□ Complete MQTT Guide: From Basics to IoT Car Implementation

A comprehensive, industry-standard guide to understanding and implementing MQTT protocol

Target Audience: *Developers, IoT Engineers, Students*
Project Context: *IoT Car Control System*
Last Updated: *January 25, 2026*

□ Table of Contents

What is MQTT?

Definition

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks.

Key Characteristics

Feature	Description	Why It Matters

Feature	Description	Why It Matters
Lightweight	Small code footprint (starting from ~30KB)	Runs on microcontrollers like ESP32
Low Bandwidth	Minimal packet overhead (2-byte header minimum)	Ideal for cellular/satellite connections
Publish-Subscribe	Decouples message sender from receiver	Scalable many-to-many communication
Asynchronous	Non-blocking message delivery	Real-time updates without polling
Quality of Service	3 levels (QoS 0, 1, 2)	Balance between reliability and speed
Persistent Sessions	Resume after disconnection	Mobile devices with intermittent connectivity

Where MQTT is Used

```
┌────────────────────────────────────────────────────────────────┐
│                    MQTT USE CASES                              │
├────────────────────────────────────────────────────────────────┤
│                                                                │
│  □ Industrial IoT (IIoT)                                      │
│     ├── Factory automation                                     │
│     ├── Sensor networks                                        │
│     └── Equipment monitoring                                   │
│                                                                │
│  □ Smart Home                                                 │
│     ├── Home Assistant                                         │
│     ├── Smart lighting (Philips Hue)                           │
│     └── Thermostats & sensors                                  │
│                                                                │
│  □ Automotive                                                 │
│     ├── Connected cars                                         │
│     ├── Fleet management                                       │
│     └── Telematics                                             │
│                                                                │
│  □ Mobile Apps                                                │
│     ├── Facebook Messenger                                     │
│     ├── Push notifications                                     │
│     └── Real-time chat                                         │
│                                                                │
│  □ Energy Management                                          │
│     ├── Smart meters                                           │
│     ├── Solar panel monitoring                                 │
│     └── Grid management                                        │
│                                                                │
│  □ Healthcare                                                 │
│     ├── Patient monitoring                                     │
│     ├── Medical device telemetry                               │
│     └── Remote diagnostics                                     │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```

Brief History

Year	Milestone
1999	Created by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom)
2010	Open-sourced and royalty-free
2013	MQTT 3.1 became OASIS standard
2014	MQTT 3.1.1 - Most widely used version
2019	MQTT 5.0 - Major update with new features

MQTT Architecture Fundamentals

The Publish-Subscribe Pattern

MQTT uses a **publish-subscribe** (pub/sub) model, which is fundamentally different from traditional request-response patterns (like HTTP).

Traditional Request-Response (HTTP)

```
Client                    Server
  │                          │
  │──── GET /temperature ───►│
  │                          │
  │◄─── 25°C ───────────────│
  │                          │
  └──────────────────────────┘


Problems:
□ Client must know server address
□ Server must be available when client requests
□ Tight coupling between client and server
□ Not scalable for many-to-many communication
```

Publish-Subscribe (MQTT)

```
Publisher              Broker              Subscriber(s)
    │                     │                      │
    │                     │  ┌──────────────────►│ Sub 1
    │                     │  │                   │
    │─── Publish ────────►│──┼──────────────────►│ Sub 2
    │   "temperature"     │  │                   │
    │   25°C              │  └──────────────────►│ Sub 3
    │                     │                      │
    └─────────────────────┴──────────────────────┘


Benefits:
□ Publishers don't know about subscribers
□ Subscribers don't know about publishers
□ Broker handles all message routing
□ Scalable many-to-many communication
□ Works even if some devices are offline
```

MQTT Architecture Components

```
┌───────────────────────────────────────────────────────────────────────┐
│                        MQTT ECOSYSTEM                                 │
├───────────────────────────────────────────────────────────────────────┤
│                                                                       │
│                    ┌──────────────────────┐                           │
│                    │   MQTT BROKER        │                           │
│                    │ (Message Router)     │                           │
│                    │                      │                           │
│                    │  ┌────────────────┐  │                           │
│                    │  │ Topic Storage  │  │                           │
│                    │  │  - sensors/#   │  │                           │
│                    │  │  - devices/#   │  │                           │
│                    │  │  - commands/#  │  │                           │
│                    │  └────────────────┘  │                           │
│                    │                      │                           │
│                    │  ┌────────────────┐  │                           │
│                    │  │ Session Store  │  │                           │
│                    │  │  - Client IDs  │  │                           │
│                    │  │  - Subscriptions│ │                           │
│                    │  │  - QoS queues  │  │                           │
│                    │  └────────────────┘  │                           │
│                    │                      │                           │
│                    └──────────┬───────────┘                           │
│                               │                                       │
│          ┌────────────────────┼────────────────────┐                  │
│          │                    │                    │                  │
│    ┌─────▼──────┐      ┌─────▼──────┐      ┌─────▼──────┐             │
│    │ Publisher  │      │ Publisher  │      │ Publisher  │             │
│    │   Client   │      │   Client   │      │   Client   │             │
│    │            │      │            │      │            │             │
│    │  ESP32     │      │  Sensor    │      │   Go       │             │
│    │  Device    │      │  Gateway   │      │  Backend   │             │
│    └────────────┘      └────────────┘      └────────────┘             │
│                                                                       │
│          ┌────────────────────┬────────────────────┐                  │
│          │                    │                    │                  │
│    ┌─────▼──────┐      ┌─────▼──────┐      ┌─────▼──────┐             │
│    │ Subscriber │      │ Subscriber │      │ Subscriber │             │
│    │   Client   │      │   Client   │      │   Client   │             │
│    │            │      │            │      │            │             │
│    │  Android   │      │   Web      │      │  Database  │             │
│    │    App     │      │  Dashboard │      │  Service   │             │
│    └────────────┘      └────────────┘      └────────────┘             │
│                                                                       │
│                                                                       │
│  Notes:                                                               │
│  • Clients can be both publishers AND subscribers                  │
│  • Broker is the single point of communication                     │
│  • Clients never communicate directly with each other              │
│  • Broker maintains persistent sessions for offline clients        │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

The Three Components

1. MQTT Broker (Server)

Role: Central message hub that routes messages between publishers and subscribers.

Popular Brokers:

Broker	Type	Best For
Mosquitto	Open-source	Development, small deployments
HiveMQ	Commercial/Cloud	Enterprise, scalability
EMQX	Open-source/Commercial	High performance, clustering
AWS IoT Core	Cloud	AWS ecosystem integration
Azure IoT Hub	Cloud	Azure ecosystem integration
VerneMQ	Open-source	Scalability, clustering

Key Responsibilities:

- Accept connections from clients
- Validate client authentication
- Route messages based on topics
- Manage subscriptions
- Queue messages for offline clients (QoS 1 & 2)
- Maintain persistent sessions

2. MQTT Publisher (Client)

Role: Sends messages to specific topics.

Examples in IoT Car Project:

- ESP32 publishing telemetry data
- ESP32 publishing status updates
- ESP32 publishing command acknowledgments

Code Example:

```
// ESP32 publishing temperature data
mqttClient.publish("iot-car/car-001/telemetry", "{\"temperature\":25}");
```

3. MQTT Subscriber (Client)

Role: Receives messages from topics they've subscribed to.

Examples in IoT Car Project:

- ESP32 subscribing to command topic
- Go backend subscribing to telemetry
- Android app subscribing to telemetry

Code Example:

```
// ESP32 subscribing to commands
mqttClient.subscribe("iot-car/car-001/command");
```

MQTT Core Concepts

1. Topics

Definition: Topics are hierarchical strings that identify the channel for a message.

Topic Structure

```
┌─────────────────────────────────────────────────────────────────┐
│                      MQTT TOPIC HIERARCHY                       │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Format: level1/level2/level3/level4                            │
│                                                                 │
│  Example: home/living-room/temperature                          │
│           └┬─┘ └────┬────┘ └─────┬──────┘                       │
│            │        │            │                              │
│          Domain   Location      Metric                          │
│                                                                 │
│  ─────────────────────────────────────────────────────────────  │
│                                                                 │
│  IoT Car Examples:                                              │
│                                                                 │
│  iot-car/car-001/telemetry                                      │
│  └──┬──┘ └──┬──┘ └───┬───┘                                      │
│   Domain  Device   Data Type                                    │
│                                                                 │
│  iot-car/car-001/command                                        │
│  iot-car/car-001/status                                         │
│  iot-car/car-001/response                                       │
│  iot-car/car-002/telemetry     ← Second car                     │
│  iot-car/fleet/broadcast       ← All cars                       │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Topic Rules

Rule	Allowed	Not Allowed
Characters	`a-z A-Z 0-9 - _ /`	Spaces, special chars
Case	Case-sensitive	`Car/001` ≠ `car/001`
Length	Up to 65,535 bytes	Keep under 200 chars
Leading `/`	`/iot-car/car-001`	Ambiguous, avoid
Trailing `/`	`iot-car/car-001/`	Ambiguous, avoid
Empty levels	`iot-car//car-001`	□ Invalid

Wildcards

MQTT supports two wildcards for **subscriptions only** (not for publishing):

Single-Level Wildcard: +

Matches **one** level in the hierarchy.

```
Subscribe to: iot-car/+/telemetry

Matches:
  □ iot-car/car-001/telemetry
  □ iot-car/car-002/telemetry
  □ iot-car/car-999/telemetry

Does NOT match:
  □ iot-car/telemetry              (missing level)
  □ iot-car/car-001/status         (different last level)
  □ iot-car/fleet/car-001/telemetry (too many levels)
```

Multi-Level Wildcard: `#`

Matches **zero or more** levels (must be last character).

```
Subscribe to: iot-car/car-001/#

Matches:
  □ iot-car/car-001/telemetry
  □ iot-car/car-001/status
  □ iot-car/car-001/command
  □ iot-car/car-001/sensors/temperature
  □ iot-car/car-001/sensors/gps/latitude

Subscribe to: iot-car/#

Matches:
  □ Everything under iot-car/
  □ iot-car/car-001/telemetry
  □ iot-car/fleet/broadcast
  □ iot-car/car-002/sensors/battery

Subscribe to: #

  ⚠ Matches ALL topics (use cautiously!)
```

Wildcard Combinations

```
Valid:
  □ iot-car/+/telemetry          # All cars' telemetry
  □ iot-car/+/sensors/+          # All cars, all sensors
  □ home/+/+/temperature         # All rooms, all devices
  □ iot-car/car-001/#            # Everything from car-001

Invalid:
  □ iot-car/car+/telemetry       # + must occupy entire level
  □ iot-car/#/telemetry          # # must be last
  □ iot-car/car-001#             # # must be after /
```

2. Messages

Definition: Payload (data) published to a topic.

Message Structure

```
┌─────────────────────────────────────────────────────────────────┐
│                     MQTT MESSAGE                                │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ FIXED HEADER (2-5 bytes)                                  │  │
│  ├───────────────────────────────────────────────────────────┤  │
│  │ • Message Type (PUBLISH, SUBSCRIBE, etc.)                 │  │
│  │ • QoS Level (0, 1, or 2)                                  │  │
│  │ • Retain Flag                                             │  │
│  │ • DUP Flag (duplicate)                                    │  │
│  │ • Remaining Length                                        │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ VARIABLE HEADER (depends on message type)                 │  │
│  ├───────────────────────────────────────────────────────────┤  │
│  │ • Topic Name: "iot-car/car-001/telemetry"                 │  │
│  │ • Packet Identifier (if QoS > 0)                          │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ PAYLOAD (0 - 256 MB)                                      │  │
│  ├───────────────────────────────────────────────────────────┤  │
│  │ {"device_id":"car-001",                                   │  │
│  │  "timestamp":1706169600,                                  │  │
│  │  "battery":85,                                            │  │
│  │  "distance_front":45}                                     │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Payload Format

MQTT is **payload-agnostic** - you can send any binary data.

Common Formats:

Format	Pros	Cons	Use Case
JSON	Human-readable, flexible	Larger size	Most IoT projects
Protocol Buffers	Compact, typed	Not human-readable	High-frequency data
MessagePack	Compact, JSON-like	Less common	Bandwidth-constrained
Plain Text	Simple	No structure	Simple sensors
Binary	Most compact	Custom parsing	Raw sensor data

Industry Standard: JSON for most IoT applications due to:

- Wide library support
- Easy debugging
- Cross-platform compatibility
- Self-documenting structure

3. Quality of Service (QoS)

See dedicated QoS section below.

4. Retained Messages

Definition: When a message is published with the **retain flag**, the broker stores it and delivers it to future subscribers immediately upon subscription.

How It Works

```
Timeline of Events:

1. Publisher sends message with retain=true

   ┌──────────┐
   │Publisher │──► Topic: "status"
   └──────────┘    Payload: "online"
                   Retain: TRUE
                        │
                        ▼
                  ┌─────────┐
                  │ Broker  │── Stores message
                  └─────────┘

2. Subscriber 1 connects (now)

   ┌────────────┐
   │Subscriber 1│◄─── Receives "online" IMMEDIATELY
   └────────────┘

3. Subscriber 2 connects (1 hour later)

   ┌────────────┐
   │Subscriber 2│◄─── STILL receives "online" IMMEDIATELY
   └────────────┘
```

Use Cases

```
 Good for:
  • Device status (online/offline)
  • Configuration updates
  • Last known values (temperature, GPS position)
  • Presence detection

 Not good for:
  • Real-time events (button presses)
  • Historical data (use database)
  • Rapidly changing values
```

Clearing Retained Messages

```
// Send empty payload with retain=true to clear
mqttClient.publish("iot-car/car-001/status", "", true);
```

5. Persistent Sessions

Definition: Broker stores client's subscriptions and queued messages even after disconnection.

How It Works

```
┌─────────────────────────────────────────────────────────────────┐
│              PERSISTENT SESSION FLOW                            │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  1. Client Connects with cleanSession=false                     │
│      ┌────────┐                                                 │
│      │ Client │──▶ CONNECT (ClientID="car-001", Clean=false)    │
│      └────────┘                                                 │
│                         │                                       │
│                         ▼                                       │
│                    ┌─────────┐                                  │
│                    │ Broker  │── Creates session for "car-001"  │
│                    └─────────┘                                  │
│                                                                 │
│  2. Client Subscribes                                           │
│      ┌────────┐                                                 │
│      │ Client │──▶ SUBSCRIBE "commands/#"                       │
│      └────────┘                                                 │
│                         │                                       │
│                         ▼                                       │
│                    ┌─────────┐                                  │
│                    │ Broker  │── Stores subscription in session │
│                    └─────────┘                                  │
│                                                                 │
│  3. Client Disconnects (unexpectedly)                           │
│      ┌────────┐                                                 │
│      │ Client │──✗ Connection lost!                              │
│      └────────┘                                                 │
│                                                                 │
│  4. Messages arrive while offline                               │
│                    ┌─────────┐                                  │
│     Publisher ──▶│ Broker  │── Queues messages for "car-001"   │
│                    └─────────┘    (if QoS 1 or 2)               │
│                                                                 │
│  5. Client Reconnects                                           │
│      ┌────────┐                                                 │
│      │ Client │──▶ CONNECT (ClientID="car-001", Clean=false)    │
│      └────────┘                                                 │
│                         │                                       │
│                         ▼                                       │
│                    ┌─────────┐                                  │
│                    │ Broker  │── Restores session               │
│                    └─────────┘── Delivers queued messages       │
│                                  Subscriptions already active!  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Configuration

```
// ESP32 - Enable persistent session
mqttClient.connect(clientId, NULL, NULL, NULL, 0, false, NULL, false);
//                                                          ↑
//                                                    cleanSession=false

// Android - Enable persistent session
MqttConnectOptions options = new MqttConnectOptions();
options.setCleanSession(false);  // Persistent session
```

6. Last Will and Testament (LWT)

Definition: A message the broker automatically sends when a client disconnects ungracefully.

How It Works

```
┌─────────────────────────────────────────────────────────────────┐
│                   LAST WILL TESTAMENT                           │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  1. Client Connects and Specifies LWT                           │
│     ┌────────┐                                                  │
│     │ESP32   │──► CONNECT                                       │
│     └────────┘      LWT Topic: "iot-car/car-001/status"         │
│                     LWT Payload: "offline"                      │
│                     LWT QoS: 1                                  │
│                     LWT Retain: true                            │
│                        │                                        │
│                        ▼                                        │
│                   ┌─────────┐                                   │
│                   │ Broker  │── Stores LWT (doesn't publish yet)│
│                   └─────────┘                                   │
│                                                                 │
│  2. Normal Operation                                            │
│     ┌────────┐    ┌─────────┐                                   │
│     │ESP32   │◄──►│ Broker  │── Everything works fine           │
│     └────────┘    └─────────┘                                   │
│                                                                 │
│  3. Client Disconnects Gracefully (sends DISCONNECT)            │
│     ┌────────┐                                                  │
│     │ESP32   │──► DISCONNECT                                    │
│     └────────┘                                                  │
│                        │                                        │
│                        ▼                                        │
│                   ┌─────────┐                                   │
│                   │ Broker  │── Discards LWT (NOT published)    │
│                   └─────────┘                                   │
│                                                                 │
│  4. Client Crashes/Network Lost (ungraceful disconnect)         │
│     ┌────────┐                                                  │
│     │ESP32   │──✗ Connection timeout!                            │
│     └────────┘                                                  │
│                        │                                        │
│                        ▼                                        │
│                   ┌─────────┐                                   │
│                   │ Broker  │── Publishes LWT: "offline"        │
│                   └─────────┘                                   │
│                        │                                        │
│                        ▼                                        │
│                   ┌──────────┐                                  │
│                   │Subscribers│◄─ Notified device is offline    │
│                   └──────────┘                                  │
│                                                                 │
```

```
|                                                                  |
└──────────────────────────────────────────────────────────────────┘
```

Implementation

```
// ESP32 - Set Last Will
mqttClient.connect(
    "car-001",                          // Client ID
    NULL,                               // Username (none)
    NULL,                               // Password (none)
    "iot-car/car-001/status",          // LWT Topic
    1,                                  // LWT QoS
    true,                               // LWT Retain
    "{\"status\":\"offline\"}",        // LWT Payload
    true                                // Clean Session
);

// When connected, immediately send "online"
mqttClient.publish(
    "iot-car/car-001/status",
    "{\"status\":\"online\"}",
    true  // Retain
);
```

Quality of Service (QoS) Levels

QoS defines the **guarantee of delivery** for a message.

QoS Level Comparison

Level	Name	Guarantee	Overhead	Use Case
QoS 0	At most once	No guarantee	Lowest	Sensor data (ok to lose)
QoS 1	At least once	Guaranteed, duplicates possible	Medium	Commands, alerts
QoS 2	Exactly once	Guaranteed, no duplicates	Highest	Billing, critical commands

QoS 0: At Most Once (Fire and Forget)

```
Publisher              Broker              Subscriber
    │                     │                      │
    │──── PUBLISH ───────►│                      │
    │   (QoS 0)           │                      │
    │                     │──── PUBLISH ────────►│
    │                     │   (QoS 0)            │
    │                     │                      │
    └─────────────────────┴──────────────────────┘

No acknowledgment!
If network fails, message is LOST □
```

Characteristics:

- □ Fastest
- □ Lowest bandwidth
- □ No retry
- □ May lose messages

When to Use:

- High-frequency sensor data (GPS, temperature)
- Data where newer values replace old ones
- Non-critical telemetry

QoS 1: At Least Once (Acknowledged Delivery)

```
Publisher              Broker              Subscriber
    │                     │                      │
    │──── PUBLISH ───────►│                      │
    │   (QoS 1, ID=123)   │                      │
    │                     │──── PUBLISH ────────►│
    │                     │   (QoS 1, ID=456)    │
    │                     │                      │
    │                     │◄──── PUBACK ─────────│
    │                     │   (ID=456)           │
    │◄──── PUBACK ────────│                      │
    │   (ID=123)          │                      │
    │                     │                      │
    └─────────────────────┴──────────────────────┘

Publisher retries if PUBACK not received!
⚠ Subscriber might receive duplicates
```

Characteristics:

- □ Guaranteed delivery
- □ Automatic retry
- ⚠ Possible duplicates
- □ Moderate overhead

When to Use:

- Control commands (forward, stop, turn)
- Alerts and notifications
- Device configuration updates

Handling Duplicates:

```
// Subscriber should implement duplicate detection
String lastCommandId = "";

void onMessage(String topic, String payload) {
    StaticJsonDocument<256> doc;
    deserializeJson(doc, payload);

    String commandId = doc["id"];  // Include unique ID in messages

    if (commandId == lastCommandId) {
        Serial.println("Duplicate detected, ignoring");
        return;
    }

    lastCommandId = commandId;
    processCommand(doc);
}
```

QoS 2: Exactly Once (Assured Delivery)

```
Publisher              Broker             Subscriber
    │                     │                      │
    │──── PUBLISH ───────►│                      │
    │  (QoS 2, ID=123)    │                      │
    │                     │                      │
    │◄──── PUBREC ────────│                      │
    │  (ID=123)           │                      │
    │                     │                      │
    │──── PUBREL ────────►│                      │
    │  (ID=123)           │                      │
    │                     │──── PUBLISH ────────►│
    │                     │  (QoS 2, ID=789)     │
    │                     │                      │
    │                     │◄──── PUBREC ─────────│
    │◄──── PUBCOMP ───────│  (ID=789)            │
    │  (ID=123)           │                      │
    │                     │──── PUBREL ─────────►│
    │                     │  (ID=789)            │
    │                     │                      │
    │                     │◄──── PUBCOMP ────────│
    │                     │  (ID=789)            │
    │                     │                      │
    └─────────────────────┴──────────────────────┘

4-way handshake!
Guarantees exactly-once delivery, no duplicates □
```

Characteristics:

- □ Exactly once guarantee
- □ No duplicates
- □ Highest latency
- □ Most bandwidth
- □ Most CPU/memory

When to Use:

- Financial transactions
- Billing/metering
- Critical commands (emergency stop)
- Rarely used in IoT due to overhead

QoS Downgrade

Important: The effective QoS is the **minimum** of publisher and subscriber QoS.

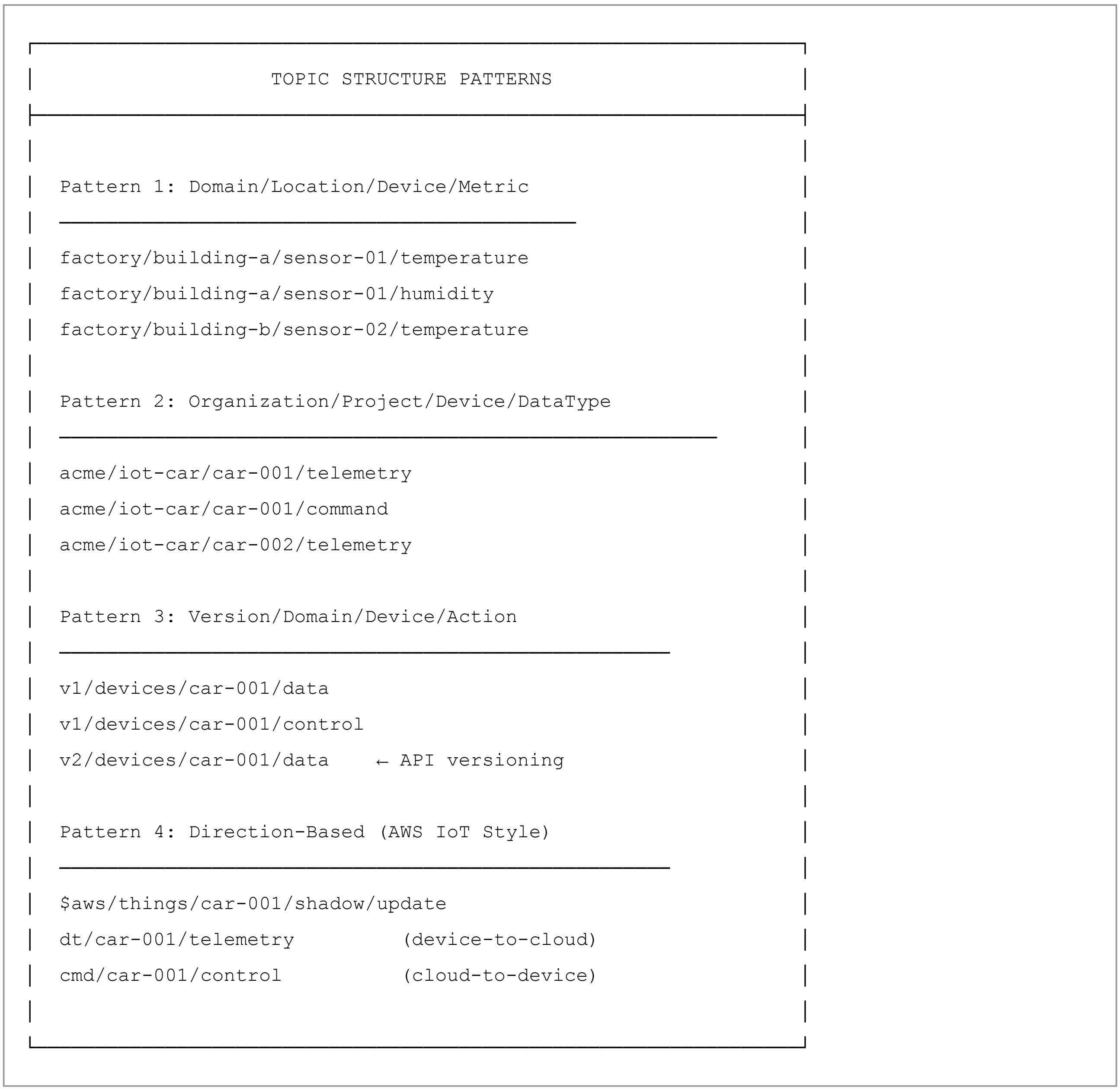
```
Publisher QoS 2  →  Broker  →  Subscriber QoS 0
                                     ↓
                           Effective QoS 0!

Publisher QoS 1  →  Broker  →  Subscriber QoS 2
         ↓
   Effective QoS 1!
```

Industry Recommendations

```
┌──────────────────────────────────────────────────────────────────┐
│                    QoS SELECTION GUIDE                           │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  Telemetry (sensor data)                →  QoS 0                │
│  Commands (forward, stop)               →  QoS 1                │
│  Critical commands (emergency stop)     →  QoS 1 or 2           │
│  Configuration updates                  →  QoS 1                │
│  Status updates (online/offline)        →  QoS 1 + Retain       │
│  File transfer                          →  QoS 2                │
│  Over cellular/satellite                →  QoS 0 or 1           │
│  Local WiFi                             →  QoS 1                │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

Topic Design Best Practices

Industry Standard Naming Conventions



```
┌────────────────────────────────────────────────────────────┐
│                   TOPIC STRUCTURE PATTERNS                 │
├────────────────────────────────────────────────────────────┤
│                                                            │
│  Pattern 1: Domain/Location/Device/Metric                  │
│  ────────────────────────────────────────                  │
│  factory/building-a/sensor-01/temperature                  │
│  factory/building-a/sensor-01/humidity                     │
│  factory/building-b/sensor-02/temperature                  │
│                                                            │
│  Pattern 2: Organization/Project/Device/DataType           │
│  ───────────────────────────────────────────────────       │
│  acme/iot-car/car-001/telemetry                            │
│  acme/iot-car/car-001/command                              │
│  acme/iot-car/car-002/telemetry                            │
│                                                            │
│  Pattern 3: Version/Domain/Device/Action                   │
│  ───────────────────────────────────────────────           │
│  v1/devices/car-001/data                                   │
│  v1/devices/car-001/control                                │
│  v2/devices/car-001/data    ← API versioning               │
│                                                            │
│  Pattern 4: Direction-Based (AWS IoT Style)                │
│  ───────────────────────────────────────────────           │
│  $aws/things/car-001/shadow/update                         │
│  dt/car-001/telemetry         (device-to-cloud)            │
│  cmd/car-001/control          (cloud-to-device)            │
│                                                            │
└────────────────────────────────────────────────────────────┘
```

IoT Car Project Topic Architecture

```
iot-car/
│
├── car-001/
│   ├── telemetry           ← Device publishes sensor data
│   ├── status              ← Device publishes online/offline (LWT)
│   ├── command             ← Device subscribes for control
│   ├── response            ← Device publishes command ACKs
│   │
│   └── sensors/            ← Future expansion
│       ├── gps/
│       │   ├── latitude
│       │   └── longitude
│       ├── battery
│       └── temperature
│
├── car-002/
│   ├── telemetry
│   ├── status
│   └── ...
│
├── fleet/
│   ├── broadcast           ← Commands to all cars
│   └── config              ← Configuration updates
│
└── admin/
    ├── logs
    └── diagnostics
```

Topic Design Rules

Rule	Good ✓	Bad ✗
Use lowercase	`iot-car/car-001`	`IoT-Car/Car-001`
Use hyphens	`car-001`, `living-room`	`car_001`, `livingRoom`
Be specific	`iot-car/car-001/sensors/gps/latitude`	`data/1/gps/lat`
Singular nouns	`iot-car/car-001`	`iot-cars/car-001`
No spaces	`iot-car/living-room`	`iot-car/living room`
No special chars	`iot-car/car-001`	`iot-car/car#001`
Consistent depth	All devices at same level	Mixed hierarchy

Access Control with Topics

```
┌─────────────────────────────────────────────────────────────────┐
│              TOPIC-BASED ACCESS CONTROL (ACL)                   │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  User: car-001-device                                           │
│  ─────────────────────                                          │
│  CAN Publish:                                                   │
│    □ iot-car/car-001/telemetry                                 │
│    □ iot-car/car-001/status                                    │
│    □ iot-car/car-001/response                                  │
│                                                                 │
│  CAN Subscribe:                                                 │
│    □ iot-car/car-001/command                                   │
│    □ iot-car/fleet/broadcast                                   │
│                                                                 │
│  CANNOT:                                                        │
│    □ iot-car/car-002/*        (other devices)                  │
│    □ iot-car/admin/*          (admin topics)                   │
│                                                                 │
│  ───────────────────────────────────────────────────────────    │
│                                                                 │
│  User: mobile-app-user-123                                      │
│  ──────────────────────────                                     │
│  CAN Publish:                                                   │
│    □ iot-car/+/command        (control any car user owns)      │
│                                                                 │
│  CAN Subscribe:                                                 │
│    □ iot-car/+/telemetry      (monitor any car user owns)      │
│    □ iot-car/+/status                                          │
│                                                                 │
│  CANNOT:                                                        │
│    □ iot-car/+/response       (internal device communication)  │
│                                                                 │
│  ───────────────────────────────────────────────────────────    │
│                                                                 │
│  User: backend-service                                          │
│  ──────────────────────                                         │
│  CAN Subscribe:                                                 │
│    □ iot-car/#                (all topics, logging)            │
│                                                                 │
│  CAN Publish:                                                   │
│    □ iot-car/fleet/broadcast  (system-wide commands)           │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

MQTT Message Structure

JSON Message Format (Industry Standard)

Telemetry Message

```
{
  "device_id": "car-001",
  "timestamp": 1706169600,
  "battery": 85,
  "sensors": {
    "distance_front": 45,
    "distance_rear": 120,
    "temperature": 28,
    "gps": {
      "latitude": 6.9271,
      "longitude": 79.8612,
      "altitude": 15
    }
  },
  "status": {
    "motors": "idle",
    "wifi_rssi": -45,
    "uptime": 3600
  }
}
```

Command Message

```
{
  "id": "cmd-1706169600-abc123",
  "timestamp": 1706169600,
  "action": "forward",
  "parameters": {
    "duration": 5000,
    "speed": 80
  },
  "priority": "normal"
}
```

Response/Acknowledgment

```
{
  "command_id": "cmd-1706169600-abc123",
  "timestamp": 1706169601,
  "status": "success",
  "message": "Command executed",
  "execution_time_ms": 50
}
```

Message Size Optimization

```
┌─────────────────────────────────────────────────────────────────┐
│                 MESSAGE SIZE COMPARISON                         │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Verbose JSON (Human-Readable):                                 │
│  ───────────────────────────────────                            │
│  {                                                              │
│    "device_identifier": "car-001",                              │
│    "timestamp_unix_epoch": 1706169600,                          │
│    "battery_percentage": 85,                                    │
│    "distance_sensor_front_cm": 45                               │
│  }                                                              │
│  Size: ~150 bytes                                               │
│                                                                 │
│  ───────────────────────────────────────────────────────────    │
│                                                                 │
│  Compact JSON (Production):                                     │
│  ───────────────────────────────────                            │
│  {"id":"car-001","ts":1706169600,"bat":85,"dist":45}            │
│  Size: ~54 bytes (64% smaller!)                                 │
│                                                                 │
│  ───────────────────────────────────────────────────────────    │
│                                                                 │
│  Binary (Protocol Buffers):                                     │
│  ───────────────────────────────────                            │
│  0x0a 0x07 0x63 0x61 0x72 0x2d 0x30 0x30 0x31 ...               │
│  Size: ~25 bytes (83% smaller!)                                 │
│                                                                 │
│  Trade-off:                                                     │
│  • JSON: Debuggable, flexible, widely supported                 │
│  • Binary: Compact, fast, requires schema                       │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Recommendation for IoT Car: Use compact JSON

- Balance between readability and size
- Easy debugging during development
- No schema management complexity

MQTT Security

Security Layers

```
┌─────────────────────────────────────────────────────────────────┐
│                    MQTT SECURITY STACK                          │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Layer 4: Authorization (ACL)                                   │
│  ─────────────────────────────                                  │
│  • Topic-based permissions                                      │
│  • Read/Write access control                                    │
│  • User/device isolation                                        │
│                                                                 │
│  Layer 3: Authentication                                        │
│  ───────────────────────────                                    │
│  • Username/Password                                            │
│  • Client certificates (X.509)                                  │
│  • Token-based (JWT)                                            │
│                                                                 │
│  Layer 2: Transport Encryption (TLS/SSL)                        │
│  ────────────────────────────────────────                       │
│  • Encrypt data in transit                                      │
│  • Prevent eavesdropping                                        │
│  • Port 8883 (MQTT over TLS)                                    │
│                                                                 │
│  Layer 1: Network Security                                      │
│  ──────────────────────────                                     │
│  • Firewall rules                                               │
│  • VPN for remote access                                        │
│  • Private networks                                             │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

TLS/SSL Implementation

Port Configuration

Port	Protocol	Security	Use Case
1883	MQTT	None □	Development, local network only

Port	Protocol	Security	Use Case
8883	MQTTS	TLS □	Production, internet-facing
9001	WebSocket	None □	Browser clients, local
8884	WebSocket	TLS □	Browser clients, production

ESP32 TLS Connection

```
#include <WiFiClientSecure.h>
#include <PubSubClient.h>

// Certificate for mosquitto broker
const char* mqtt_server_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDXTCCAkWgAwIBAgIUabcdefg...\n" \
"-----END CERTIFICATE-----\n";

WiFiClientSecure espClient;
PubSubClient mqttClient(espClient);

void setup() {
    // Load CA certificate
    espClient.setCACert(mqtt_server_cert);

    // Connect to secure broker
    mqttClient.setServer("broker.example.com", 8883);

    // Connect with username/password
    mqttClient.connect(
        "car-001",                    // Client ID
        "car-001-user",               // Username
        "secure-password-here",       // Password
        "iot-car/car-001/status",    // LWT topic
        1,                            // LWT QoS
        true,                         // LWT retain
        "{\"status\":\"offline\"}",  // LWT payload
        true                          // Clean session
    );
}
```

Authentication Methods

1. Username/Password

```
# Mosquitto configuration
allow_anonymous false
password_file /mosquitto/config/passwd

# Create password file
mosquitto_passwd -c /mosquitto/config/passwd car-001-user
```

2. Client Certificates (Mutual TLS)

```
# Mosquitto configuration
cafile /mosquitto/certs/ca.crt
certfile /mosquitto/certs/server.crt
keyfile /mosquitto/certs/server.key
require_certificate true
```

3. Access Control Lists (ACL)

```
# Mosquitto ACL file: /mosquitto/config/acl

# Device car-001
user car-001-device
topic write iot-car/car-001/telemetry
topic write iot-car/car-001/status
topic write iot-car/car-001/response
topic read iot-car/car-001/command
topic read iot-car/fleet/broadcast

# Mobile app user
user app-user-123
topic write iot-car/+/command
topic read iot-car/+/telemetry
topic read iot-car/+/status

# Backend service (full access)
user backend-service
topic readwrite iot-car/#
```

Industry Standard Patterns

1. Command-Response Pattern

```
┌─────────────────────────────────────────────────────────────────┐
│                  COMMAND-RESPONSE FLOW                          │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  App                    Broker                  Device          │
│   │                      │                       │              │
│   │──1. Command─────────▶│                       │              │
│   │  "forward"           │──2. Command──────────▶│              │
│   │                      │   "forward"           │              │
│   │                      │                       │──3. Execute  │
│   │                      │                       │   motors!    │
│   │                      │                       │              │
│   │                      │◀──4. Response─────────│              │
│   │◀─5. Response─────────│   "success"           │              │
│   │  "success"           │                       │              │
│   │                      │                       │              │
│   └──────────────────────┴───────────────────────┘              │
│                                                                 │
│  Topics:                                                        │
│  • Command:  iot-car/car-001/command                            │
│  • Response: iot-car/car-001/response                           │
│                                                                 │
│  Message Flow:                                                  │
│  1. App publishes command with unique ID                        │
│  2. Device receives and validates command                       │
│  3. Device executes action                                      │
│  4. Device publishes response with same ID                      │
│  5. App matches response to command via ID                      │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

2. Telemetry Streaming Pattern

```
Device ──────▶ Broker ──────┬─────▶ Backend (Logging)
  │               │          │
  │               │          ├─────▶ Mobile App (Display)
  │               │          │
  │               │          └─────▶ Database (Storage)
  │               │
  └──Every 1s─────┘

Topic: iot-car/car-001/telemetry
QoS: 0 (fire and forget, data is time-series)
Retain: false (historical data not needed)
```

3. Configuration Update Pattern

```
Backend ────▶ Broker ────▶ Device(s)
                                │
                                └──▶ Acknowledges update
                                └──▶ Applies configuration
                                └──▶ Reboots if needed

Topic: iot-car/car-001/config
QoS: 1 (ensure delivery)
Retain: true (persist configuration)
```

4. Presence Detection Pattern

```
Device Connects:
  ├──▶ Publish "online" (retained)
  └──▶ Set LWT to "offline" (retained)

Device Disconnects:
  └──▶ Broker automatically publishes LWT "offline"

Subscribers:
  └──▶ Always see last known status (retained message)

Topic: iot-car/car-001/status
Payload: {"status":"online","timestamp":1706169600}
QoS: 1
Retain: true
```

IoT Car Project Architecture

Complete System Architecture

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                        IoT CAR MQTT ARCHITECTURE                            │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│   ┌──────────────────────────────────────────────────────────────────┐      │
│   │                    MOSQUITTO BROKER                              │      │
│   │                    (localhost:1883)                              │      │
│   │                                                                  │      │
│   │   Topic Structure:                                               │      │
│   │   ────────────────                                               │      │
│   │   iot-car/                                                       │      │
│   │     ├── car-001/                                                 │      │
│   │     │   ├── telemetry    (ESP32 → Backend/App)                   │      │
│   │     │   ├── status       (ESP32 → All, LWT, Retained)            │      │
│   │     │   ├── command      (App → ESP32)                           │      │
│   │     │   └── response     (ESP32 → App)                           │      │
│   │     └── fleet/                                                   │      │
│   │         └── broadcast    (Backend → All cars)                    │      │
│   │                                                                  │      │
│   └────────────┬────────────────────────────────┬────────────────────┘      │
│                │                                │                           │
│         ┌──────▼────────┐              ┌────────▼────────┐                  │
│         │               │              │                 │                  │
│    ┌────┴─────┐   ┌─────┴────┐   ┌─────┴────┐   ┌────────┴───────┐          │
│    │          │   │          │   │          │   │                │          │
│    │  ESP32   │   │    Go    │   │ Android  │   │  Web Dashboard │          │
│    │ Device   │   │ Backend  │   │   App    │   │   (Future)     │          │
│    │          │   │          │   │          │   │                │          │
│    └──────────┘   └──────────┘   └──────────┘   └────────────────┘          │
│                                                                             │
│    Publishes:         Subscribes:     Publishes:                            │
│    • telemetry        • telemetry     • command                             │
│    • status (LWT)     • status        Subscribes:                           │
│    • response         • response      • telemetry                           │
│    Subscribes:                        • status                              │
│    • command                          • response                            │
│    • broadcast                                                              │
│                                                                             │
└─────────────────────────────────────────────────────────────────────────────┘
```

Message Flow Examples

Example 1: User Sends "Forward" Command

```
Step-by-Step Flow:

1. USER ACTION
   ┌─────────────┐
   │ Android App │
   │ User taps   │
   │ "FORWARD"   │
   └──────┬──────┘
          │
2. PUBLISH COMMAND
   │ Topic: iot-car/car-001/command
   │ Payload: {"id":"cmd-123","action":"forward","speed":80}
   │ QoS: 1 (ensure delivery)
   ▼
   ┌──────────────┐
   │   Mosquitto  │ ← Receives message
   │    Broker    │
   └──────┬───────┘
          │
3. ROUTE TO DEVICE
   │ Finds subscribers to "iot-car/car-001/command"
   │ ESP32 is subscribed!
   ▼
   ┌──────────────┐
   │    ESP32     │ ← Receives command
   │   (car-001)  │
   └──────┬───────┘
          │
4. EXECUTE COMMAND
   │ Parse JSON
   │ Validate action
   │ Set motor GPIOs HIGH
   │ Motors start spinning!
   ▼
   ┌──────────────┐
   │   Motors     │ ← Physical movement
   │   Running    │
   └──────┬───────┘
          │
5. SEND ACKNOWLEDGMENT
   │ Topic: iot-car/car-001/response
   │ Payload: {"command_id":"cmd-123","status":"success"}
   │ QoS: 1
   ▼
   ┌──────────────┐
   │   Mosquitto  │
   │     Broker     │
   └───────┬────────┘
           │
6. DELIVER TO APP
   │ Routes to Android App
   ▼
   ┌────────────────┐
   │ Android App    │ ← Shows "Command Executed ✓"
   └────────────────┘

Total Time: ~100-300ms
```

Example 2: ESP32 Sends Telemetry

```
Step-by-Step Flow:

1. SENSOR READING
   ┌──────────────┐
   │   ESP32      │
   │ Loop every   │
   │  1 second    │
   └──────┬───────┘
          │
2. READ SENSORS
   │ distance = readUltrasonic()      → 45cm
   │ battery = readBattery()          → 85%
   │ temperature = readTemperature()  → 28°C
   ▼
   ┌──────────────┐
   │  Build JSON  │
   └──────┬───────┘
          │
3. PUBLISH TELEMETRY
   │ Topic: iot-car/car-001/telemetry
   │ Payload: {"device_id":"car-001","battery":85,"distance":45,"temp":28}
   │ QoS: 0 (fast, ok to lose occasional message)
   ▼
   ┌──────────────┐
   │   Mosquitto  │
   │    Broker    │
   └──────┬───────┘
          │
4. ROUTE TO SUBSCRIBERS
   ├──────────────┬────────────────┐
   │              │                │
   ▼              ▼                ▼
┌─────────┐  ┌──────────┐  ┌──────────────┐
│   Go    │  │ Android  │  │     Web      │
│ Backend │  │   App    │  │  Dashboard   │
└────┬────┘  └────┬─────┘  └──────┬───────┘
     │            │                │
5. PROCESS DATA
     │            │                │
     ├─ Log to file               │
     │            └─ Update UI     │
     │                             └─ Display chart
     └─ (Future: Save to database)

Frequency: Every 1 second
Bandwidth: ~100 bytes/sec (very low!)
```

Configuration for Each Component

ESP32 Configuration

```
// include/config.h

// WiFi
const char* WIFI_SSID = "YourWiFiName";
const char* WIFI_PASSWORD = "YourWiFiPassword";

// MQTT Broker
const char* MQTT_BROKER = "192.168.1.100";  // Your PC's IP
const int MQTT_PORT = 1883;

// Device Identity
const char* DEVICE_ID = "car-001";

// Topics
const char* TOPIC_TELEMETRY = "iot-car/car-001/telemetry";
const char* TOPIC_STATUS = "iot-car/car-001/status";
const char* TOPIC_COMMAND = "iot-car/car-001/command";
const char* TOPIC_RESPONSE = "iot-car/car-001/response";

// QoS Levels
const int QOS_TELEMETRY = 0;  // Fire and forget
const int QOS_COMMAND = 1;    // Ensure delivery
const int QOS_STATUS = 1;     // Ensure delivery

// Telemetry interval
const long TELEMETRY_INTERVAL_MS = 1000;  // 1 second
```

Go Backend Configuration

```
// config.go

const (
    BrokerAddress = "localhost:1883"
    ClientID      = "go-backend-service"

    // Subscribe to all car telemetry
    TopicTelemetry = "iot-car/+/telemetry"
    TopicStatus    = "iot-car/+/status"
    TopicResponse  = "iot-car/+/response"

    // Publish to fleet
    TopicFleetBroadcast = "iot-car/fleet/broadcast"

    QOSSubscribe = 1
)
```

Android App Configuration

```
// MqttManager.java

private static final String BROKER_URL = "tcp://192.168.1.100:1883";
private static final String CLIENT_ID = "android-app-" + System.currentTimeMillis();

// Device-specific topics
private String deviceId = "car-001";
private String topicCommand = "iot-car/" + deviceId + "/command";
private String topicTelemetry = "iot-car/" + deviceId + "/telemetry";
private String topicStatus = "iot-car/" + deviceId + "/status";
private String topicResponse = "iot-car/" + deviceId + "/response";

private static final int QOS_COMMAND = 1;
private static final int QOS_SUBSCRIBE = 1;
```

Practical Implementation Examples

ESP32 Complete Implementation

```
// src/main.cpp

#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "config.h"

WiFiClient espClient;
PubSubClient mqttClient(espClient);

unsigned long lastTelemetry = 0;
String currentCommand = "stop";

void connectWiFi();
void connectMQTT();
void onMqttMessage(char* topic, byte* payload, unsigned int length);
void sendTelemetry();
void sendResponse(String commandId, String status, String message);
void executeCommand(String action);

void setup() {
    Serial.begin(115200);
    Serial.println("\n\n=== IoT Car Starting ===");

    // Setup motors (not shown)
    setupMotors();

    // Connect to WiFi
    connectWiFi();

    // Configure MQTT
    mqttClient.setServer(MQTT_BROKER, MQTT_PORT);
    mqttClient.setCallback(onMqttMessage);
    mqttClient.setBufferSize(512);  // Increase buffer for large messages

    // Connect to MQTT
    connectMQTT();
}

void loop() {
    // Maintain MQTT connection
    if (!mqttClient.connected()) {
        connectMQTT();
    }
    mqttClient.loop();
    // Send telemetry periodically
    unsigned long now = millis();
    if (now - lastTelemetry > TELEMETRY_INTERVAL_MS) {
        lastTelemetry = now;
        sendTelemetry();
    }
}

void connectWiFi() {
    Serial.print("Connecting to WiFi: ");
    Serial.println(WIFI_SSID);

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("\nWiFi connected!");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
}

void connectMQTT() {
    while (!mqttClient.connected()) {
        Serial.print("Connecting to MQTT broker...");

        // Prepare Last Will Testament
        String lwtPayload = "{\"device_id\":\"" + String(DEVICE_ID) +
                           "\",\"status\":\"offline\",\"timestamp\":" +
                           String(millis()) + "}";

        // Connect with LWT
        if (mqttClient.connect(
            DEVICE_ID,                    // Client ID
            NULL,                         // Username
            NULL,                         // Password
            TOPIC_STATUS,                 // LWT Topic
            QOS_STATUS,                   // LWT QoS
            true,                         // LWT Retain
            lwtPayload.c_str(),          // LWT Payload
            true                          // Clean Session
        )) {
            Serial.println("connected!");
            // Publish online status (overrides LWT)
            String onlinePayload = "{\"device_id\":\"" + String(DEVICE_ID) +
                                   "\",\"status\":\"online\",\"timestamp\":" +
                                   String(millis()) + "}";
            mqttClient.publish(TOPIC_STATUS, onlinePayload.c_str(), true);

            // Subscribe to command topic
            mqttClient.subscribe(TOPIC_COMMAND, QOS_COMMAND);
            Serial.print("Subscribed to: ");
            Serial.println(TOPIC_COMMAND);

        } else {
            Serial.print("failed, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" retrying in 5 seconds...");
            delay(5000);
        }
    }
}

void onMqttMessage(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message received [");
    Serial.print(topic);
    Serial.print("]: ");

    // Convert payload to string
    String message;
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.println(message);

    // Parse JSON
    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, message);

    if (error) {
        Serial.print("JSON parse failed: ");
        Serial.println(error.c_str());
        return;
    }

    // Extract command fields
    String commandId = doc["id"] | "unknown";
    String action = doc["action"] | "stop";
    int speed = doc["speed"] | 80;

    Serial.print("Executing command: ");
    Serial.println(action);

    // Execute command
    executeCommand(action);

    // Send acknowledgment
    sendResponse(commandId, "success", "Command executed");
}

void sendTelemetry() {
    // Read sensors (simplified)
    int distance = readUltrasonic();  // Implement based on your sensor
    int battery = readBattery();       // Implement based on your battery monitoring
    int temp = 25;  // Placeholder

    // Build JSON
    StaticJsonDocument<256> doc;
    doc["device_id"] = DEVICE_ID;
    doc["timestamp"] = millis();
    doc["battery"] = battery;
    doc["distance_front"] = distance;
    doc["temperature"] = temp;
    doc["current_action"] = currentCommand;
    doc["wifi_rssi"] = WiFi.RSSI();

    // Serialize to string
    String output;
    serializeJson(doc, output);

    // Publish
    mqttClient.publish(TOPIC_TELEMETRY, output.c_str(), false);  // QoS 0, no retain

    Serial.print("Telemetry sent: ");
    Serial.println(output);
}

void sendResponse(String commandId, String status, String message) {
    StaticJsonDocument<256> doc;
    doc["command_id"] = commandId;
    doc["timestamp"] = millis();
    doc["status"] = status;
    doc["message"] = message;
    String output;
    serializeJson(doc, output);

    mqttClient.publish(TOPIC_RESPONSE, output.c_str(), false);  // QoS 1 via topic default

    Serial.print("Response sent: ");
    Serial.println(output);
}

void executeCommand(String action) {
    currentCommand = action;

    if (action == "forward") {
        moveForward();
    } else if (action == "backward") {
        moveBackward();
    } else if (action == "left") {
        turnLeft();
    } else if (action == "right") {
        turnRight();
    } else if (action == "stop") {
        stopMotors();
    } else {
        Serial.println("Unknown command: " + action);
        stopMotors();  // Safety: stop on unknown command
    }
}
```

Go Backend Complete Implementation

```go
// cmd/server/main.go

package main

import (
    "encoding/json"
    "fmt"
    "log"
    "os"
    "os/signal"
    "syscall"
    "time"

    mqtt "github.com/eclipse/paho.mqtt.golang"
)

type Telemetry struct {
    DeviceID      string `json:"device_id"`
    Timestamp     int64  `json:"timestamp"`
    Battery       int    `json:"battery"`
    DistanceFront int    `json:"distance_front"`
    Temperature   int    `json:"temperature"`
    CurrentAction string `json:"current_action"`
    WiFiRSSI      int    `json:"wifi_rssi"`
}

type Status struct {
    DeviceID  string `json:"device_id"`
    Status    string `json:"status"`
    Timestamp int64  `json:"timestamp"`
}

const (
    BrokerAddress       = "localhost:1883"
    ClientID            = "go-backend"
    TopicTelemetry      = "iot-car/+/telemetry"
    TopicStatus         = "iot-car/+/status"
    TopicFleetBroadcast = "iot-car/fleet/broadcast"
)

func main() {
    log.Println("□ IoT Car Backend Starting...")

    // Configure MQTT client
    opts := mqtt.NewClientOptions()
    opts.AddBroker(fmt.Sprintf("tcp://%s", BrokerAddress))
    opts.SetClientID(ClientID)
    opts.SetDefaultPublishHandler(onMessage)
    opts.SetOnConnectHandler(onConnect)
    opts.SetConnectionLostHandler(onConnectionLost)
    opts.SetAutoReconnect(true)
    opts.SetKeepAlive(30 * time.Second)

    // Create client
    client := mqtt.NewClient(opts)

    // Connect
    if token := client.Connect(); token.Wait() && token.Error() != nil {
        log.Fatal("Failed to connect:", token.Error())
    }

    log.Println("□ Connected to Mosquitto broker")

    // Wait for interrupt signal
    sigChan := make(chan os.Signal, 1)
    signal.Notify(sigChan, os.Interrupt, syscall.SIGTERM)
    <-sigChan

    log.Println("Shutting down...")
    client.Disconnect(250)
}

func onConnect(client mqtt.Client) {
    log.Println("□ MQTT Connected!")

    // Subscribe to topics
    topics := map[string]byte{
        TopicTelemetry: 1,  // QoS 1
        TopicStatus:    1,  // QoS 1
    }

    if token := client.SubscribeMultiple(topics, nil); token.Wait() && token.Error() != nil {
        log.Println("□ Subscribe error:", token.Error())
    } else {
        log.Println("□ Subscribed to:", TopicTelemetry, TopicStatus)
    }
}

func onConnectionLost(client mqtt.Client, err error) {
    log.Printf("□ Connection lost: %v", err)
}
```

```
func onMessage(client mqtt.Client, msg mqtt.Message) {
    topic := msg.Topic()
    payload := msg.Payload()

    log.Printf("□ Received [%s]: %s", topic, string(payload))

    // Route based on topic
    if matches(topic, "iot-car/+/telemetry") {
        handleTelemetry(payload)
    } else if matches(topic, "iot-car/+/status") {
        handleStatus(payload)
    }
}

func handleTelemetry(payload []byte) {
    var telemetry Telemetry
    if err := json.Unmarshal(payload, &telemetry); err != nil {
        log.Println("□ JSON parse error:", err)
        return
    }

    log.Printf("□ Telemetry from %s: Battery=%d%%, Distance=%dcm, Action=%s",
        telemetry.DeviceID,
        telemetry.Battery,
        telemetry.DistanceFront,
        telemetry.CurrentAction)

    // TODO: Save to database, trigger alerts, etc.
}

func handleStatus(payload []byte) {
    var status Status
    if err := json.Unmarshal(payload, &status); err != nil {
        log.Println("□ JSON parse error:", err)
        return
    }

    emoji := "□"
    if status.Status == "offline" {
        emoji = "□"
    }

    log.Printf("%s Device %s is %s", emoji, status.DeviceID, status.Status)
}

// Simple topic matcher (supports single-level wildcard +)
```

```
func matches(topic, pattern string) bool {
    // Simplified implementation
    // Production: Use proper MQTT topic matching library
    return true  // Placeholder
}
```

Android App MQTT Manager

```java
// MqttManager.java

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.*;

public class MqttManager {
    private static final String BROKER_URL = "tcp://192.168.1.100:1883";
    private static final int QOS = 1;

    private MqttAndroidClient mqttClient;
    private String deviceId;
    private MqttCallback callback;

    public MqttManager(Context context, String deviceId, MqttCallback callback) {
        this.deviceId = deviceId;
        this.callback = callback;

        String clientId = "android-" + System.currentTimeMillis();
        mqttClient = new MqttAndroidClient(context, BROKER_URL, clientId);
        mqttClient.setCallback(new MqttCallbackExtended() {
            @Override
            public void connectComplete(boolean reconnect, String serverURI) {
                Log.d("MQTT", "Connected!");
                subscribeToTopics();
                if (callback != null) callback.onConnectionSuccess();
            }

            @Override
            public void connectionLost(Throwable cause) {
                Log.e("MQTT", "Connection lost", cause);
                if (callback != null) callback.onConnectionLost();
            }

            @Override
            public void messageArrived(String topic, MqttMessage message) {
                String payload = new String(message.getPayload());
                Log.d("MQTT", "Message: " + topic + " = " + payload);
                if (callback != null) callback.onMessageReceived(topic, payload);
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {}
        });
    }

    public void connect() {
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(true);
        options.setAutomaticReconnect(true);
        options.setKeepAliveInterval(30);

        try {
            mqttClient.connect(options);
        } catch (MqttException e) {
            Log.e("MQTT", "Connection failed", e);
        }
    }

    private void subscribeToTopics() {
        try {
            String telemetryTopic = "iot-car/" + deviceId + "/telemetry";
            String statusTopic = "iot-car/" + deviceId + "/status";
            String responseTopic = "iot-car/" + deviceId + "/response";

            mqttClient.subscribe(telemetryTopic, QOS);
            mqttClient.subscribe(statusTopic, QOS);
            mqttClient.subscribe(responseTopic, QOS);

            Log.d("MQTT", "Subscribed to topics");
        } catch (MqttException e) {
            Log.e("MQTT", "Subscribe failed", e);
        }
    }

    public void sendCommand(String action) {
        String topic = "iot-car/" + deviceId + "/command";
        String commandId = "cmd-" + System.currentTimeMillis();

        JSONObject json = new JSONObject();
        try {
            json.put("id", commandId);
            json.put("timestamp", System.currentTimeMillis() / 1000);
            json.put("action", action);

            String payload = json.toString();
            mqttClient.publish(topic, payload.getBytes(), QOS, false);

            Log.d("MQTT", "Command sent: " + action);
        } catch (Exception e) {
            Log.e("MQTT", "Publish failed", e);
        }
    }
    public void disconnect() {
        try {
            mqttClient.disconnect();
        } catch (MqttException e) {
            Log.e("MQTT", "Disconnect failed", e);
        }
    }

    public interface MqttCallback {
        void onConnectionSuccess();
        void onConnectionLost();
        void onMessageReceived(String topic, String payload);
    }
}
```

Troubleshooting & Debugging

Common Issues

Problem	Cause	Solution
Connection refused	Wrong broker address	Check IP, use `ipconfig`
Connection timeout	Firewall blocking	Allow port 1883 in firewall
Authentication failed	Wrong credentials	Verify username/password
Messages not received	Topic mismatch	Print topics on both sides
QoS 1/2 not working	Broker config	Enable persistence in mosquitto.conf
Retained messages pile up	Not clearing old messages	Send empty payload with retain=true
High latency	Network congestion	Use QoS 0, reduce message size
Disconnects frequently	Keep-alive too short	Increase keep-alive interval

Debugging Tools

1. Mosquitto Command Line Tools

```
# Subscribe to all topics
mosquitto_sub -h localhost -t "#" -v

# Subscribe with QoS
mosquitto_sub -h localhost -t "iot-car/#" -q 1 -v

# Publish test message
mosquitto_pub -h localhost -t "test/topic" -m "Hello MQTT"

# Publish with QoS and retain
mosquitto_pub -h localhost -t "test/status" -m "online" -q 1 -r

# Clear retained message
mosquitto_pub -h localhost -t "test/status" -m "" -r
```

2. MQTT Explorer (GUI Tool)

Download: http://mqtt-explorer.com/ (http://mqtt-explorer.com/)

Features:

- Visual topic tree
- Message history
- Publish/subscribe
- Retained message management
- Connection statistics

3. Enable MQTT Debug Logging

ESP32:

```
// Enable PubSubClient debug
#define MQTT_MAX_PACKET_SIZE 512
#define MQTT_DEBUG
```

Mosquitto:

```
# mosquitto.conf
log_type all
log_dest file /var/log/mosquitto/mosquitto.log
log_dest stdout
```

Go:

```
mqtt.DEBUG = log.New(os.Stdout, "[DEBUG] ", 0)
mqtt.ERROR = log.New(os.Stdout, "[ERROR] ", 0)
```

Network Diagnostics

```
# Check if broker is listening
netstat -an | findstr :1883

# Test connectivity
Test-NetConnection -ComputerName 192.168.1.100 -Port 1883

# Ping broker
ping 192.168.1.100

# Check firewall rules
Get-NetFirewallRule | Where-Object {$_.DisplayName -like "*mosquitto*"}
```

Performance Optimization

Message Size Optimization

```
Rule of Thumb:
• Keep messages under 1KB for most IoT applications
• Use compact JSON (no whitespace)
• Abbreviate field names (but keep readable)
• Use binary formats only if bandwidth-critical
```

Before:

```
{
  "device_identifier": "car-001",
  "timestamp_unix_epoch": 1706169600,
  "battery_percentage": 85,
  "distance_sensor_front_centimeters": 45
}
```

Size: ~150 bytes

After:

```
{"id":"car-001","ts":1706169600,"bat":85,"dist":45}
```

Size: ~54 bytes (64% reduction!)

Connection Optimization

```
// Optimize keep-alive
mqttClient.setKeepAlive(60);  // Reduce to 30-60s for faster detection

// Increase buffer size for large messages
mqttClient.setBufferSize(1024);

// Use clean session carefully
// false = broker remembers subscriptions (good for devices that sleep)
// true = fresh start every connection (good for testing)
```

Bandwidth Usage

```
Telemetry Example:
• Message size: 100 bytes
• Frequency: 1 message/second
• Bandwidth: 100 bytes/s = 0.8 Kbps

For 100 devices:
• Total bandwidth: 80 Kbps (negligible!)

MQTT is extremely efficient for IoT! □
```

Advanced Topics

MQTT 5.0 Features (Optional)

MQTT 5.0 introduces new features (not used in this project, but good to know):

Feature	Description
User Properties	Custom key-value metadata in messages
Reason Codes	Detailed error reporting
Request/Response	Built-in correlation for command-response
Topic Aliases	Reduce bandwidth by using numeric IDs
Message Expiry	Auto-delete messages after timeout
Shared Subscriptions	Load balance across multiple subscribers

MQTT over WebSocket

For browser-based clients:

```
// JavaScript client
const client = new Paho.MQTT.Client(
    "ws://192.168.1.100:9001/mqtt",  // WebSocket URL
    "web-client-" + Date.now()
);

client.connect({
    onSuccess: () => {
        console.log("Connected!");
        client.subscribe("iot-car/+/telemetry");
    }
});
```

MQTT Bridge (Multi-Broker)

Connect multiple brokers:

```
# mosquitto.conf - Bridge to cloud broker
connection cloud-bridge
address mqtt.cloud-provider.com:8883
topic iot-car/# out 1  # Forward all iot-car topics
bridge_cafile /etc/ssl/certs/ca-certificates.crt
```

Summary & Best Practices

□ DO's

- □ Use descriptive topic hierarchies (`domain/device/metric`)
- □ Include timestamps in all messages
- □ Use QoS 1 for commands, QoS 0 for high-frequency data
- □ Implement Last Will Testament for presence detection
- □ Use retained messages for status/configuration
- □ Validate and sanitize all incoming messages
- □ Log errors and connection issues
- □ Test with `mosquitto_sub` and `mosquitto_pub`
- □ Use unique client IDs
- □ Enable TLS for production deployments

□ DON'Ts

- □ Don't use QoS 2 unless absolutely necessary
- □ Don't publish large files over MQTT (use HTTP instead)
- □ Don't use spaces or special characters in topics

- □ Don't expose broker to internet without authentication
- □ Don't use wildcard `#` unnecessarily (subscribing to all topics)
- □ Don't ignore connection errors
- □ Don't forget to set keep-alive appropriately
- □ Don't publish retained messages that should be temporary

Further Reading

Official Documentation

- **MQTT 3.1.1 Specification:** https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html)
- **MQTT 5.0 Specification:** https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html)
- **Eclipse Mosquitto:** https://mosquitto.org/documentation/ (https://mosquitto.org/documentation/)
- **Eclipse Paho (Clients):** https://www.eclipse.org/paho/ (https://www.eclipse.org/paho/)

Books

- "MQTT Essentials" by HiveMQ (free online guide)
- "Building Internet of Things with the Arduino" by Charalampos Doukas

Tools

- **MQTT Explorer:** http://mqtt-explorer.com/ (http://mqtt-explorer.com/)
- **MQTTX:** https://mqttx.app/ (https://mqttx.app/)
- **HiveMQ MQTT CLI:** https://hivemq.github.io/mqtt-cli/ (https://hivemq.github.io/mqtt-cli/)

Conclusion

MQTT is the **industry standard** for IoT communication because it's:

- **Lightweight** - Runs on tiny devices
- **Reliable** - QoS levels ensure delivery
- **Scalable** - Pub/sub supports millions of devices
- **Simple** - Easy to understand and implement

Your **IoT Car project** uses MQTT exactly as industry does:

- ESP32 → Broker → Backend/App (telemetry streaming)
- App → Broker → ESP32 (command-response)
- Retained messages for status
- Last Will Testament for presence detection
- JSON for interoperability

You're building production-grade architecture! □

This guide covers MQTT from fundamentals to production patterns. Use it as a reference throughout your IoT journey!

Version: 1.0

Last Updated: January 25, 2026

Project: IoT Car MVP

Author: IoT Car Development Team