates the results to the interface after the request is completed, ensuring the fluency of user interaction.

2.6 Java Swing UI Development Technology

The system's application layer adopts Java Swing technology to build a PC-side visual management interface, which has the characteristics of strong interactivity and intuitive display:

- **Modular interface design:** The interface is divided into parking space visualization panel, serial port configuration area, RFID function operation area, log display panel and other modules, which clearly presents the system's core functions and operating status.
- **Real-time data rendering:** The UI interface updates the parking space status (displayed in different colors), equipment communication status, and alarm information in real time through event callbacks and periodic refresh mechanisms, ensuring that managers can grasp the parking lot situation in a timely manner.

3. System function

3.1 Requirement Analysis

3.1.1 Core Objectives

To address the pain points of small to medium-sized parking lots (such as campuses, industrial parks, shopping malls, and residential areas), including delayed parking space information, irregular charging, frequent ticket evasion, and poor departure experience, by achieving integrated management of parking space monitoring, closed-loop charging, anomaly alarm, and intelligent services.

3.1.2. Users and Scenarios

Users: Parking lot managers (efficient management), car owners (convenient parking), operators (compliant charging)

Scenarios: Small to medium-sized parking lots, focusing on scenarios with frequent vehicle traffic that require improved management efficiency and user experience.

3.1.3. Core Requirements

1. Parking Space Management

- ◆ Real-time perception of parking space status (available/occupied/offline), visualized on a large screen
- ◆ Support for querying parking space details (illumination value, device MAC, communication status)

2. Entry/Exit and Charging

- ◆ RFID card entry, automatically writes entry time and recommends available parking spaces
- ◆ RFID card exit, automatically calculates fees, writes back data, and lifts the barrier after payment
- ◆ Can rely on card data to complete the charging loop in weak or no network conditions

3. Anomaly Risk Control

- ◆ Comparison of WSN parking space occupancy and RFID entry counts, triggers ticket evasion alarm in case of anomalies
- ◆ Interception of repeated entries, invalid exits, etc., and provides prompts

4. Intelligent Services and Maintenance

- ◆ Departure push of weather, road conditions, and other AI travel suggestions
- ◆ Parking full (indication when parking is full), scrolling display of RFID operation logs
- ◆ Support for maintenance functions such as serial port configuration, device connection, etc.

3.1.4. Non-functional Requirements

- 1. Fast Response:** Data processing delay ≤ 2 seconds, RFID read/write response ≤ 500 milliseconds
- 2. High Reliability:** Available in case of network disconnection, device failure does not affect core business
- 3. Easy to Expand:** Support for subsequent integration of new hardware, expansion of charging strategies, and service functions.

3.2 System Function Description

3.2.1 Parking Space Monitoring

The system can monitor the status of parking Spaces, identify whether they are idle, occupied or offline, and display these information in real time through the large screen, so that parking managers and users can grasp the parking situation in time.



Fig.1 Detail Information of Parking Spaces (Occupied)



Fig.2 Detail Information of Parking Spaces (Idle)



Fig.3 Detail Information of Parking Spaces (Offline)

3.2.2 Admission of Vehicles

When the vehicle enters the market, the system will automatically write the admission time through the card operation, and recommend the appropriate parking space for the vehicle according to the parking status to guide the vehicle to park quickly.



Fig.4 Success Entry and Recommend Parking

3.2.3 Entrance of Vehicles

When the vehicle leaves the parking lot, the system will charge based on the card swipe and write the fee information back. After the payment is completed, the system controls to lift the barrier and allow the vehicle to leave, achieving the automated management of vehicle departure.



Fig.5 Success Exit and Costing

3.2.4 Risk Control Alarm

The system has the function of risk control alarm, which can effectively prevent fare evasion, intercept repeat admission, and send an alarm in time when there is an abnormal situation to ensure the normal operation of the parking lot.

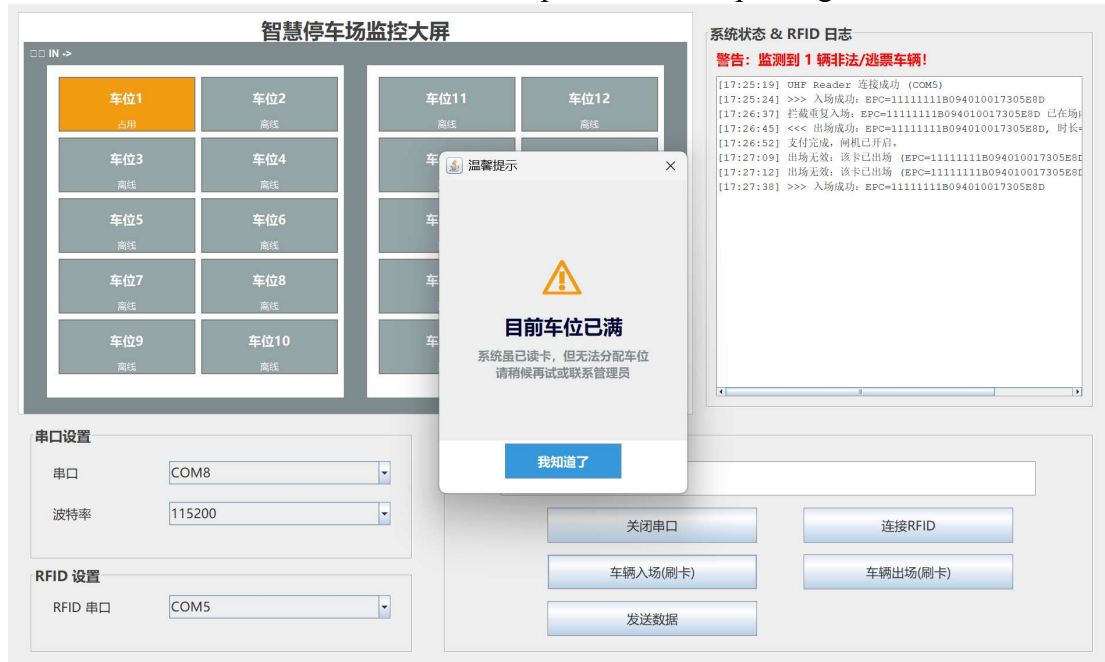


Fig.6 Parking Spaces All Occupied

3.2.5 Intelligent Services

It provides intelligent services to provide users with information such as weather and road condition risk tips when the vehicle leaves, and improves the user's travel experience with the help of AI technology.



Fig.7 AI Advice

3.2.6 Operation and Maintenance Assistance

The system provides assistance for the operation and maintenance work, and facilitates the maintenance and management of parking lots by the rolling display of RFID logs and the pop-up window function of parking details.



Fig.8 Lilegal Encroachment

4. Design and Implementation of system

4.1 Overall Design

This system targets real-world parking management needs in campus/industrial-park scenarios and is designed around five key directions. At the perception layer, the system integrates wireless sensing nodes (WSN) deployed at parking spaces with RFID readers at entrances/exits. At the management layer, it provides a PC-side visual interface and business logic processing, enabling core functions such as parking-occupancy detection, entry registration, exit billing, data write-back, and anomaly alarming, while also supporting extended services including weather queries and LLM-based travel suggestions.

4.1.1 Design Goals

1. **Parking-space monitoring and visualization:** Acquire parking-space occupancy status in near real time, manage a three-state model (Free / Occupied / Offline), and present results in a graphical interface to support driver guidance and operator scheduling.
2. **Closed-loop entry/exit billing:** Use RFID to write entry information, read it at exit, compute fees, and write back key records to the card to form a traceable and

verifiable business loop, reducing strong dependence on a central database.

3. **Risk control and consistency checking:** Perform dual-source consistency validation by comparing physical occupancy statistics (WSN) with entry-business counts (RFID), detecting potential ticket evasion or abnormal cases and triggering alarms and log records to improve reliability and security.
4. **Intelligent service extensibility:** Beyond the core parking workflow, provide interfaces for intelligent modules such as weather retrieval and LLM-based travel recommendations, enabling the system to evolve toward a “parking + travel service” platform.
5. **O&M and usability:** Offer O&M features including serial debugging, log output, and device status indication, ensuring the system can still provide interpretable states and prompts under node offline conditions or communication fluctuations, facilitating deployment, maintenance, and troubleshooting.

4.1.2 Design Constraints

1. **Hardware and communication constraints:** The system must be compatible with the given WSN nodes and RFID devices. Communication mainly relies on serial/gateway links with uncertain bandwidth and latency; therefore, the design must support frame validation, exception handling, and graceful degradation.
2. **Real-time constraints:** Parking-state updates and UI refresh should provide a near-real-time user experience. RFID swiping and gate-release procedures should be low-latency to avoid reducing traffic throughput.
3. **Reliability constraints:** The system should handle node offline, packet loss, and read/write failures by providing offline detection and error prompts, and ensure traceability through comprehensive logging.
4. **Engineering implementation constraints:** The PC client is implemented in Java Swing; concurrency between the UI thread and background tasks must be considered to prevent network calls (e.g., weather/LLM requests) from blocking the interface and degrading interaction stability.
5. **Extensibility constraints:** Modular interfaces should be preserved so that additional sensor types, billing strategies, or third-party services can be integrated smoothly without breaking the existing core workflow.

4.1.3 Four-Layer System Architecture

To improve maintainability, scalability, and engineering clarity, the system is designed using a layered architecture. The overall structure can be described as a four-layer architecture.

(1) Perception Layer

The perception layer is responsible for on-site data acquisition and actuator control in the parking lot. It mainly consists of:

- **WSN parking-space nodes:** Deployed at each parking space to collect environmental signals such as illuminance (lux). These measurements are used to infer parking occupancy and to determine whether a node is online/offline.
- **RFID readers/writers:** Installed at the entrance and exit to perform vehicle identification, write entry information, and read exit records, providing the basis for billing and gate release decisions.
- **Indicator light control:** Actuators that execute decisions made at the platform layer, such as red/green light indication, forming a closed-loop “sense–decide–act” workflow.

(2) Transmission Layer

The transmission layer ensures reliable data delivery from field devices to the edge/PC side and distributes control commands back to device endpoints. In this system, data aggregation and command delivery are mainly implemented via serial communication and gateway forwarding:

- WSN node data are aggregated by a **Coordinator** and then sent to the PC via a **serial** interface.
- RFID devices and controllers communicate with the PC through serial links (or equivalent channels) to support card reading/writing and gate control.

(3) Platform Layer (Edge Logic Layer)

The platform layer runs on the edge side (PC) and hosts the core business logic and decision-making, acting as the “brain” of the system. Its main responsibilities include:

- **Parking-state machine management:** Determine and update three states (Free / Occupied / Offline) based on WSN reports.
- **Billing and closed-loop workflow:** Handle entry writing, exit reading, fee calculation, and generation of key records that can be written back to the RFID card.
- **Risk control and consistency checking:** Compare physical occupancy statistics from WSN with entry counts from RFID to detect potential ticket evasion or abnormal events, then trigger alarms.

(4) Application Layer

The application layer provides a visual and interactive interface for operators, as well as O&M access points. It is implemented as a Java Swing PC client, including:

- **Parking-space dashboard visualization:** Intuitive display of parking-space states, summary statistics, and abnormal status prompts.
- **Logs and event panels:** Display RFID entry/exit records, WSN message reception, alarms, and runtime logs for traceability and troubleshooting.
- **Configuration and debugging panels:** Support serial port selection/connection, threshold parameter tuning, manual control, and device testing for maintenance purposes.

With this layered architecture, the system implements a complete end-to-end pipeline,

ensuring stable operation of the core parking workflow while leaving room for future extensions.

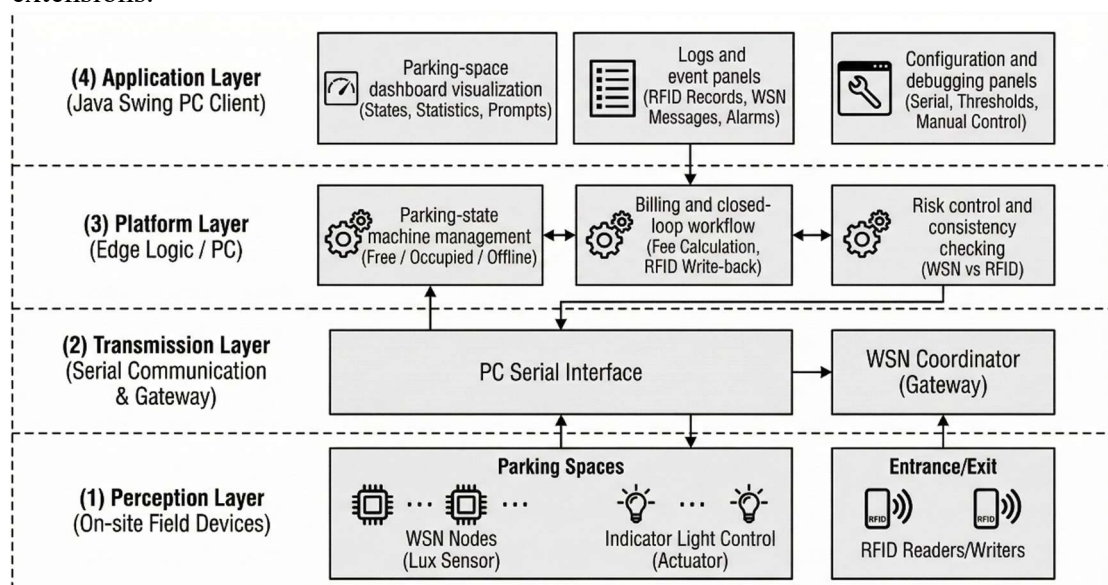


Fig. 9 Overall System Architecture of the Intelligent Parking IoT System

4.2 Physical Deployment

This system adopts a physical deployment scheme consisting of RFID entry/exit nodes, parking-space sensing nodes, and a management-side PC client.

4.2.1 Device List and Placement

The main devices and their typical placement are as follows:

1. RFID Entry/Exit Node (Deployed at the entrance and exit)

RFID reader/writer: Performs vehicle identification. At entry, it writes key entry data (vehicle identifier and timestamp) to the RFID card; at exit, it reads the entry data for validation and triggers the settlement workflow.

Settlement & release logic (controller/logic unit): After fee calculation on the management side, control commands are issued to determine whether to release the barrier (normal case) or block it (abnormal case).

Indicator lights: Provides red/green indication and status feedback to drivers.

2. Parking-Space Node (Deployed at each parking space)

WSN sensing node: Collects environmental signals such as illuminance (lux) to infer occupancy status. Nodes periodically report measurements to support both occupancy detection and online/offline determination.

3. Management Side (PC Client)

PC dashboard (Java Swing client): Provides parking visualization, event/log display, parameter configuration, and debugging interfaces. It serves as the core for business logic execution and system O&M.

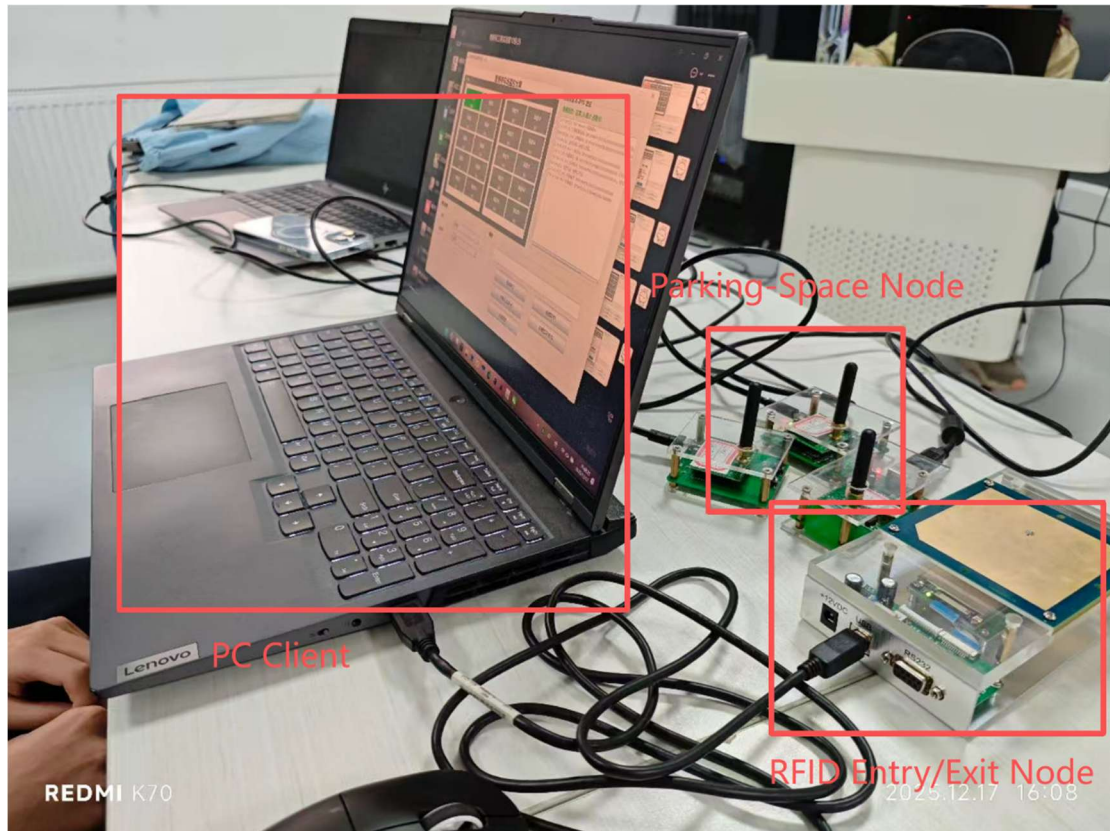


Fig. 10 Physical Deployment of Overall System Architecture

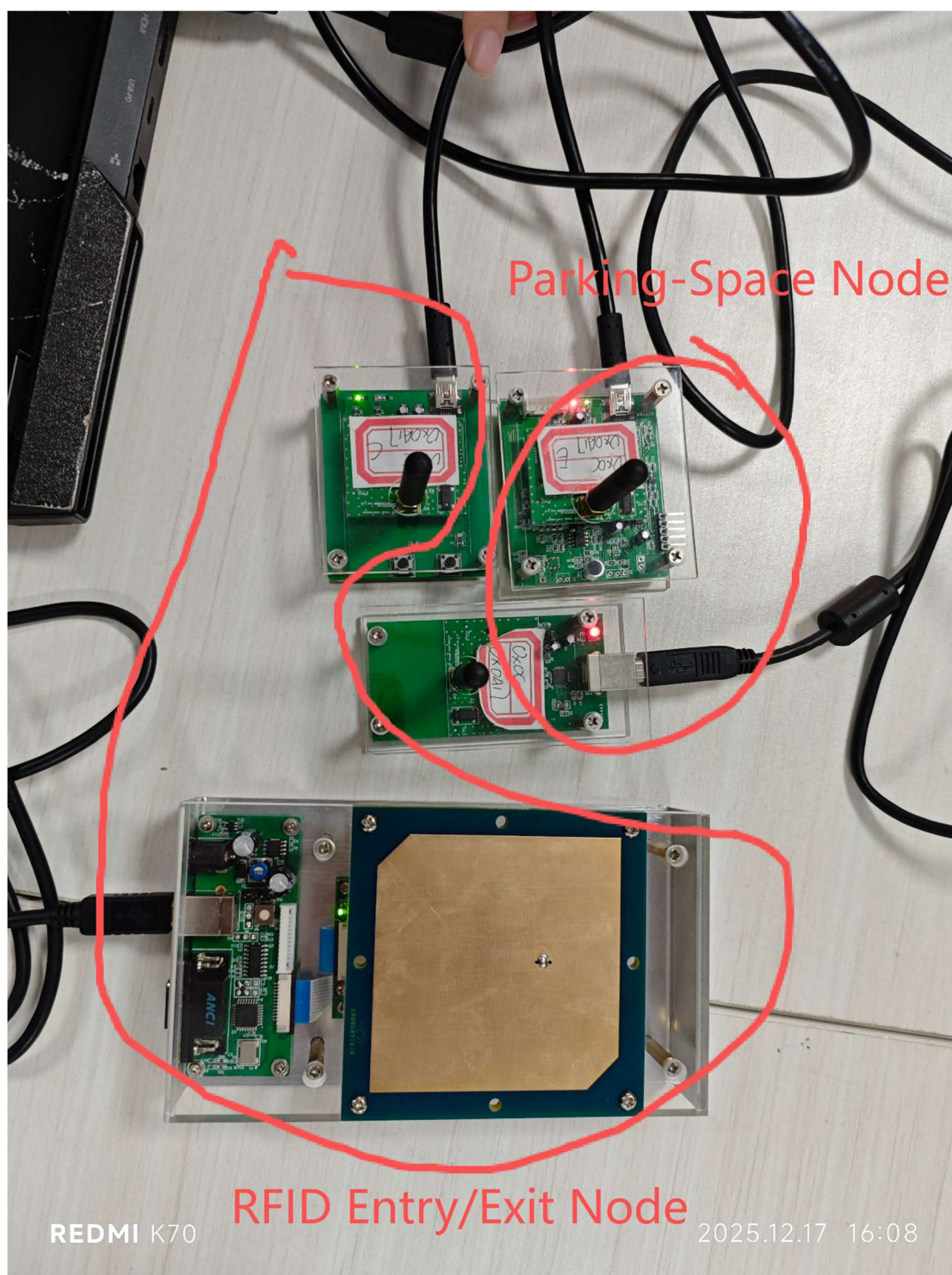


Fig. 11 Physical Deployment of RFID Entry/Exit Nodes and WSN Parking-Space Sensors

4.2.2 Data Flow and Control Flow

To clearly describe system operation, information exchange is divided into data flow (uplink) and control flow (downlink).

(1) WSN Data Flow (Parking-space status uplink)

WSN nodes periodically sample lux and report to the management side via a gateway/coordinator:

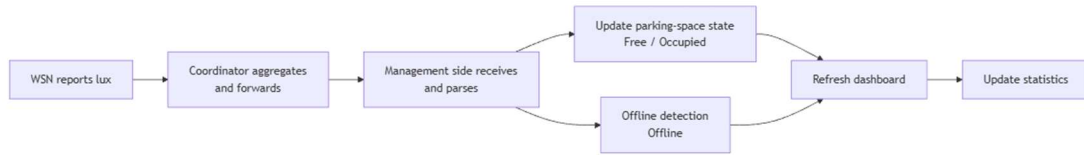


Fig.12 WSN Data Flow and Parking-Space State Update Process

In addition to updating occupancy states, the management side determines node online/offline status based on reporting intervals and a heartbeat timeout mechanism, ensuring that offline conditions caused by communication failures or power loss are reflected as an interpretable Offline state for O&M purposes.

(2) RFID Control Flow (Entry/exit business workflow: uplink + downlink)

The RFID workflow is event-driven and forms a closed loop: a swipe triggers processing (read/write/fee calculation), followed by execution (release/block).

Entry procedure:

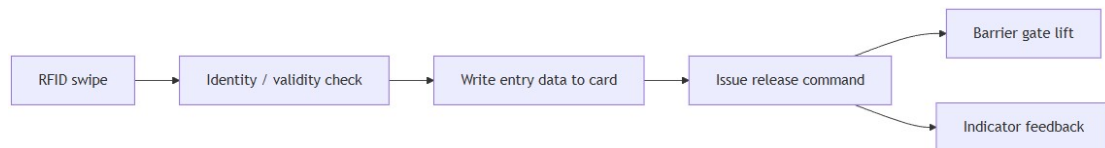


Fig.13 RFID-Based Entry Workflow

Exit procedure:

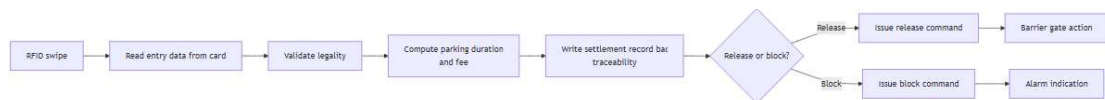


Fig.14 RFID-Based Exit Workflow

4.2.3 Edge Closed-Loop (Sense–Decide–Act)

The system embodies a typical Sense–Decide–Act edge-loop pattern at both parking-space sensing and entry/exit control. The key idea is to make decisions locally/near-field and execute actions immediately, reducing dependence on remote cloud processing and improving real-time responsiveness and robustness.

- **Sense:** WSN nodes measure lux (and related signals); RFID devices capture swipe events and card data.
- **Decide:** The management-side edge logic maps lux to parking states (Free/Occupied/Offline) via threshold rules and a state machine; it performs fee calculation and basic risk checks based on card data.
- **Act:** Control commands are sent to actuators for immediate execution, including barrier gate release/blocking, indicator feedback, alarms, and logging.

In particular, the occupancy decision based on an illuminance threshold (combined with offline timeout detection) is a representative edge strategy.

4.3 Module Design with Integrated Workflows

This system adopts a modular and layered software architecture, organized into the UI presentation layer, business logic layer, device adaptation/protocol parsing layer, and data model layer.

4.3.1 UI Presentation and Interaction Module (Java Swing)

The UI module provides visualization of parking-lot operation and **user interaction** entry points. The main window is implemented with a partitioned layout, including a **parking-space visualization panel**, **serial-port configuration** area, RFID function area, **operation** buttons, system **status** bar, and an RFID **log** panel. The parking panel supports inspecting per-slot details (Free/Occupied/Offline), lux value, node identifier, and last update time.

Integrated workflow (UI update pipeline):

- 1) The UI receives business-layer outputs (parking states, statistics, alarms) via periodic refresh and/or event callbacks.
- 2) When WSN reports or RFID events occur, the business layer updates the data model, and the UI maps the model to colors, labels, and logs to achieve near real-time display.
- 3) Time-consuming operations (e.g., weather/LLM requests) are executed in background threads to avoid blocking the Swing event thread, then UI updates are applied on the UI thread.

```
private void initComponents() { 1个用法
    Font baseFont = new Font( name: "Microsoft YaHei", Font.PLAIN, size: 22);
    // 1. 可视化面板
    parkingUI = new ParkingLotPanel();
    parkingUI.setSpotClickListener(( int block, int row, int col) -> {
        int spotId = block * 10 + row * 2 + col + 1;
        showSpotDetailDialog(spotId);
    });

    parkingScrollPane = new JScrollPane(parkingUI);
    parkingScrollPane.setBounds( x: 20, y: 20, width: 1160, height: 690);
    parkingScrollPane.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
    parkingScrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    parkingScrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER);
    add(parkingScrollPane);

    // 2. 串口设置
    serialPortPanel.setBorder(createBigTitleBorder("串口设置"));
    serialPortPanel.setFont(baseFont);
    serialPortPanel.setBounds( x: 40, y: 730, width: 660, height: 230);
    serialPortPanel.setLayout(null);
```

Fig.15 UI Partitioned Layout in the Java Swing Client

```

/**
 * 更新状态的核心方法
 * @param statusType 0=Free, 1=Occupied, 2=Offline
 */
public void updateSpotStatus(int block, int row, int col, int statusType) { 1个用法
    if (block < 2 && row < 5 && col < 2) {
        SwingUtilities.invokeLater(() -> {
            if(parkingSpots[block][row][col] instanceof ParkingSpotPanel) {
                ((ParkingSpotPanel) parkingSpots[block][row][col]).setStatus(statusType);
            }
        });
    }
}

```

Fig.16 Parking-Space Visualization with Three-State Rendering (Free / Occupied / Offline)

4.3.2 Device Access and Protocol Parsing Module (WSN / Serial / RFID)

The device adaptation layer is responsible for low-level communication and protocol parsing, ensuring that upper layers receive clean and usable structured data.

- **WSN frame parsing:** Validate incoming bytes using frame header/tail checks, parse short address/MAC, and extract the lux field as the input for occupancy inference.
- **Serial I/O:** Manage serial connections and listeners, and support sending control frames (e.g., FFFF...FEFE patterns) to drive field devices such as lights or barrier controllers.
- **RFID access:** Initialize the UHF reader, read EPC and user-memory content, and support user-memory read/write operations to enable entry writing, exit settlement, and data-loop closure.

```

private static WsnFrame parseWsnFrame(byte[] env) { 1个用法
    if (env == null || env.length < FRAME_LEN) return null;
    if ((env[0] & 0xFF) != 0xFF || (env[1] & 0xFF) != 0xFF) return null;
    if ((env[24] & 0xFF) != 0xFE || (env[25] & 0xFF) != 0xFE) return null;

    WsnFrame f = new WsnFrame();
    f.shortAddr = bytesToHex(env, off: 2, len: 2);
    f.mac = bytesToHex(env, off: 4, len: 8);
    f.lux = u16LE(env, off: 22);
    return f;
}

```

Fig.17 WSN Frame Parsing with Header/Tail Validation and Illuminance Extraction

```
private void sendData(java.awt.event.ActionEvent evt) { 1个用法
    String data = "FFFF" + dataInput.getText().toString() + "FEFE";
    System.out.println(data);
    try {
        SerialPortManager.sendToPort(serialport, ByteUtils.hexStr2Byte(data));
    } catch (Exception e) {
    }
}
```

Fig.18 Serial Control Frame Encapsulation and Transmission (FFFF...FEFE)

```
public ReadResult readEpcAndData(byte[] password, int area, int start, int len, 2个用法
    int perTryTimeoutMs, long totalTimeoutMs) {
    if (!inited) return new ReadResult( epcHex: "", dataHex: "", status: -1, rwStatus: -1);
    long t0 = System.currentTimeMillis();
    int status = -1;
    while (System.currentTimeMillis() - t0 < totalTimeoutMs) {
        RwData rw = new RwData();
        status = Linkage.getInstance().readTagSync(password, area, start, len, perTryTimeoutMs, rw);

        if (status == 0 && rw.status == 0) {
            String data = (rw.rwDataLen > 0) ? StringUtils.byteToHexString(rw.rwData, rw.rwDataLen) : "";
            String epc = (rw.epcLen > 0) ? StringUtils.byteToHexString(rw.epc, rw.epcLen) : "";
            return new ReadResult(epc, data, status, rw.status);
        }

        // 小睡一下，避免硬轮询打满 CPU/串口
        try { Thread.sleep( millis: 30); } catch (InterruptedException ignored) {}
    }
    return new ReadResult( epcHex: "", dataHex: "", status, rwStatus: -1);
}
```

Fig.19 Read Data from RFID Tag

```
// 写 data (用于把停车信息写进 USER 区)
public boolean writeData(byte[] password, int area, int start, byte[] dataBytes, 2个用法
    int perTryTimeoutMs, long totalTimeoutMs) {
    if (!inited) return false;
    // writeTagSync 的 len 参数一般是 word 长度，所以 byte 长度必须是偶数
    if (dataBytes == null || dataBytes.length == 0 || (dataBytes.length % 2 != 0)) return false;
    int lenWords = dataBytes.length / 2;
    long t0 = System.currentTimeMillis();
    int status = -1;
    while (System.currentTimeMillis() - t0 < totalTimeoutMs) {
        RwData rw = new RwData();
        // 这里用你库的 writeTagSync (你的 Demo 里肯定有)
        status = Linkage.getInstance().writeTagSync(password, area, start, lenWords, dataBytes, perTryTimeoutMs,
        if (status == 0 && rw.status == 0) return true;
        try { Thread.sleep( millis: 30); } catch (InterruptedException ignored) {}
    }
    return false;
}
```

Fig.20 Write Data to RFID Tag

4.3.3 Parking-Space State Machine Module (Free / Occupied / Offline)

The business layer introduces a parking-space state machine to manage three states consistently:

- **Free / Occupied:** Determined by lux threshold rules.
- **Offline:** Determined by heartbeat timeout (e.g., lastSeen compared against OFFLINE_MS). The output of this module is used not only for UI visualization but also as the foundation for space recommendation, statistics, and risk-control reconciliation.

Integrated workflow (parking-state update):

- 1) The device layer parses (MAC, lux, timestamp).
- 2) The state machine updates the corresponding slot's lastSeen and lux.
- 3) Apply threshold rules to infer Free/Occupied.
- 4) A periodic task checks lastSeen timeout and marks Offline when needed.
- 5) Output the three-state results and statistics to the UI for rendering.

```
private void updateSlotByMac(String macAddr, double luxVal) { 1个用法
    String mac = normalizeAddr(macAddr);
    Integer idx = macToSlot.get(mac);
    if (idx != null) {
        Slot s = slots[idx];
        s.lastLux = luxVal;
        s.lastSeenMs = System.currentTimeMillis();
        s.status = (luxVal < LUX_THRESHOLD) ? SlotStatus.OCCUPIED : SlotStatus.FREE;
    }
}
```

```
private void refreshSlotPanelText() { 2个用法
    int occ = 0, free = 0, off = 0;
    for (int i = 0; i < slots.length; i++) {
        Slot s = slots[i];
        if (s.lastSeenMs == 0 || (now - s.lastSeenMs) > OFFLINE_MS) {
            s.status = SlotStatus.OFFLINE;
        }
        switch (s.status) {
            case OCCUPIED:
                occ++;
                break;
            case FREE:
                free++;
                break;
            case OFFLINE:
                off++;
                break;
        }
    }
}
```

Fig. 20 Three-State Parking-Slot State Machine with Offline Timeout and Statistics Aggregation

4.3.4 Billing and In-Card Data Loop Module

This module computes parking fees and writes key business records back to the RFID card's user memory to form an offline-verifiable closed loop:

- At entry, write entry information such as vehicle identifier and entry timestamp (or time in minutes).
- At exit, read entry records → compute duration and fee → write back exit records and fee. The pricing policy is configurable (e.g., per-minute billing with a minimum of 1 minute), ensuring interpretability and ease of extension.

```
private static byte[] packInfo(int inMin, int outMin, int feeCents) { 2个用法
    byte[] b = new byte[12];
    UhfReaderService.putShortBE(b, off: 0, MAGIC);
    UhfReaderService.putIntBE(b, off: 2, inMin);
    UhfReaderService.putIntBE(b, off: 6, outMin);
    UhfReaderService.putShortBE(b, off: 10, feeCents);
    return b;
}

private static ParkInfo unpackInfo(byte[] b) { 2个用法
    ParkInfo p = new ParkInfo();
    if (b == null || b.length < 12) return p;
    p.magic = UhfReaderService.getShortBE(b, off: 0);
    p.inMin = UhfReaderService.getIntBE(b, off: 2);
    p.outMin = UhfReaderService.getIntBE(b, off: 6);
    p.feeCents = UhfReaderService.getShortBE(b, off: 10);
    return p;
}

private static int feeCentsByMinutes(int minutes) { 1个用法
    if (minutes < 1) minutes = 1;
    double fee = minutes * PRICE_PER_MIN;
    return (int) Math.round(fee * 100.0);
}
```

Fig. 21 RFID In-Card Data Packing, Unpacking, and Fee Computation Logic

```

private void actionListener() { 1个用法
    btnEntry.addActionListener( ActionEvent e -> {
        new javax.swing.SwingWorker<ReadResult, Void>() {
            protected void done() {
                int inMin = nowMin();
                byte[] toWrite = packInfo(inMin, outMin: 0, feeCents: 0); // 写入入场时间

                // 写入数据
                if (uhf.writeData(new byte[]{0, 0, 0, 0}, DATA_AREA, DATA_START, toWrite, perTryTimeoutMs: 3000, totalT

                    logRfid( msg: ">>> 入场成功: EPC=" + rr.epcHex);
                    rfidInCount++;
            }
        }
    }
}

```

Fig. 22 RFID Entry Workflow

```

private void actionListener() { 1个用法
    btnExit.addActionListener( ActionEvent e -> {
        new javax.swing.SwingWorker<ReadResult, Void>() {
            protected void done() {
                }

                int outMin = nowMin();
                int totalMinutes = Math.max(1, outMin - info.inMin);
                int fee = feeCentsByMinutes(totalMinutes); // 计算费用 (分)
                double feeYuan = fee / 100.0; // 转换为元

                // 计算人性化的时长显示 (例如 125分钟 -> 2小时 5分钟)
                int hours = totalMinutes / 60;
                int mins = totalMinutes % 60;
                String durationStr = (hours > 0 ? hours + "小时 " : "") + mins + "分钟";

                // 写入出场信息 (写入出场时间和费用)
                byte[] toWrite = packInfo(info.inMin, outMin, fee);

                if (uhf.writeData(new byte[]{0, 0, 0, 0}, DATA_AREA, DATA_START, toWrite, perTryTimeoutMs: 3000, t
            }
        }
    }
}

```

Fig. 23 RFID Exit Workflow with In-Card Billing

4.3.5 Risk Control and Consistency Verification Module

A lightweight yet effective consistency reconciliation mechanism is adopted:

- The WSN side provides the physical occupancy count (occ) derived from the state machine statistics.
- The RFID side provides the entry count (rfidInCount) derived from business events. If $occ > rfidInCount$, the system flags potential risks such as ticket evasion, missing swipes, or device anomalies, and triggers UI alarms and logging to assist operators in rapid diagnosis.

Integrated workflow (risk-control alarming):

- 1) Periodically compute occ/free/offline statistics.
- 2) Obtain rfidInCount.
- 3) Compare and compute the difference.

- 4) On anomaly: update alarm labels, logs, and visual highlights; otherwise keep normal status display.

```
// 核心逻辑：防逃票报警
// 逻辑：如果物理占用的车位(occ) 大于 系统记录的入场数(rfidInCount)，说明有人没刷卡就停进去了
if (occ > rfidInCount) {
    int diff = occ - rfidInCount;
    alarmLabel.setText("警告: 监测到 " + diff + " 辆非法/逃票车辆!");
    alarmLabel.setForeground(Color.RED);
} else {
    alarmLabel.setText("系统状态: 正常 (入场:" + rfidInCount + " 占用:" + occ + ")");
    alarmLabel.setForeground(new Color(r: 46, g: 204, b: 113)); // 绿色
}
}
```

Fig. 24 RFID Anti-Ticket-Evasion Logic

4.3.6 Intelligent Travel Service Module (Weather + LLM)

After completing the core parking workflow, the system provides an extended intelligent service module combining weather data and an LLM. The weather module retrieves real-time/forecast data via HTTP GET. The LLM module sends an HTTP POST request to the DeepSeek API to generate brief travel/driving suggestions.

Integrated workflow (non-blocking recommendation):

- 1) Trigger (typically after exit settlement completes).
- 2) Show a “loading” dialog.
- 3) Fetch weather and construct the prompt in a background thread.
- 4) Call the LLM to obtain suggestions.
- 5) Switch back to the UI thread, close the loading dialog, and display the final recommendation.

```
/**
 * 简单的 HTTP GET 请求 (用于高德天气)
 */
private String httpGet(String urlStr) { 1个用法
    try {
        java.net.URL url = new java.net.URL(urlStr);
        java.net.HttpURLConnection conn = (java.net.HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setConnectTimeout(5000);
        if (conn.getResponseCode() == 200) {
            try (java.util.Scanner s = new java.util.Scanner(conn.getInputStream(), charsetName: "UTF-8").useDelimiter("\\A")) {
                return s.hasNext() ? s.next() : "";
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "";
}
```

Fig. 25 Non-Blocking Intelligent Travel Service Pipeline Based on Weather API

```
private String httpPostDeepSeek(String prompt) { 1个用法
    try {
        java.net.URL url = new java.net.URL(spec: "https://api.deepseek.com/chat/completions");
        java.net.HttpURLConnection conn = (java.net.HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setRequestProperty("Authorization", "Bearer " + DEEPSEEK_API_KEY);
        conn.setDoOutput(true);

        // 构建 JSON 字符串 (手动拼接, 避免依赖库)
        String jsonBody = "{"
            + "\"model\": \"deepseek-chat\","
            + "\"messages\": ["
            + "  {\"role\": \"user\", \"content\": \"" + prompt + "\"}"
            + "]"
            + "}";

        try (java.io.OutputStream os = conn.getOutputStream()) {
            byte[] input = jsonBody.getBytes(charsetName: "UTF-8");
            os.write(input, off: 0, input.length);
        }
    }
}
```

Fig. 26 Non-Blocking Intelligent Travel Service Pipeline Based on LLM

5. Extension of the system

5.1 Functional Extension

5.1.1 Upgraded Parking Space Recommendation Strategy

The current parking space recommendation only supports the "first idle parking space" strategy. In the future, it can be upgraded to a multi-factor intelligent recommendation mechanism: Integrate distance factors: recommend the nearest parking space to the entrance to reduce the time for car owners to find parking spaces; Support appointment functions: open online parking space appointment, reserve designated parking spaces for users in advance, and lock the parking space state during the appointment period; Adapt to special parking spaces: identify disabled parking spaces, parent-child parking spaces, electric vehicle charging parking spaces, etc., and recommend them according to the user's vehicle type and needs.

5.1.2 Configurable Charging Strategy

The current charging logic is fixed (0.5 yuan/minute). In the future, a configurable charging management module can be added: Support setting of cap price: limit the maximum parking fee per day to avoid excessive fees for long-term parking; Implement peak-valley pricing: set different charging standards for peak periods (such as

commuting hours, weekends) and off-peak periods to guide vehicle flow and balance parking space usage; Add membership discount functions: launch membership cards, support prepaid recharge, discount parking fees, and improve user stickiness.

5.1.3 Rich Intelligent Service Functions.

On the basis of the existing AI travel guide, expand more user-oriented intelligent services: Real-time traffic information push: integrate traffic data interfaces to inform car owners of road congestion, construction, and accident information near the parking lot, and recommend the optimal driving route; Parking space reverse search: add a Bluetooth or GPS-based reverse search function, and guide car owners to find their parked vehicles through the mobile app; Electric vehicle charging management: for charging parking spaces, support real-time monitoring of charging status (such as charging progress, remaining time), and push charging completion reminders.

5.2 Technical Extension.

5.2.1 Platformization and Multi-Parking Lot Management

Expand from a single parking lot management to a multi-parking lot integrated management platform: Realize centralized monitoring: integrate the data of multiple parking lots into a unified background, and display the number of idle parking spaces, occupancy rate, charging amount, and other data of each parking lot on the large screen; Support unified operation and maintenance: manage the equipment of multiple parking lots in a unified manner, monitor the working status of WSN nodes and RFID devices in real time, and send fault alarms to operation and maintenance personnel; Realize cross-parking lot services: support users to query the idle status of multiple parking lots online, make appointments, and pay uniformly, improving the convenience of travel.

5.2.2 Access to Mobile Terminal

Develop supporting mobile apps or mini-programs to realize the mobileization of parking services: Mobile entry and exit: support Bluetooth, QR code, license plate recognition and other entry and exit methods, replacing traditional RFID card swiping, and realizing "no sense entry and exit"; Online payment: support WeChat, Alipay and other mobile payment methods, and complete parking fee payment through the app without stopping at the exit; Remote management: open the management background on the mobile terminal, allowing managers to view parking lot data, handle alarms, and configure parameters anytime and anywhere.

5.2.3 WSN Network Scaling and Routing Enhancement

Due to hardware constraints in the current prototype, the system was validated with **only one WSN end node** during testing. As a result, the present deployment mainly demonstrates the feasibility of **single-node sensing, frame parsing, and state updating**, but does not fully reflect multi-node networking performance under a real parking-lot scale. In future iterations, the WSN subsystem can be extended to a **multi-**

node, multi-hop networking architecture:

- **Add more end nodes (End Devices):** Deploy multiple sensing end devices across different parking spaces or zones. Each node periodically reports illuminance (lux) data, enabling full-lot occupancy sensing and improving spatial coverage.
- **Introduce router nodes for routing (Routers):** Add ZigBee/WSN router nodes to support **multi-hop forwarding**, expanding communication range and improving connectivity in environments with obstacles (e.g., underground garages, concrete walls).
- **Form a mesh topology:** With routers, the network can evolve from a single-link setup to **mesh** topologies, improving scalability and resilience. If a link becomes unstable, routing can provide alternative paths to maintain data delivery.

5.3 Application Scenario Extension

The current system is mainly aimed at small and medium-sized parking lots. In the future, it can be adapted to larger-scale application scenarios through technical optimization: Large commercial complexes: for parking lots with hundreds or thousands of parking spaces, optimize WSN node networking and data transmission mechanisms to improve the real-time performance and reliability of data.

6. Analysis of Internet of Things technology

Based on the design and implementation of the Intelligent Parking IoT Management System, this chapter analyzes the core meaning of IoT technology, its scenario-specific characteristics in parking management, and major development trends. The discussion emphasizes how “perception–transmission–processing–application” is concretely implemented in this project, and how engineering decisions (edge-side logic, offline robustness, and multi-technology integration) translate into practical value for real users.

6.1 Core Connotation of IoT Technology in the System

In essence, IoT is a “connected-and-intelligent” system that links physical entities to digital services through a full pipeline: sensing, communication, computation, and application delivery. In this parking system, the sensing layer is realized through a combination of WSN and RFID, enabling both “thing–thing” and “human/vehicle–thing” interaction. WSN end nodes measure environmental illuminance (lux) to infer whether a parking bay is occupied, while RFID readers identify vehicles and provide the trigger for entry/exit business operations. This multi-source sensing design is the foundation that makes the system measurable and controllable in the physical world.

The transmission layer connects distributed devices to the management side through serial links and gateway/coordinator forwarding. WSN data is aggregated by the coordinator and delivered to the edge management terminal, while control commands are issued from the management side back to RFID devices and indicators, forming a

bidirectional channel. In parking scenarios, the transmission layer is not “just communication”; its stability and latency directly affect whether vehicles can pass efficiently and whether the displayed occupancy status remains trustworthy.

The processing layer is where raw sensor readings become actionable decisions. Instead of forwarding everything to a cloud backend, this system adopts edge-side business logic to perform thresholding, state transitions, fee computation, and risk checks locally. For example, lux values are mapped to parking states via a three-state model (Free/Occupied/Offline), and RFID card data is converted into a fee through time-based calculation. This “data-to-information” conversion is the core of IoT intelligence, because it is the step that makes sensing meaningful and makes control possible.

Finally, the application layer turns processed information into user-facing services. For managers and operators, it provides visual monitoring, abnormal alarms, and log traceability for daily operation. For drivers, it supports smoother exit experiences and value-added services such as AI travel suggestions. In other words, IoT’s value is ultimately realized only when the system can reliably deliver services that reduce friction and solve real operational pain points.

6.2 Application Characteristics of IoT Technology in Parking

Scenarios

A key characteristic of IoT in parking is the need for multi-technology synergy rather than reliance on a single technique. This project integrates WSN sensing, RFID identification, edge-side computing, and external API/LLM integration into one coherent workflow. For instance, WSN-derived occupancy supports parking space recommendation and availability monitoring, while RFID enables a closed-loop charging process by reading/writing key records to card memory. These components reinforce each other, allowing the system to address monitoring, billing, risk control, and user experience simultaneously.

Another parking-specific requirement is the balance between real-time responsiveness and reliability. Vehicle passage at entrances/exits demands low latency, while the system must remain functional under unstable networks or partial device failures. This system addresses that by pushing decisions to the edge and adopting mechanisms such as offline detection for WSN nodes and an RFID in-card data loop that preserves critical billing data even when connectivity is weak. As a result, the core workflow (read/write, fee calculation, release decision) can remain available under adverse conditions, which is crucial for real deployments.

IoT systems in practice are also increasingly user-centric. “Connecting devices” alone is not the goal; delivering better services is. In this project, car owners benefit from travel assistance through the combination of weather data and LLM-generated recommendations at departure time, while managers gain visibility through a dashboard, alarms, and logs that reduce troubleshooting cost. This service orientation reflects the trend that IoT applications must be designed around users’ operational behaviors and experience expectations, not only around sensor connectivity.

Scalability is another essential feature in the parking domain because requirements

evolve: more parking bays, more device types, more pricing rules, and more service integrations. The layered and modular design used in this system supports extending the device layer (new WSN nodes or RFID readers), upgrading business logic (recommendation strategies and charging policies), and integrating new application services (reservation, mobile payment, multi-lot management). This makes the system adaptable as the scenario grows.

6.3 Understanding the Development Trend of IoT Technology

From the system implementation experience, IoT development is increasingly moving toward **intelligence**, **integration**, and **ecosystem-level collaboration**. First, edge intelligence will deepen. As sensor volume and data frequency increase, purely cloud-based processing cannot always satisfy low-latency requirements. The edge-side processing adopted here (local state inference, risk checks, and control decisions) is a practical example of this trend. In the future, more advanced tasks such as occupancy prediction, anomaly behavior detection, and adaptive control policies could be deployed at the edge to further reduce end-to-end latency and dependency on remote services.

Second, AIoT will become more pervasive. This system already demonstrates a lightweight AIoT pattern by using an LLM to generate travel recommendations as a value-added service. A more advanced direction is deeper AI integration across layers: improving sensing quality through data-driven calibration, enabling predictive analytics in the processing layer (e.g., peak-hour occupancy forecasting), and delivering more personalized services in the application layer (e.g., dynamic guidance based on user history).

Third, IoT ecosystems will expand beyond single isolated systems. A parking solution is naturally positioned to connect with urban transportation platforms, multi-parking-lot resource sharing, and mobile apps. As systems become interconnected, users can query availability across lots, reserve spaces, and complete unified payment, while operators can manage multiple sites through one platform. This cross-domain interconnection can dramatically expand the value boundary of a parking IoT system. Finally, security and standardization will become increasingly critical. As more devices and services are connected, risks related to device authentication, data integrity, privacy protection, and protocol interoperability grow. Even at the prototype level, measures such as frame validation and unique identification contribute to robustness. Future implementations should further incorporate secure transmission, access authentication, privacy-aware data handling, and standardized data formats/interfaces to reduce integration cost and improve cross-vendor compatibility.

6.4 Practical Enlightenment from System Development

This project reinforces several practical lessons about applying IoT technology. Most importantly, successful IoT design must be scenario-driven. The system's architecture and functions directly target pain points in small to medium parking lots—delayed availability information, irregular charging, ticket evasion risks, and departure experience—so every module has a clear operational purpose rather than being

technology for technology's sake.

At the same time, engineering choices should balance technical ambition with cost and feasibility. Instead of adopting high-cost solutions that may be difficult to deploy, the system uses mature technologies such as serial communication, WSN sensing, and RFID-based identification, while still achieving meaningful improvements through careful integration and edge-side logic. This trade-off makes the solution more realistic for adoption in resource-constrained scenarios.

Finally, reliability and maintainability are non-negotiable in IoT deployments because physical environments are noisy and failures are inevitable. Mechanisms such as WSN offline detection, RFID in-card data looping, and comprehensive log output improve fault tolerance and traceability, enabling the system to remain usable and diagnosable even under abnormal conditions. These design choices demonstrate that robust IoT engineering is not only about adding features, but also about designing for failures and long-term operation.

In summary, IoT is most valuable when it turns physical-world signals into stable, interpretable, and actionable services through multi-technology integration. By implementing sensing, transmission, edge processing, and user-oriented applications in one system, this project provides a concrete example of how IoT solves real parking management problems and how it can evolve further with advances in edge computing and AI.