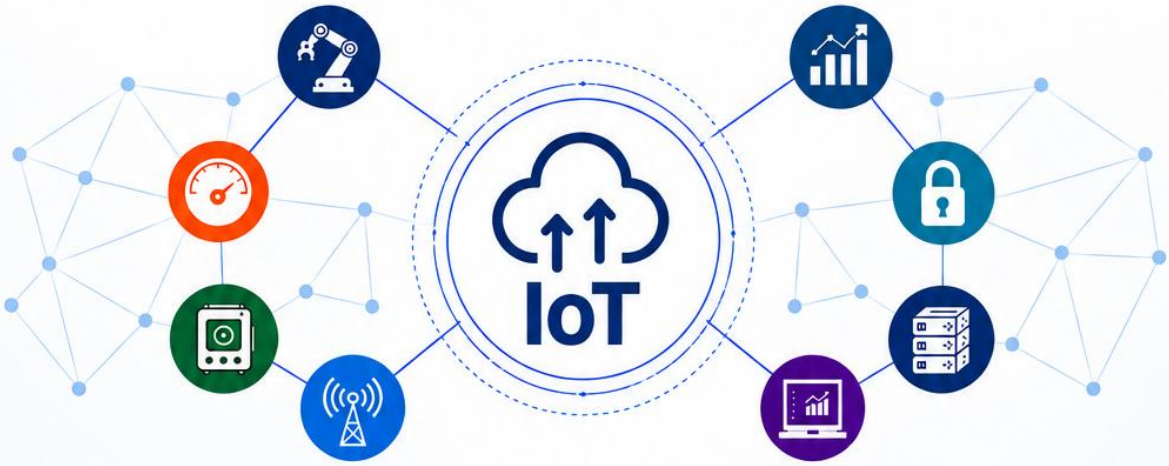
FROM DEVICE TO CLOUD. FROM DATA TO INSIGHT.

IoT & System Integration

A Technical Guide to Connecting IoT Devices, Protocols & Networks

DEEPAK KUMAR



Modbus | BACnet | M-Bus | Wireless M-Bus | LoRaWAN | NB-IoT

Bridge the gap between field devices and intelligent applications.

Design | Connect | Integrate | Deliver

About This Guide:

This guide is written from an **IoT System Integration perspective**, with a focus on understanding how devices, interfaces, protocols, gateways, networks, and platforms work together to form an end-to-end IoT solution. Some technologies are discussed based on practical project experience, while others are included at an overview level to provide broader awareness of technologies that a System Integrator may encounter.
This guide is intended as a **technical learning and reference resource**. It does not replace official protocol specifications, standards, product manuals, or vendor documentation.

Author's Note

This guide has been prepared by combining practical system integration experience with reference from technical documentation, industry references, and AI-assisted tools used for content organization and drafting. The content has been reviewed and selected for relevance from a System Integrator's perspective.

As technologies, standards, and product implementations continue to evolve, readers are encouraged to verify implementation-specific details with the latest official standards and manufacturer documentation.

Who Should Read This Guide

This guide is useful for **IoT engineers, System Integrators, solution engineers, technical professionals, and beginners** who want to understand how different technologies are connected to build complete IoT solutions.

Table of Contents:

1. **Introduction to IoT**:

   1.1 What is IoT?
   1.2 Evolution of IoT
   1.3 The Present Technology Era
   1.4 M2M, IoT and IIoT
   1.5 IoT Ecosystem
   1.6 System Integrator

2. **Understanding IoT Devices**:

   2.1 Introduction
   2.2 Sensors
   2.3 Actuators
   2.4 Meters
   2.5 Controllers and PLCs
   2.6 IoT Gateway
   2.7 Key Device Specifications for System Integrators
   2.8 IoT Integrator Device Check List
   2.9 System Integrator's Approach

3. **Industrial Protocols**:

   3.1 Protocol, Interface and Network
   3.2 Modbus
   3.3 BACnet
   3.4 M-Bus and Wireless M-Bus
   3.5 DLMS/COSEM
   3.6 CAN
   3.7 Profibus
   3.8 Profinet
   3.9 Industrial Ethernet
   3.10 OPC UA

4. **Wireless Connectivity**:

   4.1 Understanding Wireless Connectivity
   4.2 Wi-Fi
   4.3 Bluetooth Low Energy (BLE)
   4.4 Zigbee
   4.5 RFID
   4.6 NFC
   4.7 Wi-Fi vs BLE vs Zigbee vs RFID vs NFC
   4.8 How a System Integrator Should Select Wireless Technology
   4.9 System Integrator's View

5. **LPWAN & LoRaWAN:**

   5.1 Introduction to LPWAN
   5.2 LoRa vs LoRaWAN
   5.3 LoRaWAN Architecture
   5.4 Uplink and Downlink Communication
   5.5 LoRaWAN Device Activation
   5.6 LoRaWAN Device Classes
   5.7 Adaptive Data Rate – ADR

5.8 Data Rate and Spreading Factor
5.9 RSSI and SNRC
5.10 Frequency Plans
5.11 LoRaWAN Troubleshooting Approach
5.12 Common LoRaWAN Problems
5.13 The System Integrator's View

6. **NB-IoT**

6.1 What is NB-IoT
6.2 Control Plane vs User Plane
6.3 NB-IoT Deployment
6.4 NB-IoT Band
6.5 PDN
6.6 PSM and eDRX
6.7 Coverage Extension and Power Class
6.8 Release 13,14,15 Features
6.9 NB-IoT Integration with Cloud

7. **IoT Application Protocols**:

7.1 MQTT
7.2 HTTP/HTTPS
7.3 REST API
7.4 WebSocket
7.5 JSON
7.6 TCP and UDP

8. **Networking for System Integrators**:

8.1 OSI/TCP-IP
8.2 Ethernet
8.3 IP addressing
8.4 Subnetting
8.5 TCP/UDP
8.6 Ports
8.7 DNS
8.8 DHCP
8.9 Firewall and VPN

9. **System Integration Solutions**

9.1 LoRaWAN Solution
9.2 Modbus to LoRaWAN Solution
9.3 LoRaWAN to BACnet Solution
9.4 Wireless M-Bus Solution

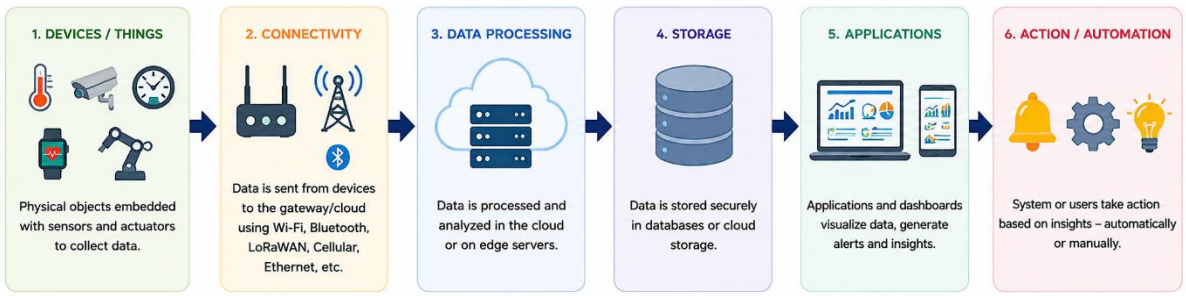
Disclaimer

Chapter 1

Introduction to IoT

1.1 What is IoT?

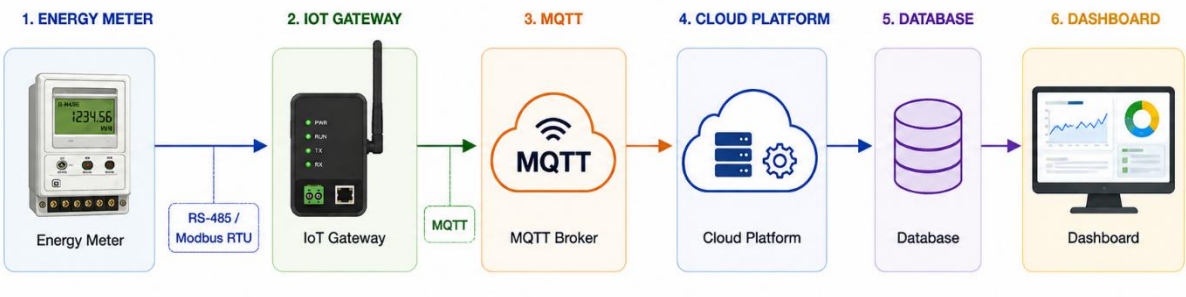
The **Internet of Things (IoT)** is the concept of connecting physical devices to digital systems so that data can be collected, exchanged, monitored, and used for further action. These physical devices can be as simple as a temperature sensor or as complex as an industrial machine, energy meter, PLC, HVAC system, vehicle, or production line.



The term *Internet of Things* can sometimes create the impression that every IoT device must be directly connected to the Internet. In practice, this is not always the case. A field device may communicate through RS-485, LoRaWAN, Wireless M-Bus, BLE, or another local communication technology. A gateway or another intermediate system then carries the data towards an IP network or application.

Consider an energy meter installed inside a factory. The meter may communicate using **Modbus RTU over RS-485**. An IoT gateway reads the required Modbus registers and publishes the values to a cloud platform using **MQTT over Ethernet or a cellular connection**. The cloud application stores and displays the data for the customer.

The complete communication path may therefore look like this:



This example highlights an important point: **IoT is not a single technology**. A practical IoT solution is built by combining several technologies, protocols, devices, and software systems.

From a System Integrator's point of view, the challenge is usually not to invent these technologies. The challenge is to make them work together.

1.2 Evolution of IoT

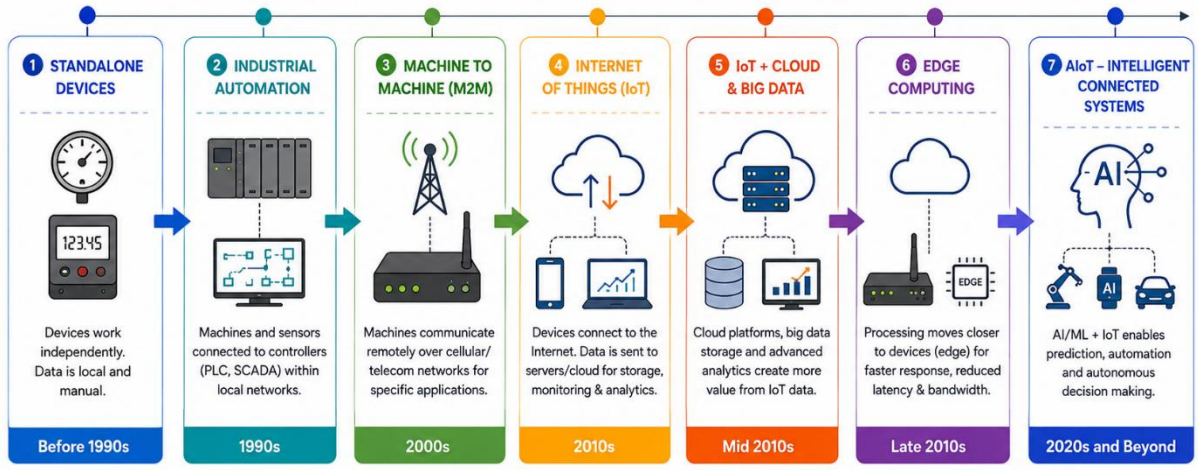
IoT is the result of a gradual evolution rather than a single invention. Industrial systems were exchanging data long before the term IoT became popular.

Early automation systems connected sensors and machines with controllers such as PLCs. Technologies including **RS-232, RS-485, Modbus, CAN and PROFIBUS** allowed equipment on a factory floor to communicate. Most of these systems, however, were designed to operate within a local environment.

Telecommunication networks later made it practical to communicate with equipment installed at remote locations. GSM and GPRS, for example, were widely used for applications such as remote meter reading, fleet tracking, ATMs and telemetry. This type of communication became commonly associated with **Machine-to-Machine (M2M)** systems.

As Internet connectivity became widespread and cloud computing matured, the architecture expanded. Instead of sending information from one machine to one dedicated server, organizations could connect large numbers of heterogeneous devices to common platforms and make the data available to multiple applications.

The development can broadly be viewed as:



Today, IoT systems are increasingly combined with edge computing, analytics and artificial intelligence. The purpose is no longer limited to collecting data remotely. Modern systems are expected to identify abnormal conditions, generate alerts, predict failures and, where appropriate, automate actions.

1.3 The Present Technology Era

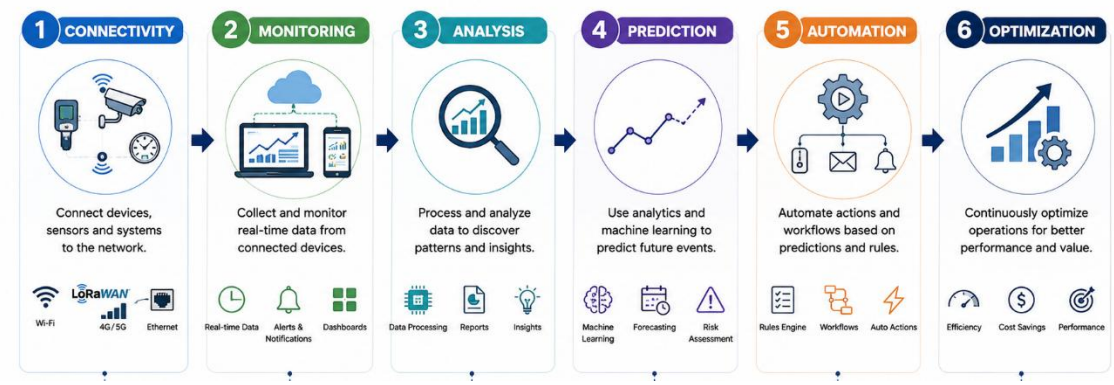
There is no universally accepted technical name for the current technology age. Terms such as **Digital Age, Industry 4.0, AI Era and Intelligent Age** are used in different contexts.

For industrial IoT, the concept of **Industry 4.0** is particularly relevant. It represents the increasing integration of industrial equipment, automation, connectivity, data systems and intelligent software.

From a System Integration perspective, today's environment can be viewed as an era of **intelligent connected systems**. IoT, cloud computing, edge computing, LPWAN, 5G, APIs, data analytics and AI increasingly operate as parts of the same solution.

The progression is therefore moving from simply connecting equipment toward:

Connectivity → Monitoring → Analysis → Prediction → Automation → Optimization



This transition has also increased the importance of System Integration because organizations often have to connect modern IoT technologies with equipment and applications that have already been operating for many years.

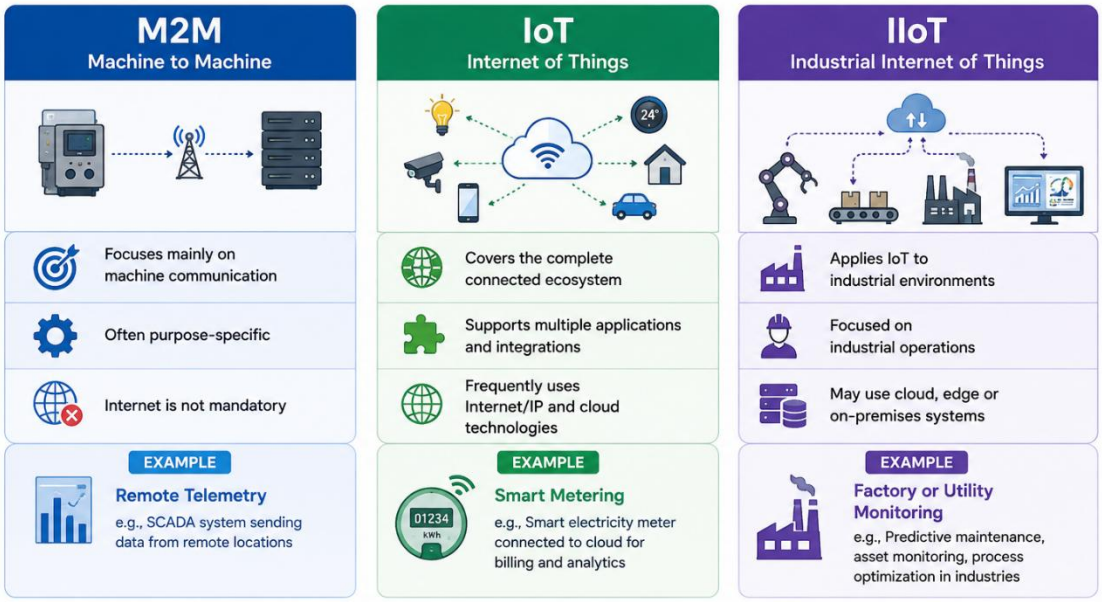
1.4 M2M, IoT and IIoT

The terms **M2M, IoT and IIoT** are closely related and are sometimes used interchangeably, but there are useful differences between them.

Machine-to-Machine (M2M) refers primarily to communication between machines or between a machine and a central system, normally with little human involvement. A remote meter sending its reading through a cellular modem to a central server is a typical example. M2M systems are often designed for a specific purpose and do not necessarily depend on Internet or cloud technologies.

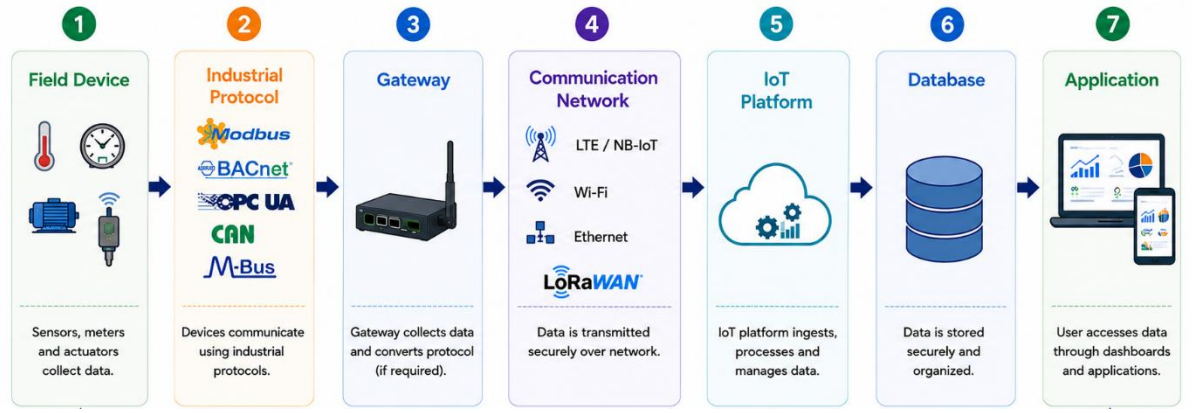
Internet of Things (IoT) is a broader concept. It includes the physical devices as well as connectivity, gateways, platforms, databases, APIs, applications and device-management functions. The same device data may be consumed by several applications instead of remaining inside one dedicated system.

Industrial Internet of Things (IIoT) applies IoT concepts specifically to industrial and operational environments such as manufacturing, utilities, energy, oil and gas, building automation and transportation. IIoT systems normally have stronger requirements for reliability, security, availability and interoperability because a communication or system failure may affect an operational process.



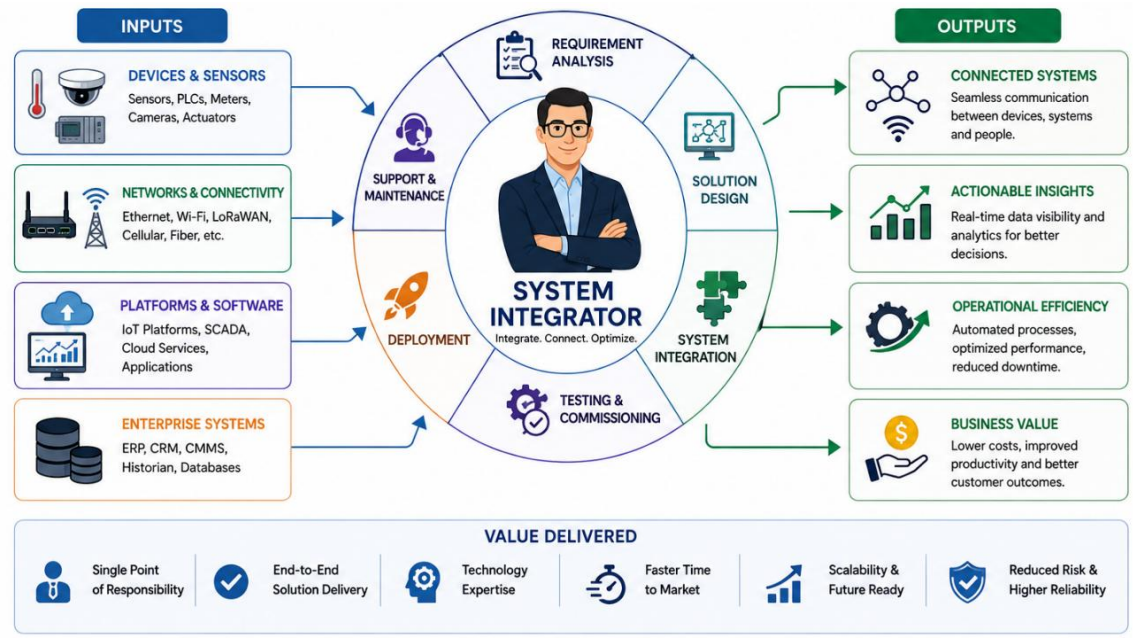
1.5 IoT Ecosystem

An **IoT ecosystem** is the complete environment required to move information between the physical world and the applications that use that information.



No single vendor necessarily supplies all these components. This is one of the main reasons System Integration is required.

1.6 System Integrator



A **System Integrator (SI)** is a person or organization that brings together different **devices, hardware, software, communication networks, protocols, platforms, and applications** and makes them work as one complete system.

For example:

Energy Meter → RS-485 / Modbus RTU → IoT Gateway → MQTT → Cloud Platform → REST API → Customer Application

Each component may work perfectly on its own. The System Integrator's responsibility is to ensure that **they communicate correctly with each other and that the complete solution meets the customer's requirement**.

A System Integrator can therefore be considered the **technical bridge between devices, communication networks, IT infrastructure, and business applications**.

1.6.1 Why is a System Integrator Important?

Modern IoT and industrial systems are heterogeneous. A typical project may involve equipment from several vendors and technologies developed for completely different purposes.

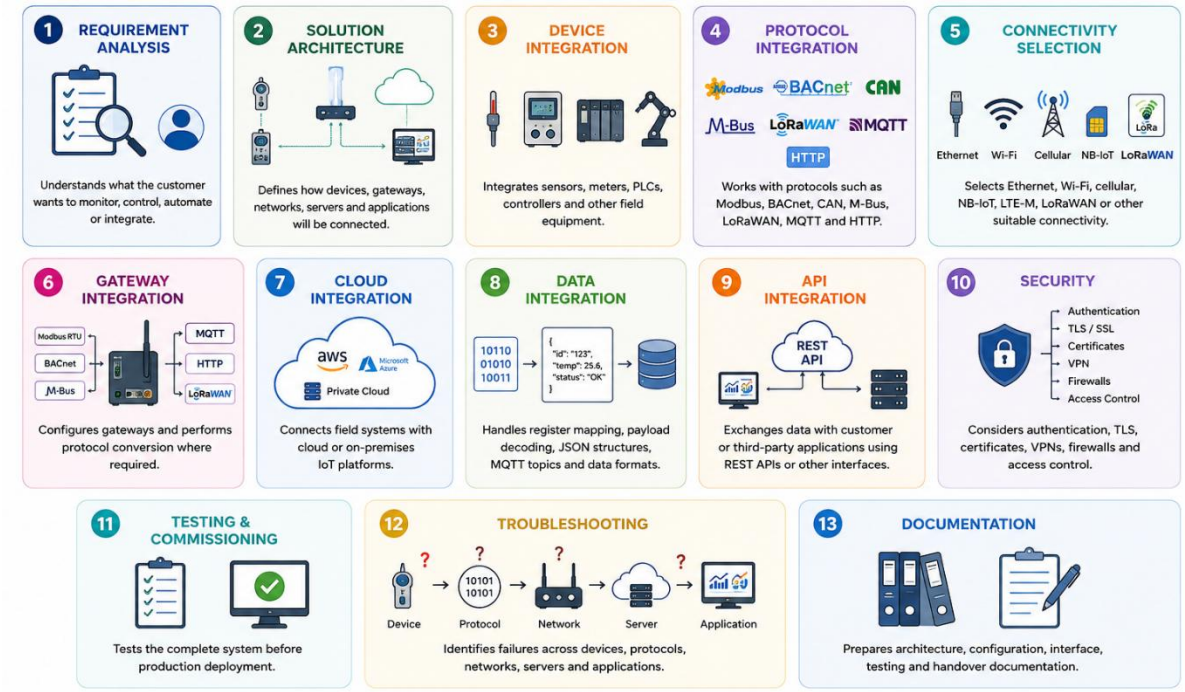
The challenge is not simply connecting all these devices to the Internet. The real challenge is making these systems **exchange the correct data, in the correct format, securely and reliably**.

This is where the System Integrator becomes important.

Without proper integration, an energy meter may successfully provide data through Modbus, and a cloud application may work correctly, but the customer still receives nothing because the gateway configuration, register mapping, MQTT topic, firewall rule, payload format, or API integration is incorrect.

The System Integrator looks at the **complete communication chain**, rather than only one product.

1.6.2 Role of a System Integrator in IoT



Chapter 2

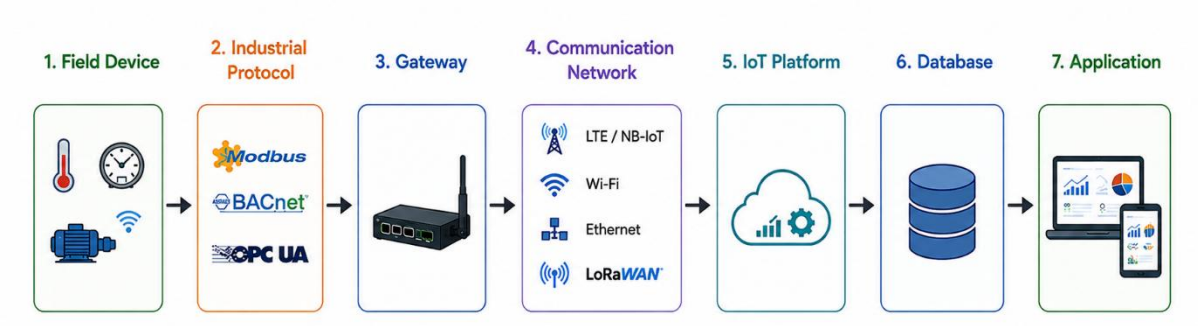
Understanding IoT Devices

2.1 Introduction

IoT systems interact with the physical world through field devices that measure parameters, monitor equipment, control processes, record consumption, and communicate data to IoT platforms.
For an **IoT System Integrator**, it is not necessary to understand the internal electronic design of every device.
The main requirement is to understand:
Device Function → Power → Interface → Protocol → Data → Network → IoT Platform

The main device categories are **Sensors, Actuators, Meters, Controllers, PLCs, and IoT Gateways**.

A typical IoT architecture is:



2.2 Sensors

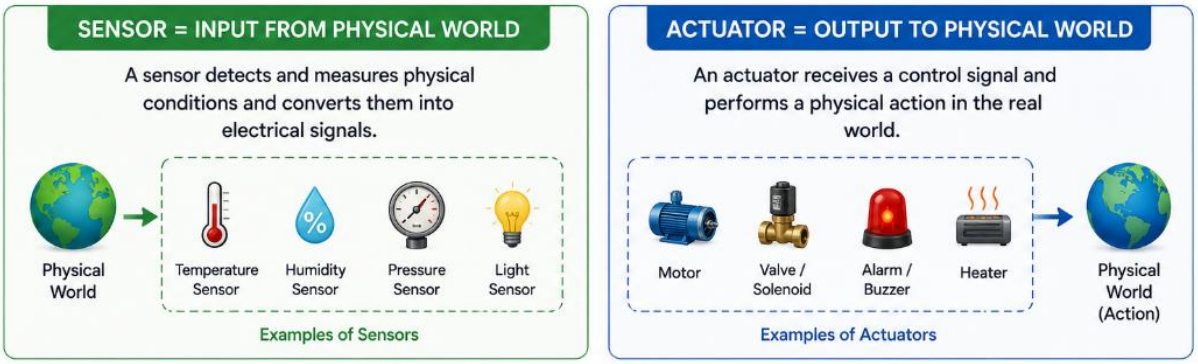
A **sensor** measures a physical parameter and converts it into an electrical or digital signal.

Parameter	Example Sensor	Typical Output
Temperature	RTD / Temperature Sensor	• PT100 • 4–20 mA • Modbus
Pressure	Pressure Transmitter	• 4–20 mA • 0–10 V
Humidity	RH Sensor	• Analog • RS485
Level	Ultrasonic / Radar Sensor	• 4–20 mA • Modbus
Motion	PIR Sensor	• Digital
Air Quality	CO_2 / PM Sensor	• RS485 • UART
Flow	Flow Sensor	• Pulse • 4–20 mA • Modbus
Water Meter	Water Meter	• Pulse • LoRAWAN • Modbus • MBUS

A System Integrator should check the sensor's **measurement range, accuracy, resolution, output type, power requirement, and environmental suitability** before selection and integration.

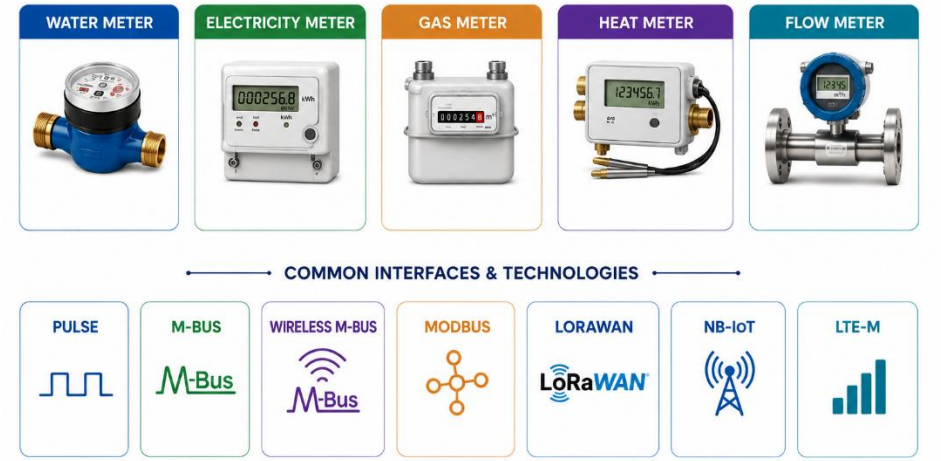
2.3 Actuators

An **actuator** performs a physical action based on a control command.



2.4 Meters

Meters measure consumption or accumulated quantities.



Example architecture:

Energy Meter → Modbus → IoT Gateway → MQTT/HTTP → IoT Platform

Energy Meter → NBIOT→ RAN → IoT Platform

***Detail architecture will be explained in upcoming chapters.*

2.5 Controllers and PLCs

A **controller** receives information from sensors, executes predefined logic, and generates an output.

Example:
Level Sensor → Controller → Pump
Logic may be:
Level < 20% → Pump ON
Level > 90% → Pump OFF
Controllers are commonly used in HVAC, pumping, building automation, and industrial machinery.

A **PLC (Programmable Logic Controller)** is an industrial computer used for automation and real-time process control.

Typical architecture:
Sensors / Meters → PLC → IoT Gateway → IoT Platform

PLCs commonly provide:

- Digital Input/Output – DI/DO
- Analog Input/Output – AI/AO
- RS232/RS485
- Ethernet

An IoT integrator does not necessarily need advanced PLC programming knowledge but should understand **I/O mapping, addressing, communication configuration, register mapping, and data extraction**.

2.6 IoT Gateways

An **IoT Gateway** connects field devices with an IoT platform or cloud environment.

Typical architecture:

Field Device → Industrial Protocol → IoT Gateway → Network → IoT Platform

A gateway may provide:

- **Protocol Conversion:** Modbus RTU → LoRAWAN/MQTT
- **Data Aggregation:** Collect data from single or multiple devices
- **Edge Processing:** Perform local calculations/filtering
- **Data Buffering:** Store data during network failure
- **Security:** TLS, authentication, certificates, VPN
- **Cloud Connectivity:** AWS, Azure, or private IoT platforms

Example:

Energy Meter → RS485/Modbus → Gateway → LTE/Ethernet → MQTT/TLS → IoT Platform

LoRaWAN Gateway
A LoRaWAN gateway mainly forwards LoRaWAN radio packets between end devices and a LoRaWAN Network Server (LNS).
An integrator should understand:
Frequency Band | Antenna | Channel Configuration | Internet Connectivity | Packet Forwarder | LNS Configuration

2.7 Key Device Specifications for System Integrators

Before integrating a new device, review its **datasheet, user manual, and communication protocol document**.

1. Device Function: Understand what the device measures, monitors, or controls.

2. Electrical Specifications: Check: Supply Voltage | Power Consumption | Battery | Wiring | Connectors

3. Physical Interface: Determine how the device physically connects:

- RS232 | RS485 | Ethernet | USB | GPIO | Analog | Digital | Pulse
  Remember:
- RS485 = Physical/Electrical Interface
  Modbus RTU = Communication Protocol

4. Communication Protocol: Identify how information is exchanged:

- Modbus | BACnet | M-Bus | Wireless M-Bus | OPC UA | MQTT | HTTP | LoRaWAN | NB-IoT | LTE-M
- Both communicating systems must support compatible protocols.

5. Communication Parameters

- For Modbus RTU, check: Baud Rate | Data Bits | Parity | Stop Bits | Slave ID
- Example: 9600 | 8 | None | 1 | Slave ID 10
- For IP devices, check: IP Address | DHCP/Static IP | Subnet | Gateway | DNS | TCP/UDP Port

6. Data Format

- The integrator must understand how raw device data is converted into useful information.
- **Modbus:** Register → Data Type → Scaling → Unit
- **MQTT:** Topic → JSON Payload → Parameter → Timestamp
- **LoRaWAN:** FPort → Binary Payload → Decoder → Engineering Value

7. Network Requirements

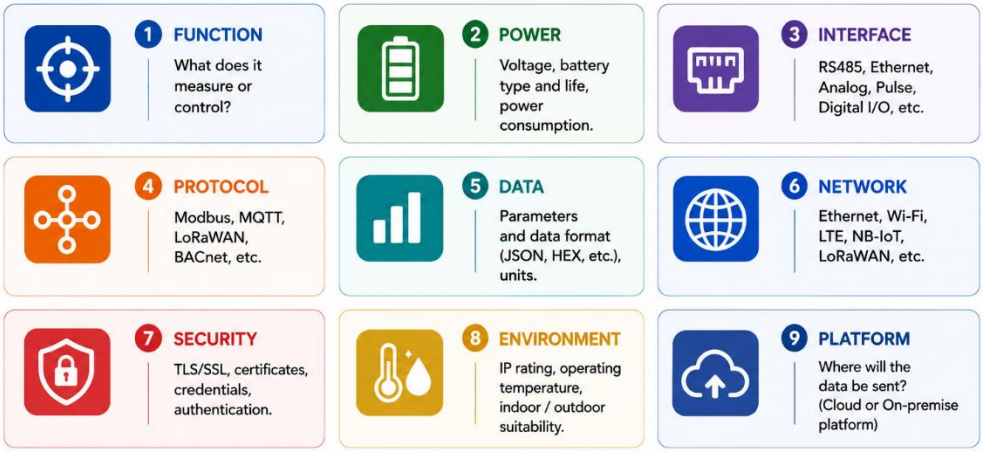
- Understand the complete communication path.
- **Ethernet:** Device → LAN → Firewall → Internet → Server
- **Cellular:** Device → SIM/eSIM → NB-IoT/LTE → APN → IoT Server
- **LoRaWAN:** Device → Gateway → Internet → LNS → Application
- Check firewall ports, VPN, APN, static IP, internet connectivity, and inbound/outbound
- communication requirements.

8. Environmental Requirements

- Check: IP Rating | Temperature | Humidity | Indoor/Outdoor | Mounting | Cable Length | Antenna | RF Frequency
- Correct installation is as important as protocol compatibility.

2.8 IoT Integrator Device Checklist

When receiving a new device, use this checklist:



2.9 System Integrator's Approach

A System Integrator should not begin with the question, **"Which IoT technology should I use?"** The first step is to understand the device, its interfaces, and the path its data must follow.

Each stage answers a different engineering question:

What is the device? → How can I access its data? → What language does it speak? → Is protocol conversion required? → How will the data be transported? → Where will it be processed and stored? → How will the user consume it?

Example 1 – Energy Meter
Energy Meter → RS485 → Modbus RTU → IoT Gateway → LTE → MQTT → IoT Platform
Here, RS485 is the physical interface, Modbus RTU is the field protocol, the gateway reads and converts the data, LTE provides connectivity, and MQTT carries the data to the IoT platform.

Example 2 – LoRaWAN Water Meter
Water Meter → LoRaWAN → LoRaWAN Gateway → Ethernet/LTE → LNS → IoT Platform/Application
In this architecture, the meter does not communicate directly with the application. The LoRaWAN Gateway receives the radio packets, the LNS manages the LoRaWAN network, and the processed device data is then delivered to the required platform or application.

Think in Interfaces, Not Products

One of the most useful skills for a System Integrator is the ability to work with a device that has **never been seen before**.

Instead of trying to memorize thousands of meters, sensors, controllers, gateways, and platforms, break every new device into a few engineering questions:

What does it do? → What powers it? → What interface does it expose? → Which protocol does it use? → What data does it provide? → What network is available? → Where must the data finally go?

Once these questions are answered, an unfamiliar product becomes much easier to integrate.

The Important Difference

A device being **connected** does not mean it is **integrated**.

For example, a gateway may successfully communicate with an energy meter over Modbus, but the solution is still incomplete if:

- the wrong registers are mapped,
- engineering units or scaling are incorrect,
- timestamps are inconsistent,
- MQTT topics or payloads do not match the platform,
- network/firewall rules block communication, or
- the application cannot interpret the received data.

Therefore, a System Integrator should validate not only **communication**, but also the **meaning, integrity, security, and usability of the data from source to destination**.

Answering these questions provides the foundation for designing a reliable, secure and scalable end-to-end IoT solution.

Chapter 3

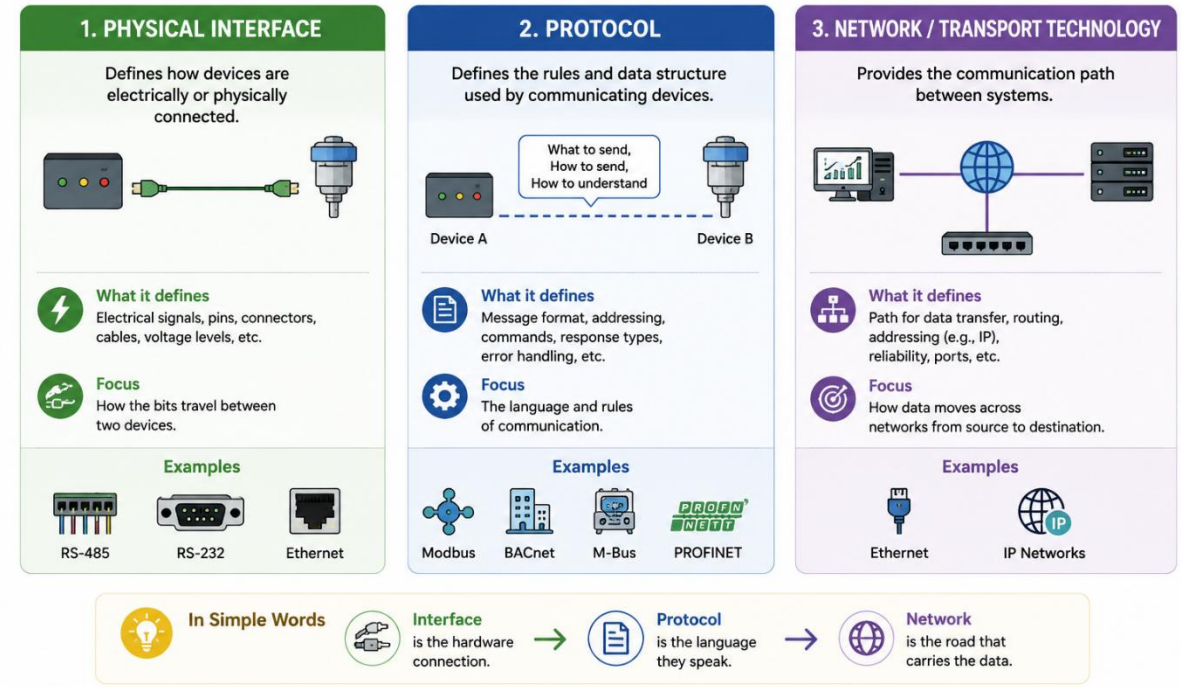
Industrial Protocols

Note: *The protocols in this chapter are covered from a System Integrator's viewpoint. The level of detail varies depending on their relevance to common IoT and industrial integration scenarios. For implementation, always refer to the latest official protocol specifications and device manufacturer documentation.*

Industrial systems contain equipment from many manufacturers and generations. A single site may have an energy meter using **Modbus RTU**, a building controller using **BACnet**, a utility meter using **M-Bus**, a PLC communicating through **PROFINET**, and an IoT gateway sending data to a platform using **OPC UA**.
For a System Integrator, the objective is not to become a protocol developer. The important skill is to understand where a protocol is used, how devices are physically connected, how data is addressed, and what is required to integrate that data with another system.

3.1 Protocol, Interface and Network — Know the Difference

These terms are often used interchangeably, but they are not the same.



3.2 Modbus

Modbus is one of the most widely encountered industrial communication protocols. It is commonly found in energy meters, power meters, PLCs, VFDs, temperature controllers, sensors and other industrial equipment.

The two forms most relevant to a System Integrator are **Modbus RTU** and **Modbus TCP**.

Modbus RTU: Modbus RTU is commonly implemented over an **RS-485 serial network**.

Several devices can share the same RS-485 bus, with each device configured with a unique **Slave ID/Unit Address**.

Important parameters include:

Slave ID | Baud Rate | Parity | Stop Bits | Register Address | Function Code | Data Type

The device manufacturer normally provides a **Modbus Register Map**.

For example:

Register	Parameter	Data Type	Unit
40001	Voltage	UInt16	V
40002	Current	UInt16	A
40003–40004	Energy	Float32	kWh

The exact addresses and formats depend on the manufacturer.

***A System Integrator should therefore never assume that two Modbus meters use the same register map.*

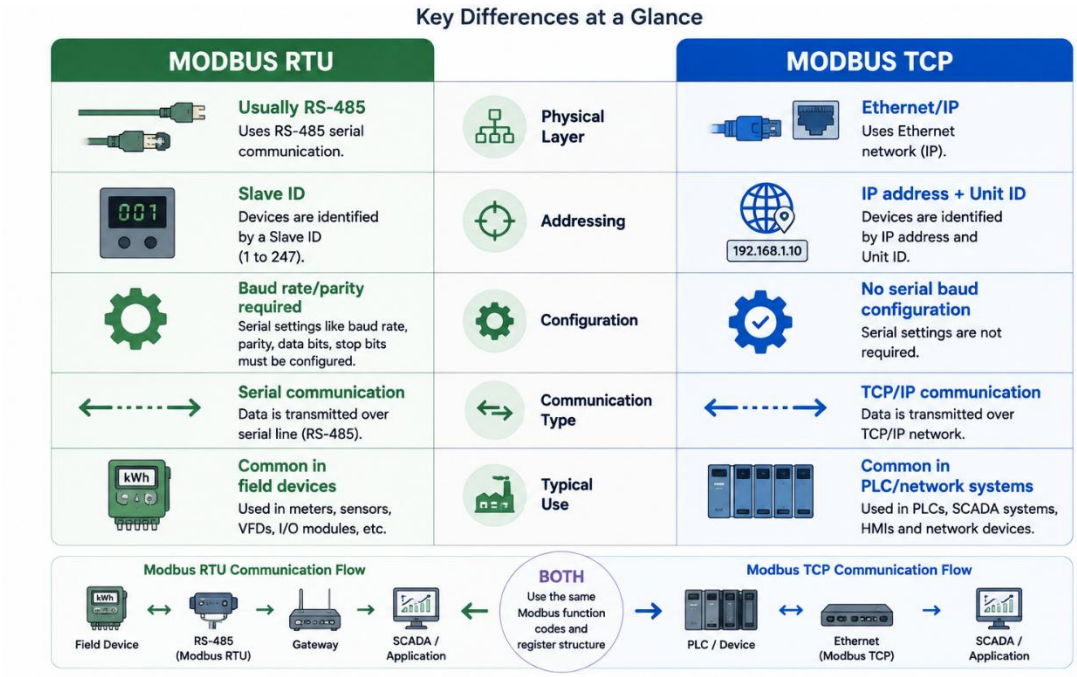
Modbus TCP: Modbus TCP carries Modbus communication over **Ethernet/IP**.

Instead of serial parameters such as baud rate and parity, the integrator mainly works with:

IP Address | TCP Port | Unit ID | Register Address

The standard Modbus TCP port is **502**.

RTU vs TCP



For an SI, Modbus troubleshooting should follow the communication chain:

Wiring → Serial/IP Settings → Device Address → Register → Data Type → Scaling

3.3 BACnet

BACnet – Building Automation and Control Networks is widely used in **BMS and building automation**.
Typical systems include:
HVAC | AHU | Chiller | VAV | Lighting | Energy Monitoring | Building Controllers

The two forms commonly encountered are:
BACnet MS/TP:

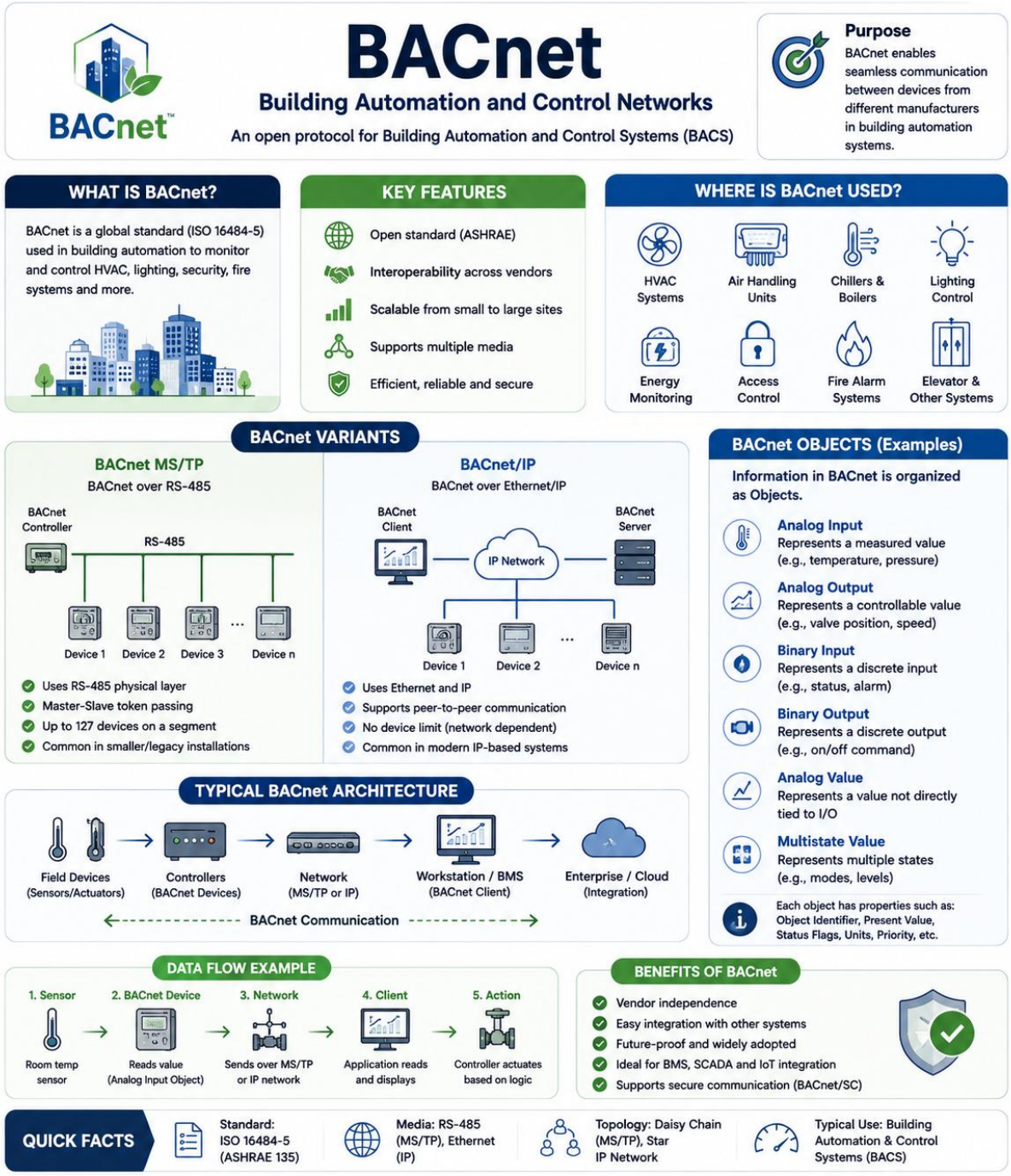
BACnet MS/TP typically operates over **RS-485**.
Typical architecture:
HVAC Controller → RS-485 → BACnet MS/TP → BMS Controller/Gateway

Important integration parameters include:
MAC Address | Device Instance | Baud Rate | Max Master | BACnet Objects

BACnet/IP:

BACnet/IP operates over an IP/Ethernet network.
Typical architecture:
Building Controller → Ethernet → BACnet/IP → BMS
Instead of reading register numbers as in Modbus, BACnet generally represents information as objects.
Examples include: **Analog Input | Analog Output | Binary Input | Binary Output | Analog Value**

For example, room temperature might appear as:
Analog Input → Present Value → 24.5 °C
This object-oriented structure is one of the major differences between BACnet and Modbus.



3.4 M-Bus and Wireless M-Bus

M-Bus:

M-Bus (Meter-Bus) is designed primarily for reading consumption meters.
It is commonly used with: **Water Meters | Heat Meters | Gas Meters | Electricity Meters**
Typical architecture:
Meters → M-Bus → M-Bus Master/Gateway → IoT Platform
Unlike RS-485 Modbus, M-Bus has its own electrical characteristics. Therefore, an RS-485 port should not be directly connected to an M-Bus device.
A System Integrator normally needs to understand:
Primary/Secondary Address | Baud Rate | Meter Data Records | M-Bus Master

Wireless M-Bus:

Wireless M-Bus (wM-Bus) provides wireless communication for metering applications.
Typical architecture: **Water Meter → Wireless M-Bus → Receiver/Gateway → Platform**
It is commonly used where installing communication cables to individual meters would be difficult.
From an integration perspective, important considerations include:
Frequency | Wireless Mode | Encryption Key | Meter ID | Gateway Compatibility
Wireless M-Bus and wired M-Bus serve similar metering applications but use different communication media.

3.5 DLMS/COSEM

DLMS/COSEM is widely associated with **smart electricity metering and utility AMI systems**.
DLMS defines communication services, while COSEM provides a standardized model for representing meter information.
Typical architecture:
Smart Electricity Meter → Communication Network → Head-End System
Communication may use technologies such as cellular, RF, PLC or other supported media depending on the meter and utility architecture.
Unlike simple register-based systems, DLMS/COSEM data is commonly identified using **OBIS codes**.
For example, OBIS codes may represent:
Voltage | Current | Active Energy | Maximum Demand | Billing Information
For an IoT/System Integrator, important areas are:
Meter Address | OBIS Code | Communication Profile | Authentication | Encryption/Security Keys
DLMS/COSEM is particularly important when integrating with utility smart-metering systems.

3.6 CAN

CAN – Controller Area Network is a robust communication technology originally developed for automotive applications but is also used in industrial machines, batteries, vehicles and embedded systems.
Typical applications include:
Vehicles | Battery Management Systems | Industrial Machines | Embedded Controllers
Typical architecture:
Controller/Sensor → CAN Bus → CAN Gateway → Application
CAN communication is based on **message identifiers**, rather than the traditional master/slave polling model used by Modbus RTU.
Important parameters include:
CAN ID | Bit Rate | Frame Format | Data Length | Signal Definition
Receiving a CAN frame is not sufficient to understand its meaning. The integrator also needs the manufacturer's message definition or database describing how individual bytes/bits map to engineering parameters.

3.7 PROFIBUS

PROFIBUS is a fieldbus technology commonly found in industrial automation installations.

It is frequently used between:
PLC → Remote I/O → Drives → Industrial Instruments
A commonly encountered variant is **PROFIBUS DP**, used for communication between automation controllers and distributed field devices.
Typical architecture:
PLC → PROFIBUS → Remote I/O / Drive / Field Device
Important considerations include:
Device Address | Baud Rate | Bus Termination | Network Configuration | Device Description
Many existing factories continue to operate PROFIBUS equipment, so System Integrators may need gateways to connect legacy PROFIBUS networks to newer Ethernet or IoT systems.

3.8 PROFINET

PROFINET is an Industrial Ethernet technology widely used for industrial automation.
Typical architecture:
PLC → Industrial Ethernet / PROFINET → Remote I/O / Drive
Compared with PROFIBUS, PROFINET uses Ethernet infrastructure and is designed for modern industrial automation networks.
A System Integrator may need to understand:
Device Name | IP Address | Network Topology | Controller/Device Relationship | Device Configuration
A common modernization path may therefore look like:
PROFINET Device/PLC → Industrial Gateway/Edge System → MQTT/OPC UA → IoT Platform
PROFINET should not be confused with ordinary Ethernet. **Ethernet provides the underlying networking technology; PROFINET defines industrial communication on top of Ethernet.**

3.9 Industrial Ethernet

Ethernet has become a fundamental communication technology in modern industrial systems.
It provides the network infrastructure over which several protocols can operate, including:
Modbus TCP | BACnet/IP | PROFINET | OPC UA | MQTT | HTTP
Typical architecture:
PLC/Gateway → Ethernet Switch → Industrial Network → Server/Platform
For System Integrators, Ethernet integration normally requires understanding:
IP Address | Subnet Mask | Default Gateway | DNS | DHCP/Static IP | TCP/UDP Ports | VLAN | Firewall
This creates another important distinction:
Ethernet does not define what the industrial data means. It provides the communication network over which higher-level protocols exchange that data.

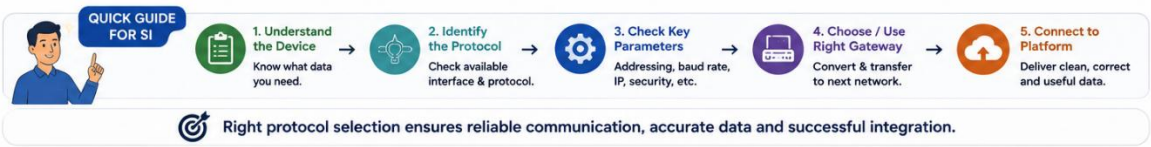
3.10 OPC UA

OPC UA – Open Platform Communications Unified Architecture is commonly used for secure and structured data exchange between industrial systems and higher-level applications.
Typical architecture:
PLC/SCADA → OPC UA Server → OPC UA Client/Edge Gateway → IoT Platform
OPC UA represents information using a structured **address space containing nodes and objects**.
An IoT gateway or application can act as an OPC UA client, browse the available data, subscribe to required variables and forward selected information to another system.
Important integration parameters include:
Endpoint URL | Security Mode | Security Policy | Certificate | Authentication | Node ID | Subscription
For example:
PLC → OPC UA → Edge Gateway → MQTT → Cloud
This architecture is common when OT information must be securely integrated with IT or cloud systems.

3.11 Protocol Selection — System Integrator View

There is no single protocol suitable for every application. In many projects, the protocol is already determined by the equipment.

PROTOCOL	TYPICAL USE / AREA	MEDIUM / INTERFACE	KEY FOCUS FOR SI (System Integrator)	EXAMPLE APPLICATION
Modbus RTU	Meters, Sensors, Field Devices, PLCs	RS-485	Registers, Baud Rate, Slave ID, Parity	Energy Meter to Gateway
Modbus TCP	PLCs, Controllers, Network Devices	Ethernet / IP	IP Address, Port (502), Unit ID, Registers	PLC to SCADA / IoT System
BACnet MS/TP	Building Automation (BMS)	RS-485	MAC Address, Device ID, Baud Rate, Objects	BMS Controller Integration
BACnet/IP	Building Automation (BMS) over IP	Ethernet / IP	IP Address, Network, BACnet Objects	BMS to Central Platform
M-Bus	Utility Meters (Water, Heat, Gas, Energy)	M-Bus (Wired)	Primary/Secondary Address, Baud Rate, Data Records, Master	Meter to M-Bus Master
Wireless M-Bus	Utility Meters in hard to wire locations	RF (868 MHz typical)	Frequency, Mode, Encryption Key, Meter ID, Gateway Compatibility	Meter to Wireless Gateway
DLMS / COSEM	Smart Electricity Meters (AMI)	Multiple (PLC / RF / Cellular, etc.)	OBIS Codes, Security, Authentication, Profiles	Smart Meter to Head-End System
CAN	Machines, Vehicles, BMS, Embedded Systems	CAN Bus	CAN ID, Bit Rate, Message Mapping	BMS / Vehicle Data Integration
PROFIBUS	Industrial Automation (Legacy Systems)	Fieldbus (RS-485)	Device Address, Baud Rate, Network Config.	PLC to Field Devices
PROFINET	Industrial Automation (Modern Systems)	Industrial Ethernet	IP Address / Device Name, Network Topology, Config.	PLC to PROFINET Devices
Ethernet (IP)	Network Infrastructure for Industrial Systems	Ethernet / IP	IP, Subnet, Gateway, Ports, VLAN, Security	Network Connectivity & Data Transport
OPC UA	OT to IT Integration, SCADA, MES, Cloud	Ethernet / IP	Endpoint, Security, Certificates, Nodes, Subscriptions	SCADA / MES to Cloud



Right protocol selection ensures reliable communication, accurate data and successful integration.

3.12 The System Integrator's Method

For a System Integrator, successful protocol integration means more than establishing communication. The real objective is to preserve the correct meaning of data while moving it from the field device to the system that needs it.

A voltage value that successfully reaches the cloud with the wrong register, byte order, scaling or engineering unit is connected data, but incorrectly integrated data.

That distinction is one of the most important principles of industrial IoT integration.

A Step-by-Step Approach to Integrate Any Industrial Device

When an unfamiliar industrial device arrives,
do not start by searching for a gateway.
Start with the device documentation and identify

Physical Interface → Protocol → Communication Settings → Addressing → Data Map → Data Type → Security → Destination

		Example
1	**PHYSICAL INTERFACE** How is it physically connected?	**RS-485** (2-wire differential)
2	**PROTOCOL** How does it communicate?	**Modbus RTU**
3	**COMMUNICATION SETTINGS** What are its communication settings?	**Slave ID 10 / 9600 / 8N1** (Baud Rate / Data Bits / Parity / Stop Bits)
4	**ADDRESSING** Where is the required parameter?	**Register 40001** (Holding Register)
5	**DATA MAP** What data is available and where?	• Voltage – 40001 • Current – 40002 • Energy – 40003/40004 • Power – 40005
6	**DATA TYPE** How should the raw value be interpreted?	**Float32 / Scaling** (IEEE 754) Scale = 0.01
7	**SECURITY** Is any security required for communication?	**None / Basic** (or Password / Encryption as per device)
8	**DESTINATION** How will the information reach the application?	**MQTT / OPC UA / API** (To IoT Platform / Cloud / SCADA)

EXAMPLE FLOW

Energy Meter

1. **RS-485** — How is it physically connected?
2. **Modbus RTU** — How does it communicate?
3. **Slave ID 10 / 9600 / 8N1** — What are its communication settings?
4. **Register 40001** — Where is the required parameter?
5. **Float32 / Scaling** — How should the raw value be interpreted?
6. **Gateway** — How will the protocol be converted?
7. **MQTT / OPC UA / API** — How will the information reach the application?

WHAT A GOOD SI DOES

- Reads the documentation first
- Understands the data meaning, not just connectivity
- Validates data accuracy (value, unit, scale)
- Ensures reliable and secure data flow end-to-end

KEY PRINCIPLE

Understand the Device › Identify the Details › Map & Interpret the Data › Convert & Transmit › Deliver to Application

Follow this method and you can integrate any industrial device with confidence!

Chapter 4

Wireless Connectivity

Wireless connectivity allows IoT devices to exchange data without a physical communication cable. However, **wireless is not a single technology**. Different technologies are designed for different requirements such as data rate, range, power consumption, number of devices, mobility, cost, and network topology.

For a System Integrator, the objective is not simply to choose the technology with the longest range or highest speed. The correct question is:

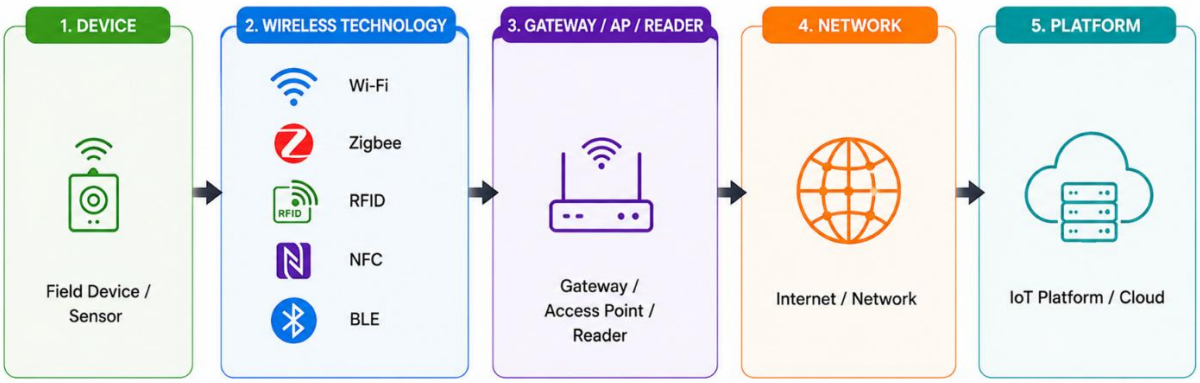
What data needs to be transferred, over what distance, how often, with what power source, and to which system?

4.1 Understanding Wireless Connectivity

Before selecting a wireless technology, a System Integrator should understand a few basic parameters.

Frequency Band: The radio frequency used by the technology, such as 2.4 GHz.
Range: The practical communication distance between devices. Actual range depends heavily on walls, interference, antenna design, transmit power, installation height, and the surrounding environment.
Data Rate: The amount of data that can be transferred within a given time.
Power Consumption: An important consideration for battery-operated sensors and meters.
Topology: How devices communicate with each other, such as point-to-point, star, or mesh.
Interference: Multiple technologies may operate in the same frequency band. Wi-Fi, BLE and Zigbee can all use 2.4 GHz, so RF planning can become important.

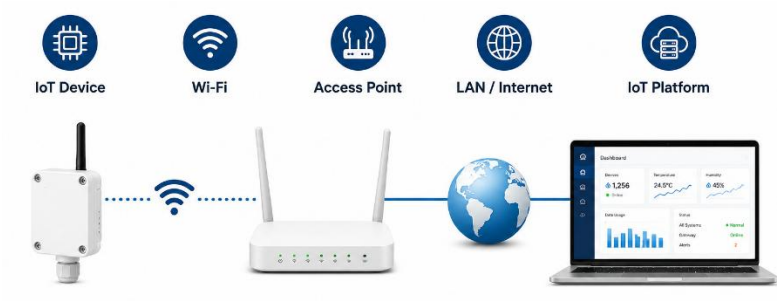
From an integration perspective:



***The exact architecture changes depending on the technology.*

4.2 Wi-Fi

Wi-Fi is a wireless networking technology based on the IEEE 802.11 family of standards. It is widely used in homes, offices, commercial buildings and industrial environments.
Unlike many low-power IoT technologies, Wi-Fi provides relatively high data throughput and direct integration with **IP networks**.
Typical IoT architecture:



Why Wi-Fi is useful for IoT

One major advantage of Wi-Fi is that existing IT infrastructure can often be used.
A Wi-Fi-enabled IoT device can connect to an access point and communicate with local or cloud applications using IP-based protocols such as MQTT | HTTP/HTTPS | WebSocket

What a System Integrator should check

When integrating a Wi-Fi device, check:
SSID | Password/Security | Frequency Band | IP Configuration | Signal Strength | Firewall | Internet Access
IP configuration is particularly important.
A device may receive an address automatically through **DHCP**, or a **static IP** may be configured depending on the project.
For industrial installations, the integrator should also consider signal coverage, network segmentation, firewall policies and whether the customer's IT team permits IoT devices on the corporate network.

Successful Wi-Fi association does not guarantee application connectivity.

A device may be connected to Wi-Fi but still fail to reach the IoT platform due to DNS failure, incorrect routing, firewall or port restrictions, certificate issues, or unavailable Internet connectivity. Therefore, a System Integrator should verify the complete communication path from the device to the platform, not just the Wi-Fi connection.

4.3 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) is designed for short-range wireless communication with low power consumption.

It is commonly used in:
Sensors | Beacons | Wearables | Asset Tracking | Configuration Interfaces | Battery-Powered Devices
A typical IoT architecture is:

Unlike Wi-Fi, many BLE devices do not communicate directly with a cloud platform. A **smartphone, gateway, or edge device** commonly acts as the bridge between BLE and an IP network.

What an Integrator should check

Important BLE parameters include:
Device Address/Identity | Service UUID | Characteristic UUID | Pairing | Advertising | Security | Gateway Compatibility
BLE is particularly useful when the device must operate for a long time on a battery and only needs to exchange relatively small amounts of data.

4.4 Zigbee

Zigbee is a low-power wireless technology commonly used for sensors, automation and building applications. It is based on IEEE 802.15.4 at the lower layers.
Common applications include - **Lighting | Occupancy Sensors | Temperature Sensors | Smart Buildings | Home Automation**
One of Zigbee's important characteristics is its support for **mesh networking**.

A typical Zigbee network contains three types of devices:

Coordinator
The coordinator establishes and manages the Zigbee network. A Zigbee network normally has one coordinator.
Router
Routers participate in the network and can forward traffic from other Zigbee devices, helping extend network coverage.
End Device
An end device is usually a sensor, switch, or other endpoint. Battery-operated end devices can be designed to spend significant time in low-power states.

Why Mesh Matters

Suppose a sensor cannot communicate reliably with the coordinator because of distance or building structure.
A suitable Zigbee router may provide an intermediate path:
Sensor → Router → Router → Coordinator
This can improve network coverage, although mesh design still requires appropriate planning and sufficient router placement.

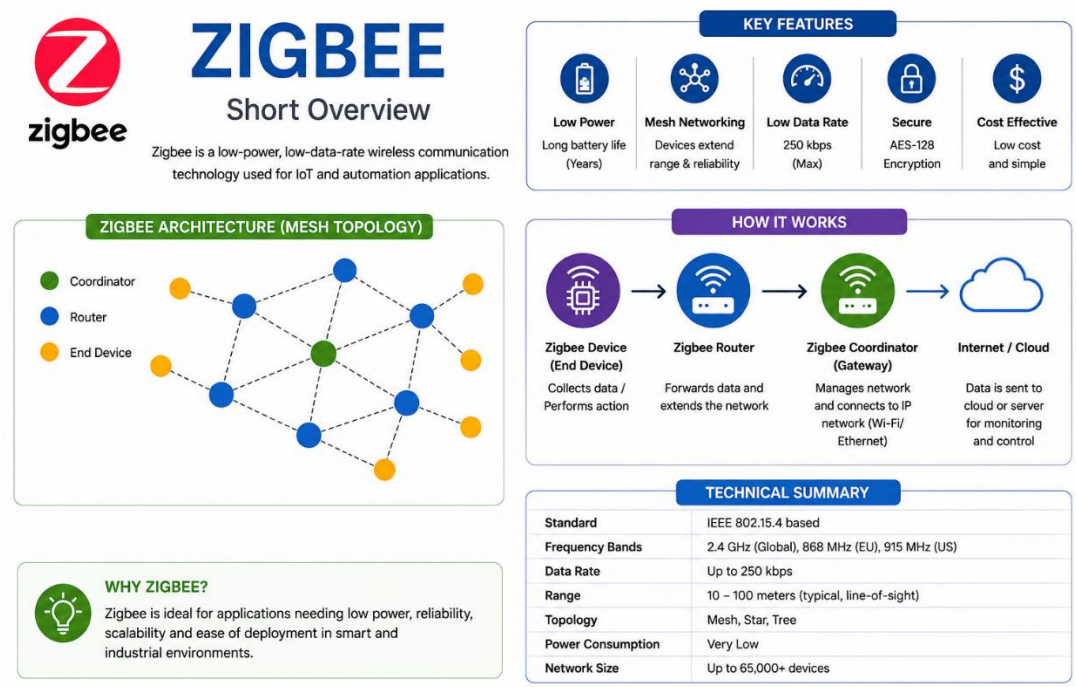
What a System Integrator should check

Frequency | Channel | Coordinator | Device Compatibility | Network Joining | Security | Coverage | Gateway Support

A System Integrator should also remember that **using the same radio technology does not automatically guarantee application-level interoperability**. Device profiles, supported clusters, vendor implementation, and gateway compatibility must also be checked.

*It means **two Zigbee devices can use the same Zigbee radio standard but still may not work correctly together**.*
For example, suppose you have:
Zigbee Temperature Sensor → Zigbee Gateway
*Both support Zigbee and can operate at **2.4 GHz**, but this alone does not guarantee that the gateway can understand the sensor data.*
The System Integrator should also check:

- ***Device Profile** – Defines the application type and expected device behavior.*
- ***Clusters** – Define specific functions/data, such as temperature measurement, occupancy, switching, etc.*
- ***Vendor Implementation** – Some manufacturers use proprietary features or manufacturer-specific data.*
- ***Gateway Compatibility** – The gateway must support the device and its required clusters/features.*



4.5 RFID

RFID — Radio Frequency Identification is primarily used to identify and track objects using radio communication.
A basic RFID system consists of: **RFID Tag → RFID Reader → Application/Database**
Typical applications include **Asset Tracking | Inventory | Warehousing | Access Control | Manufacturing | Logistics**
Unlike Wi-Fi or Zigbee, RFID is generally not used as a conventional device-to-cloud communication network. Its main purpose is to allow a reader to identify or obtain information associated with a tagged object.

RFID Tags

Passive Tags: Do not normally contain their own battery for communication. They obtain energy from the reader's RF field.
Active Tags: Have their own power source and can support different operating characteristics and longer communication distances depending on the system.

Different RFID systems operate at different frequency ranges, commonly including:

LF | HF | UHF

The appropriate technology depends on application requirements such as read distance, environment, tag type and material.

Typical IoT Integration

An RFID reader can become part of a larger IoT solution:

The IoT integration therefore often happens at the **reader level**, rather than directly at the tag.

What a System Integrator should check

Important parameters include:
RFID Frequency | Tag Type | Read Range | Reader Interface | Antenna | Tag ID/Data | Application Interface
Environmental conditions also matter. Metal, liquids, reader placement, antenna orientation and RF interference can significantly affect RFID performance.

4.6 NFC

NFC — Near Field Communication is a very short-range wireless technology operating at **13.56 MHz**.
It is commonly used for:
Contactless Cards | Access Control | Payments | Device Pairing | Product Identification | Device Configuration

A simple architecture may be:
NFC Tag → Smartphone/Reader → Application
The intentionally short operating distance makes NFC useful when the user is expected to bring a phone, card or reader very close to the device.

NFC in IoT: NFC can be particularly useful for **device commissioning and maintenance**.
For example: **IoT Device → NFC → Technician's Smartphone**
The technician may use NFC to read device information or configure supported parameters without requiring the device to expose a conventional wired interface.

A product may therefore use two different communication technologies:

NFC → Local Configuration

and

LoRaWAN/NB-IoT/Wi-Fi → Operational Data Communication

This is an important System Integration concept: **one IoT device may contain multiple communication technologies, each serving a different purpose.**

What a System Integrator should check
Important NFC considerations include:
Tag Type | Reader Compatibility | Data Format | Read/Write Support | Security | Application Support

4.7 Wi-Fi vs BLE vs Zigbee vs RFID vs NFC

These technologies should not be compared only by range because they serve different purposes.

The table is a general comparison. Actual range and power consumption depend on the specific device, radio configuration, antenna, environment and implementation.

Parameter	Wi-Fi	BLE	Zigbee	RFID	NFC
Primary Purpose	High-speed wireless networking	Low-power short-range communication	Low-power mesh networking	Identification & tracking	Very short-range communication
Frequency	2.4 / 5 GHz	2.4 GHz	2.4 GHz, 868 MHz, 915 MHz*	LF, HF, UHF	13.56 MHz

Typical Range	~30–100+ m	~1–50 m	~10–100 m per hop	Depends on type; UHF can reach several meters	Up to ~4 cm
Data Rate	High; up to several Gbps depending on standard	Up to ~2 Mbps	Up to 250 kbps	Depends on RFID type	106–424 kbps
Power Consumption	High	Very Low	Low	Very Low for passive tags	Very Low
Network Topology	Mainly Star	Star	Mesh / Star / Tree	Reader–Tag	Point-to-Point
Typical Devices	Routers, cameras, laptops, gateways	Sensors, wearables, medical devices	Sensors, switches, smart lights, gateways	Tags, readers, tracking devices	Smartphones, cards, readers
Typical Applications	Internet access, cameras, data transfer	Sensors, wearables, beacons	Smart buildings, automation, sensors	Inventory, asset tracking, access control	Payments, access control, pairing
Main Advantage	High speed and IP connectivity	Very low power and easy smartphone integration	Mesh networking with low power	Simple identification and tracking	Very short-range and convenient
Main Limitation	Higher power consumption	Shorter range and lower throughput	Lower data rate	Requires compatible tags/readers	Very short communication range

4.8 How a System Integrator Should Select Wireless Technology

Instead of asking **"Which wireless technology is best?"**, start with the application.

Consider these questions:

1. What is being connected?
A sensor, controller, camera, asset tag, mobile phone, or industrial machine?

2. How much data is generated?
A few bytes of temperature data and a video stream have completely different requirements.

3. How far must communication travel?
Across a room, building, factory, campus, or city?

4. How is the device powered?
Mains-powered devices can tolerate higher power consumption, while battery devices may require aggressive power optimization.

5. How often is data transmitted?
Every second, every minute, once per hour, or only when an event occurs?

6. Is infrastructure available?
Existing Wi-Fi, dedicated gateway, smartphone, RFID reader, or Zigbee coordinator?

7. Where must the data finally go?
Local application, BMS, SCADA, database, cloud, or IoT platform?

This gives a practical selection path:

Application → Data → Range → Power → Environment → Infrastructure → Security → Platform

4.9 System Integrator's View

A wireless technology should never be selected from the datasheet range alone.

For example, a specification may indicate a certain communication distance under ideal conditions, while the actual installation contains **concrete walls, metal equipment, electrical panels, machinery, interference and poor antenna locations**.

Therefore, an integrator should validate the complete communication path:

Device → Radio Link → Gateway/Access Point/Reader → IP Network → Platform → Application

If data does not reach the application, troubleshooting should follow the same path in sequence rather than immediately blaming the wireless signal.

For example:

Is the device transmitting? → Is RF communication working? → Is the gateway receiving it? → Does the gateway have network connectivity? → Is the required port/service accessible? → Is the platform receiving the data? → Is the application interpreting it correctly?

The role of a System Integrator is not simply to make a wireless connection. It is to ensure that the right technology carries the right data reliably from the physical device to the application that needs it.

This end-to-end approach becomes increasingly important as IoT solutions combine multiple wireless and wired technologies within the same architecture.

Chapter 5

LoRaWAN

LoRaWAN is one of the most widely used technologies for connecting battery-powered IoT devices over long distances. It is particularly useful when devices generate small amounts of data and may need to operate for several years without frequent battery replacement.

For a System Integrator, understanding LoRaWAN is not limited to knowing how a LoRaWAN gateway works. The integrator needs to understand the complete path:

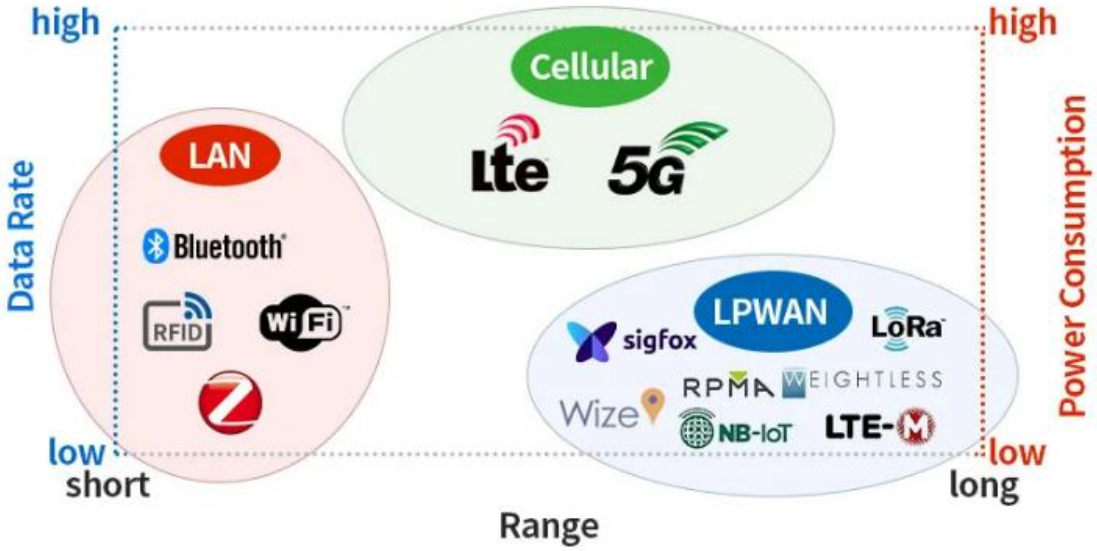
LoRaWAN is an LPWAN technology designed for low-power, wide-area IoT connectivity. The LoRa Alliance describes it as an end-to-end architecture for connecting battery-operated devices to regional, national, and global networks.

5.1 Introduction to LPWAN

LPWAN stands for **Low Power Wide Area Network**.

It describes a category of wireless technologies designed for applications that require:

- Long communication range
- Low power consumption
- Small data volumes
- Large numbers of devices
- Low operating cost
- Long battery life



The basic idea is simple:

Send a small amount of information over a long distance while consuming very little energy.

5.2 LoRa vs LoRaWAN

One of the most important concepts for a System Integrator is understanding the difference between **LoRa** and **LoRaWAN**.

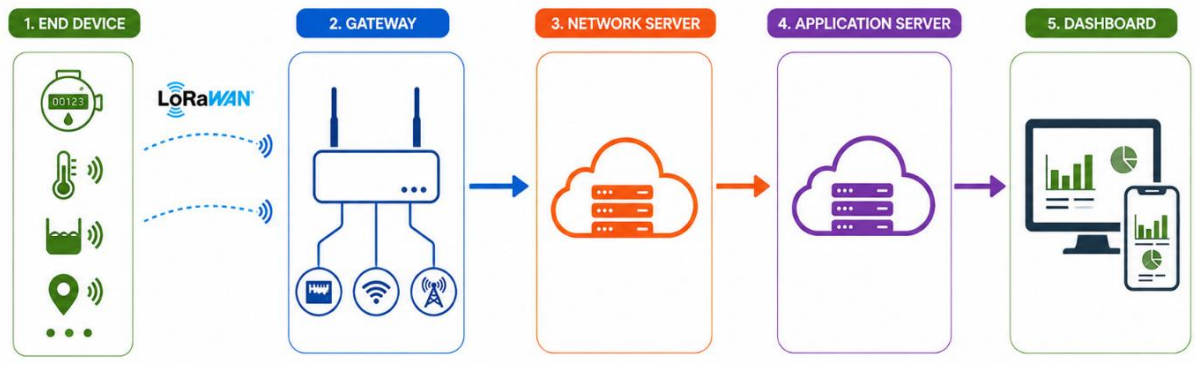
LoRa: **LoRa** is a radio modulation technology used for long-range, low-power communication.
It defines how information is transmitted over the radio link.

LoRaWAN: **LoRaWAN** is the networking protocol and architecture built around LoRa radio technology.
It defines how end devices communicate with the network, how devices are authenticated, how messages are managed, how downlinks are handled, and how devices interact with the network infrastructure.

A simple way to remember it:
LoRa = Radio Technology
LoRaWAN = Network Protocol

The LoRa Alliance (https://lora-alliance.org/) maintains the LoRaWAN specification, while LoRa is the physical-layer modulation used by many LoRaWAN devices.

5.3 LoRaWAN Architecture



1. End Device

The end device is the field device that generates or consumes data.
Example: **Water Meter | Energy Meter | Gas Meter | Temperature Sensor | Pressure Sensor**

2. LoRaWAN Gateway

Gateway: A **gateway** is a device that connects field devices or sensors to a network or server, often converting communication protocols when required.

A **LoRaWAN Gateway** receives LoRaWAN radio messages from end devices and forwards them to the **LoRaWAN Network Server (LNS)** over an IP network.

A gateway typically supports backhaul connectivity such as **Ethernet, Wi-Fi, or Cellular** and can receive messages from multiple devices within its coverage area.

Key points for a System Integrator:

- Frequency plan
- Antenna and installation
- Channel configuration
- Gateway connectivity/backhaul
- Packet Forwarder
- LNS configuration
- Gateway coverage and RF performance

Types of LoRaWAN Gateway

- Indoor (picocell) – For deep indoor locations
- Outdoor (macrocell) – For Large coverage at outdoor

Configuration of Gateway:

- **Install Packet Forwarder** – Install and configure the appropriate packet-forwarder software on the gateway to receive LoRaWAN packets and forward them to the LNS.

- **Configure Frequency Plan** – Select the correct regional frequency plan and configure the required channels.
- **Configure LNS Server** – Enter the **LNS server address (IP/Hostname)** and the required **communication ports** (Uplink and Downlink port) in the packet-forwarder configuration.

3. Network Server

The **LoRaWAN Network Server (LNS)** manages the LoRaWAN network.

A **Network Server** manages the communication between gateways and end devices, handling device authentication, message routing, deduplication, and network-level functions.

Depending on the implementation, it may perform functions such as:

- Device authentication
- MIC validation
- Message deduplication
- Frame-counter handling
- Uplink routing
- Downlink scheduling
- ADR management
- MAC command handling
- Gateway management
- Network-level security functions

When several gateways receive the same uplink, the network server can process the duplicate receptions and select the appropriate gateway/path for further network handling.

4. Application Server

The Application Server is responsible for application-level data.

An **Application Server** processes and interprets device data, applies application logic, stores information, and provides data to dashboards or other applications.

For example:

Payload → Decode → Temperature = 24.6°C

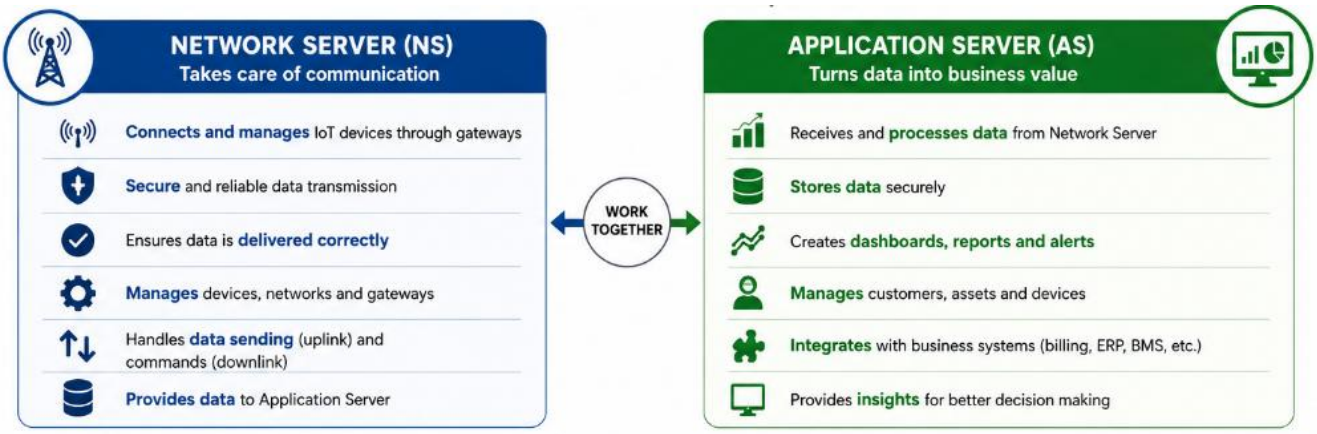
The application server may:

- Decode payloads
- Apply business logic
- Store data
- Generate alarms
- Provide APIs
- Send data to dashboards

This separation is important:

Network Server = Manages the LoRaWAN network

Application Server = Manages application data



5. Dashboard

A **Dashboard** presents processed IoT data in a visual form, allowing users to monitor device status, measurements, trends, alarms, and other key information.

The dashboard can be part of the Application Server, depending on the system architecture

Two common architectures

1. Combined:
LNS → Application Server + Dashboard

The application server handles data processing and also hosts the web dashboard.

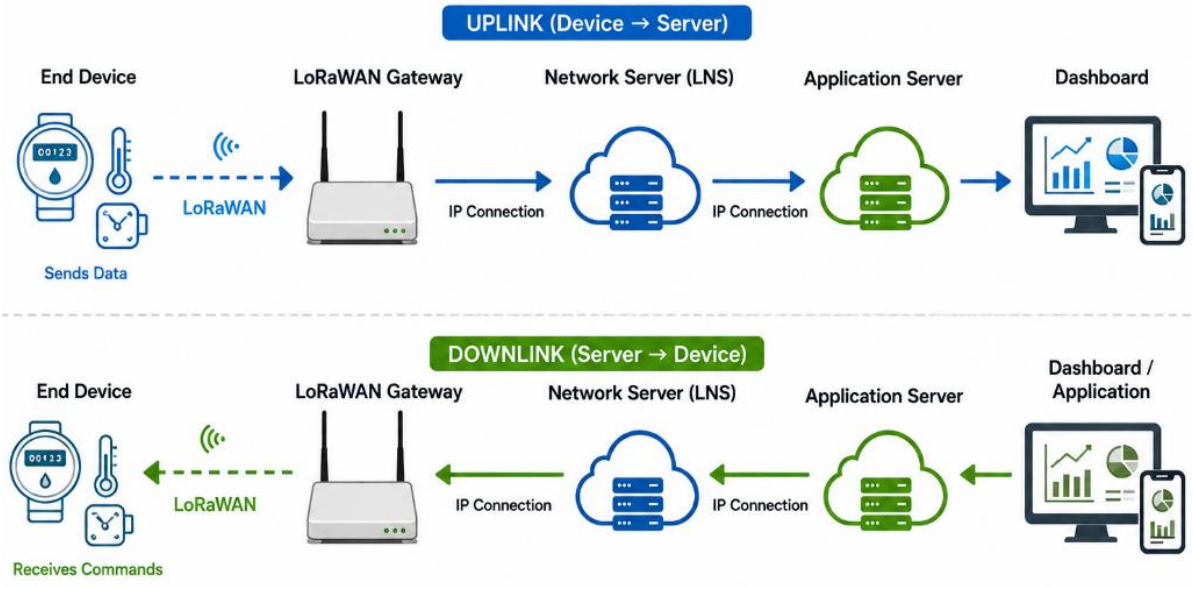
2. Separate:
LNS → Application Server → Dashboard

Simplified Architecture:

End Device → LoRaWAN Gateway → Network Server (LNS) → Application Server / Dashboard

5.4 Uplink and Downlink Communication

Uplink: Data travels from the device toward the application.
Downlink: Data travels from the application/network toward the device.
Downlink should be used carefully because it can have a significant impact on battery-operated devices and network capacity.



5.5 LoRaWAN Device Activation

Before a LoRaWAN device can communicate correctly with the network, it needs the appropriate activation and security configuration.

Two commonly encountered activation methods are:

OTAA – Over-The-Air Activation
ABP – Activation By Personalization

- **Over-The-Air-Activation (OTAA)** - the most secure and recommended activation method for end devices. Devices perform a join procedure with the network, during which a dynamic device address is assigned and security keys are negotiated with the device.

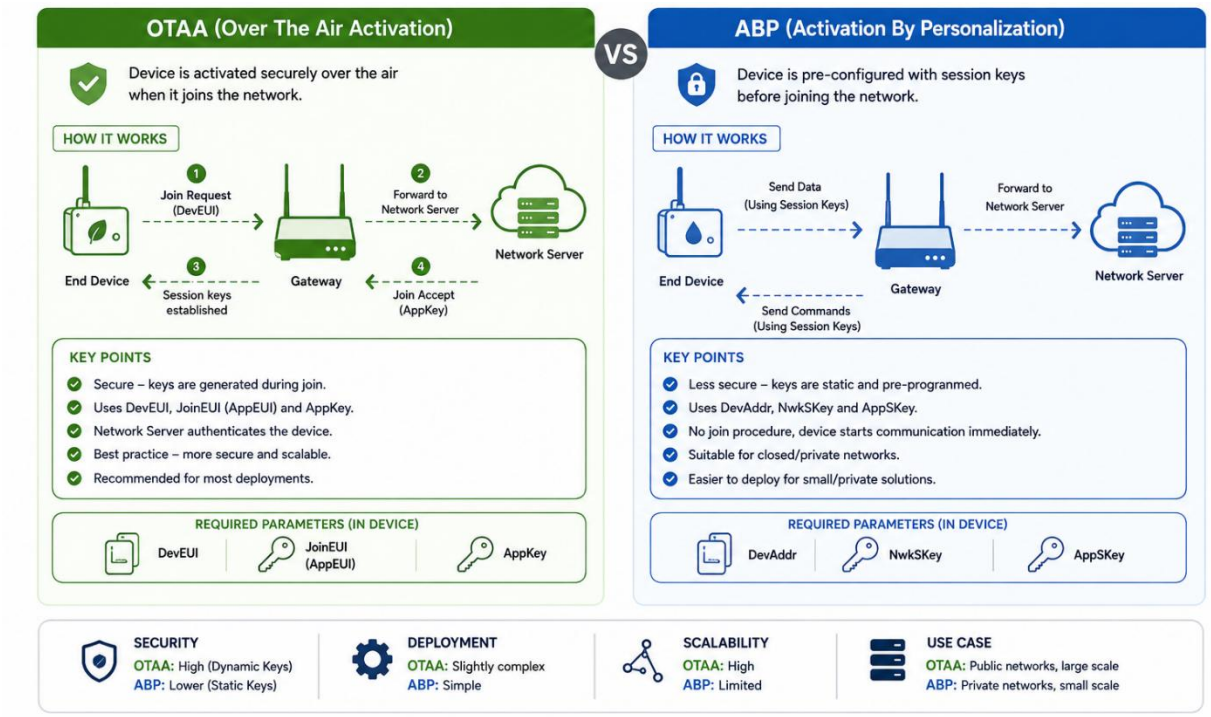
- **Activation By Personalization (ABP)** - requires hardcoding the device address as well as the security keys in the device. ABP is **less secure** than OTAA and also has the downside that devices can not switch network providers without manually changing keys in the device

OTAA

- Keys Required
  - DevEUI | JoinEUI | AppKey

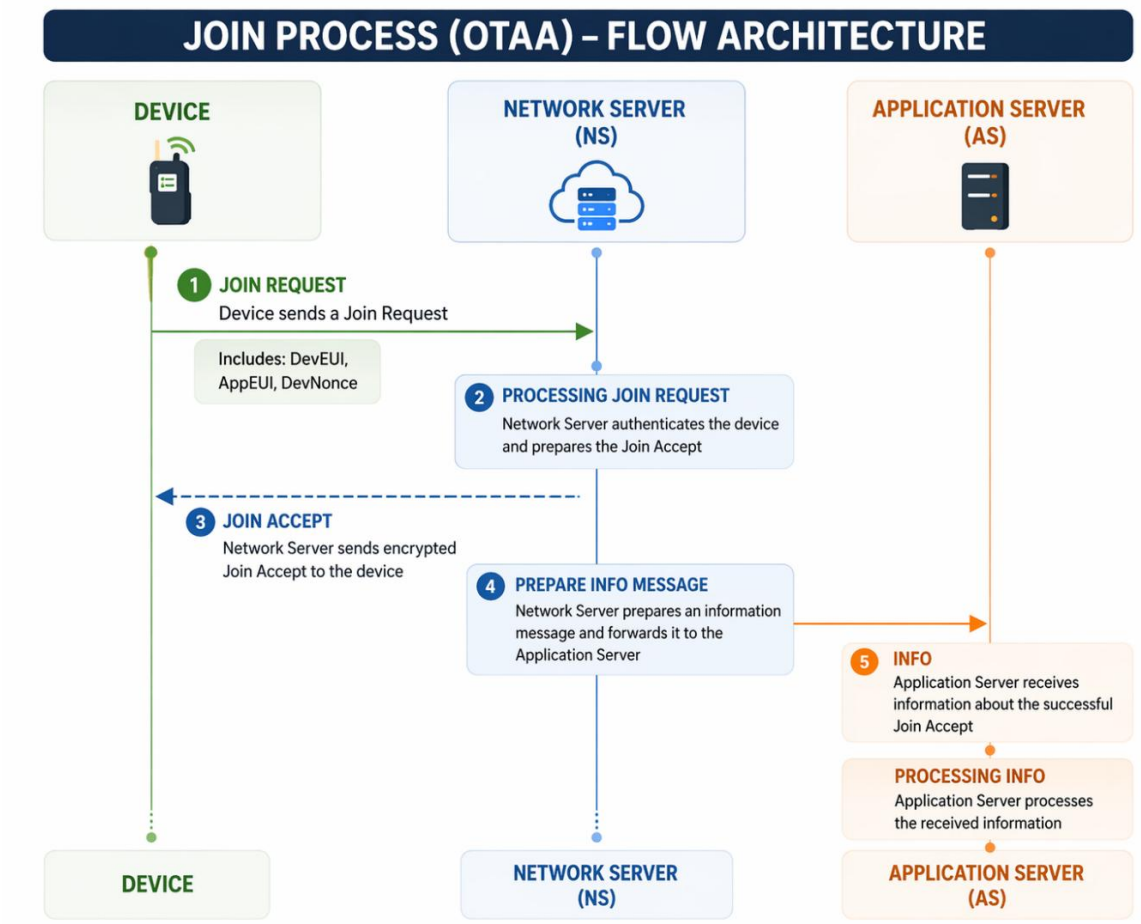
ABP

- Keys Required
  - DevEUI | DevAddr | AppSKey | NwkSKey



Join Process:

- Device sends Join Request
- Network Server authenticates
- Sends Join Accept
- Session keys generated
- Device starts communication



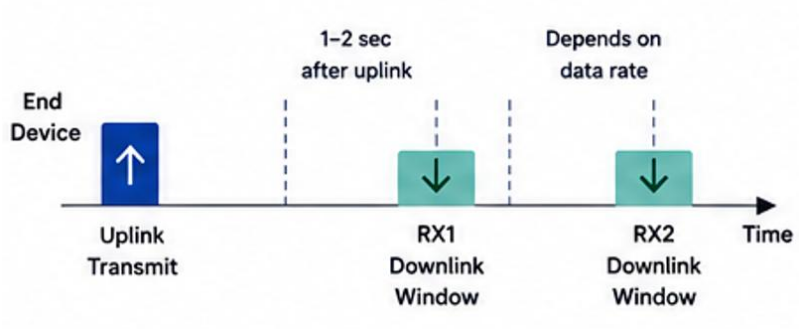
5.6 LoRaWAN Device Classes

LoRaWAN defines three main device classes:

Class A

Class A is the basic and lowest-power operating mode.

The device transmits an uplink when required by its application and then opens two receive windows for possible downlink communication.



The device can remain asleep until it needs to transmit again.

Best suited for

Battery-powered Sensors | Water Meters | Gas Meters | Environmental Sensors

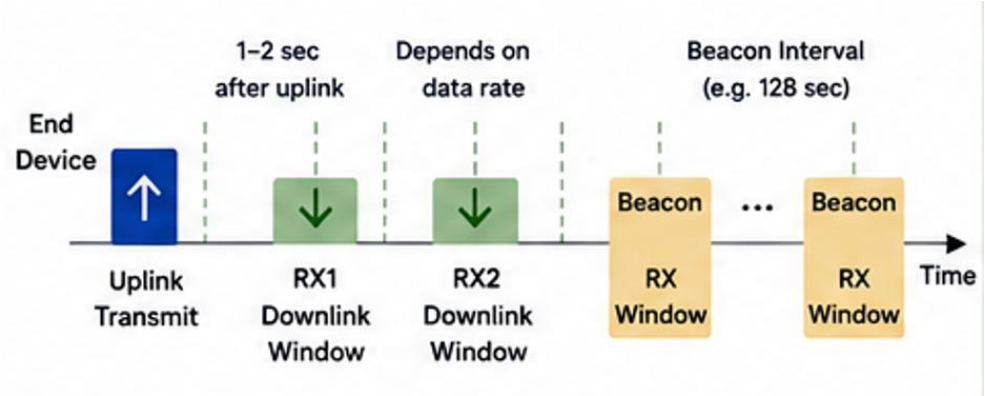
Class A provides excellent battery efficiency, but it is not suitable when the application needs an immediate downlink at any time.

The LoRaWAN specification requires all end devices to support Class A.

Class B

Class B adds scheduled receive opportunities called **ping slots**.

The device synchronizes its timing using network beacons and periodically opens additional receive windows.



Class B provides more predictable downlink availability than Class A but requires additional power and timing management.

It can be considered when the application requires scheduled downlink communication.

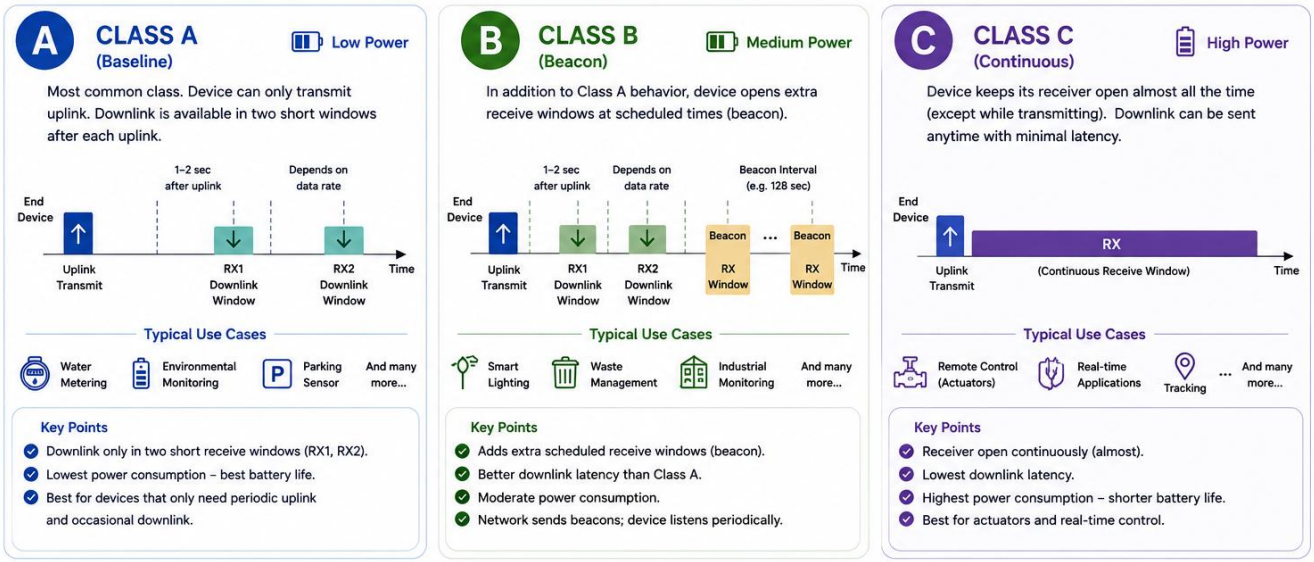
Class C

Class C keeps the receive path open for most of the time when the device is not transmitting.

This provides low downlink latency but consumes considerably more power than Class A.

Class C is therefore more appropriate for:

Mains-Powered Devices | Actuators | Controllers | Applications Requiring Frequent Downlinks



Feature	Class A	Class B	Class C
Downlink Availability	Only in RX1 & RX2 after uplink	In RX1, RX2 + scheduled beacon windows	Anytime (almost instant)
Downlink Latency	High	Medium	Low (very low)
Power Consumption	Lowest	Medium	Highest
Battery Life	Longest	Medium	Shortest
Use Case	Sensors, periodic data	Periodic data + better downlink	Real-time control, actuators

Note:
Choose the device class based on application requirements for downlink latency and power consumption.

5.7 Adaptive Data Rate – ADR

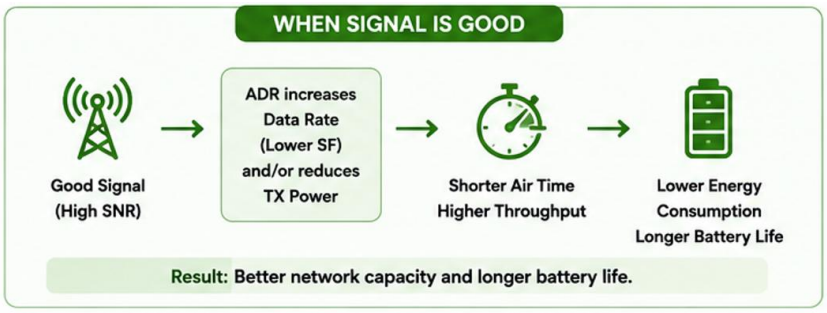
ADR (Adaptive Data Rate) is a LoRaWAN mechanism used to optimize communication parameters based on network conditions. The objective is generally to allow suitable devices to use an appropriate data rate and transmit power. A device with a strong and stable link may be able to communicate using a higher data rate and shorter time-on-air. A device with a weaker link may require a more robust configuration.

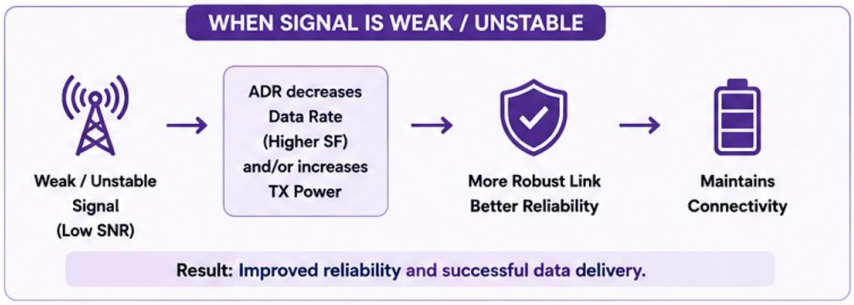
ADR can therefore help improve:

- Battery life
- Network capacity
- Time-on-air
- Overall network efficiency

It is most useful for devices that are **stationary or have relatively stable radio conditions**. Mobile devices may require different considerations.

The LoRaWAN ecosystem identifies ADR as an important mechanism for optimizing network operation.





5.8 Data Rate and Spreading Factor

Data Rate is the speed at which data is transmitted over the LoRaWAN radio link, usually expressed in bits per second (bps).

- Higher DR → faster transmission, shorter range – DR5
- Lower DR → slower transmission, longer range – DR0

Spreading Factor (SF)

Spreading Factor determines how the LoRa signal is spread over time. LoRaWAN commonly uses SF7 to SF12.

Lower SF → Higher Data Rate → Shorter airtime
Higher SF → Lower Data Rate → Longer airtime → Greater robustness/range

DR	SF	ToA (ms)	RSSI(dBm)
5	7	34	-124
4	8	68	-127
3	9	115	-130
2	10	231	-133
1	11	462	-135
0	12	925	-137

ToA – Time on Air

5.9 RSSI and SNR

Two important radio parameters frequently seen during LoRaWAN troubleshooting are:

RSSI – Received Signal Strength Indicator

SNR – Signal-to-Noise Ratio

RSSI

RSSI indicates the received signal strength, normally expressed in **dBm**.
The value is usually negative.
For example:
-50 dBm → Stronger signal
-80 dBm → Weaker
-110 dBm → Very weak
The exact acceptable value depends on the device, spreading factor, bandwidth, receiver sensitivity, and environment.
Therefore, RSSI should not be evaluated in isolation.

SNR

SNR represents the relationship between the received signal and the noise level.
A higher SNR generally indicates a cleaner signal.

LoRa modulation can operate with relatively low or negative SNR under suitable configurations, so:
A negative SNR does not automatically mean that the communication is failing.

For troubleshooting, always look at:

RSSI + SNR + Data Rate/SF + Packet Success + Gateway Location

rather than relying on one number.

5.10 Frequency Plans

LoRaWAN does not use one universal frequency plan for every country.

Regional parameters define the applicable channel plans and operating requirements.

Common examples include: **EU868 | US915 | AU915 | AS923 | IN865**

The actual frequency plan must be selected according to the deployment country, regulatory requirements, device hardware, and network configuration.

For example, selecting a US915 device for an AU915 deployment is not simply a matter of changing a software setting if the hardware or regional implementation does not support the required band.

A System Integrator should verify:

- Country
- Regulatory frequency range
- Device frequency plan
- Gateway frequency plan
- Channel configuration
- Transmit power limits
- Duty-cycle or dwell-time requirements where applicable
- Network-server regional configuration
- Antenna compatibility
- Device certification/type approval requirements

The LoRa Alliance maintains Regional Parameters specifically to address the regulatory and technical requirements of different regions. These parameters continue to evolve; the latest published RP2-1.0.5 release added new data rates for certain regions, so integrators should always check the current regional specification rather than relying on an old frequency table.

5.11 LoRaWAN Troubleshooting Approach

A good System Integrator should troubleshoot LoRaWAN **layer by layer** rather than changing multiple settings at the same time.

Use this path:

1. Device

- Is the device powered?
- Is the battery healthy?
- Is the device transmitting?

2. Frequency Plan

- Is the device configured for the correct regional plan?
- Does the gateway support the same frequency plan?

3. RF Signal

- Is the device within gateway coverage?
  Check:
- **RSSI | SNR | Spreading Factor | Gateway Location**

4. Gateway

- Is the gateway online?
- Is the packet forwarder running?
- Is the gateway connected to the network server?

5. Network Server

- Is the device registered?
- Is the DevEUI correct?
- For OTAA, are the JoinEUI/AppEUI and AppKey correct?
- Are uplinks reaching the LNS?

6. Application Server

- Is the correct application configured?
- Is the payload decoder correct?
- Is the correct FPort being processed?

7. Dashboard

- Is the application forwarding the decoded information correctly?
- Are database, API, MQTT or dashboard services operating?

5.12 Common LoRaWAN Problems

Device Does Not Join

Check:

- **DevEUI → JoinEUI/AppEUI → AppKey → Frequency Plan → Gateway Coverage → Gateway Configuration → LNS Configuration**
- A single incorrect activation parameter can prevent successful joining.

Gateway Is Online but No Uplink Is Visible

Check:

- **Device Transmission → Frequency Plan → Gateway Channels → RF Coverage → Packet Forwarder → Gateway-to-LNS Connection**
- The gateway being online does not necessarily mean that it is receiving the target device's RF packets.

Uplink Is Received but Data Is Wrong

Check:

- **FPort → Payload Format → Byte Order → Data Type → Scaling → Decoder**
- This is usually an application-level integration problem rather than an RF problem.

Device Has Very Poor RSSI/SNR

Check:

- **Distance → Antenna → Antenna Cable → Gateway Height → Indoor Obstructions → Metal Structures → Interference → Device Installation**
- Do not immediately change the spreading factor without understanding the actual RF environment.

Downlink Is Not Working

Check:

- **Device Class → Receive Window → Gateway Availability → LNS Configuration → Frequency Plan → Downlink Queue → Device Power State**
- For Class A devices, remember that downlink opportunities are associated with the device's uplink activity.

5.13 The System Integrator's View

A LoRaWAN deployment should never be considered complete simply because the device appears **"Connected"**.

The complete chain must be validated:

Device → RF → Gateway → IP Backhaul → LNS → Application Server → Decoder → Database → Dashboard

Each layer answers a different question:

- **Device:** Is the device operating?
- **RF:** Is the radio communication working?
- **Gateway:** Did the gateway receive the packet?
- **Network:** Did the LNS process the packet?
- **Application:** Was the payload correctly decoded?
- **Database:** Was the information stored?
- **Dashboard:** Is the correct information visible to the user?

This is where the role of the System Integrator becomes important. A device manufacturer may be responsible for the meter or sensor, a gateway manufacturer may be responsible for the gateway, and an LNS vendor may provide the network server. The System Integrator is often responsible for making sure that **all of these components work together as one solution**.

A strong LoRaWAN integrator therefore does not troubleshoot only the radio signal. The integrator follows the complete data path and identifies exactly where the communication or data flow stops.

The most useful LoRaWAN troubleshooting question is not "Is the device connected?" but "At which layer did the data flow stop?"

That mindset makes it possible to troubleshoot large deployments systematically rather than changing device, gateway, or server settings randomly.

Chapter 6

NB-IoT

6.1 What is NB-IoT?

NB-IoT (Narrowband Internet of Things) is a 3GPP-standardized cellular IoT technology designed to connect low-power devices that transmit small amounts of data over wide areas using mobile operator networks.

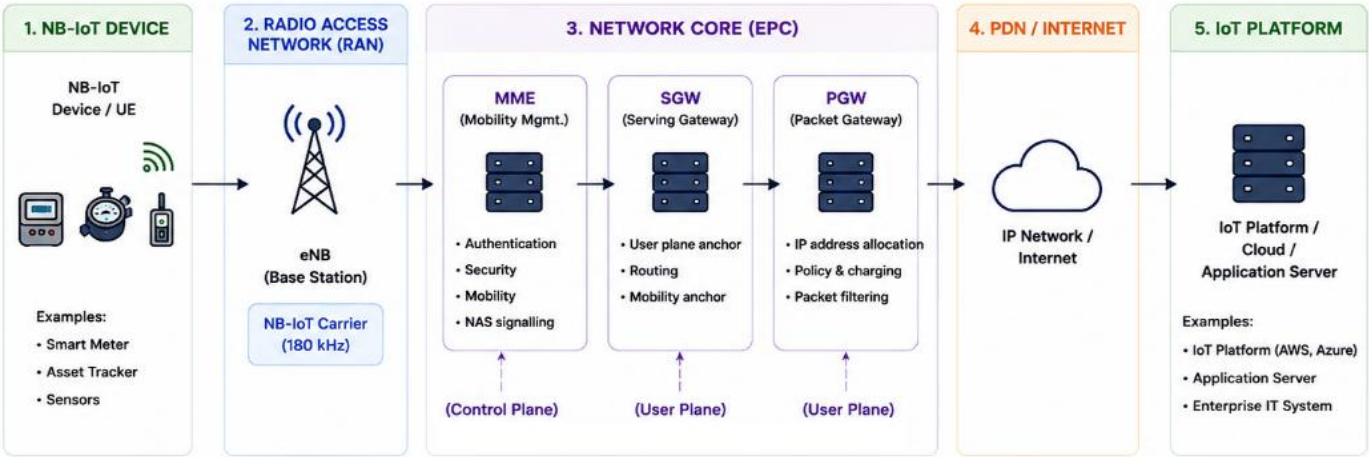
Simple architecture:

NB-IoT Device → Cellular Network → Core Network → IoT Platform

Key Characteristics

- **Low Power** – 10+ years battery life
- **Deep Coverage** – up to 20 dB+ better than LTE
- **Massive Connections** – up to 100,000 devices per cell
- **Low Throughput** – Uplink up to ~60 kbps; Downlink up to ~250 kbps
- **Secure** – SIM-based authentication and encryption

Actual performance depends on network configuration, device implementation, coverage conditions, deployment mode, power-saving configuration, and operator network.



6.2 Control Plane vs User Plane

1. Control Plane

The **Control Plane** carries **signalling and control information** required to establish, manage, and maintain communication between the NB-IoT device and the cellular network.

It is used for functions such as:

- Device registration
- Authentication
- Security
- Mobility management
- Session management
- Paging
- Network control/signalling

Main components involved:

NB-IoT Device (UE) → eNB → MME

The **MME** is the major EPC component responsible for control-plane functions in the architecture shown in your guide. The **S1-MME** interface connects the eNB and MME for control-plane signalling.

2. User Plane

The **User Plane** carries the **actual application data** generated by the IoT device.

For example:

- Water-meter reading
- Gas consumption
- Temperature
- Battery level
- Alarm/status data

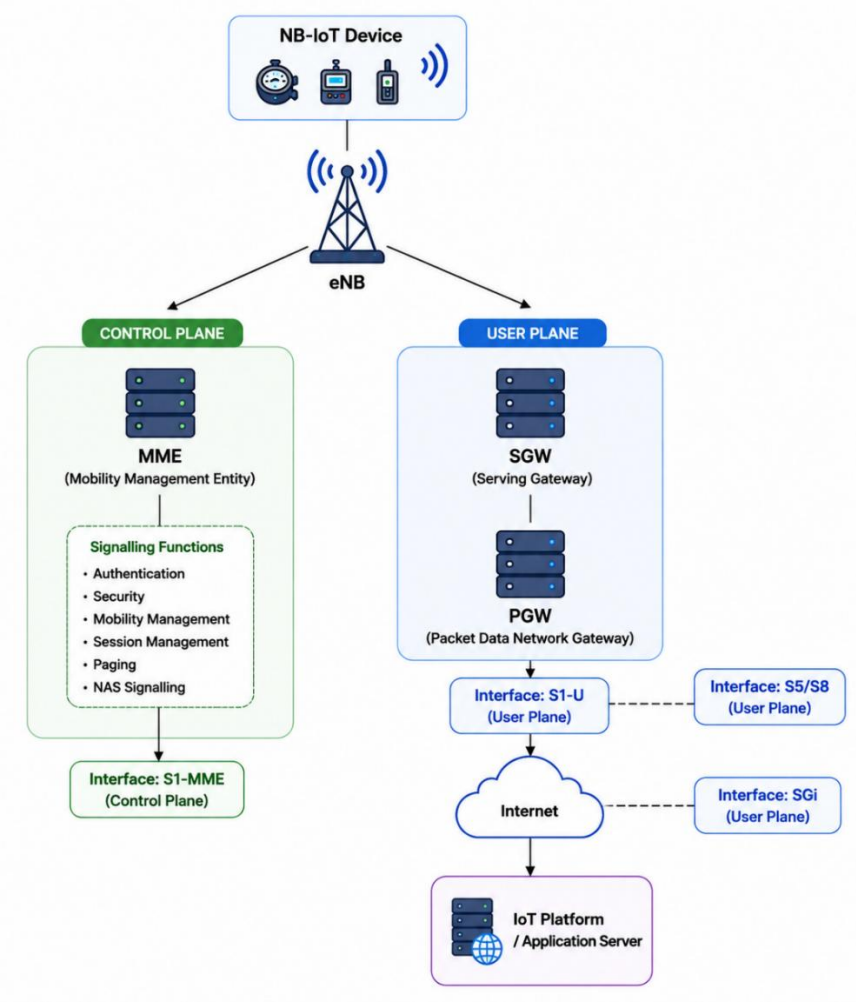
The typical path is:

NB-IoT Device → eNB → SGW → PGW → Internet/PDN → IoT Platform

The **SGW and PGW** are the main EPC components involved in the user-data path in the architecture shown in your material.

Is it Physical or Logical?

It is primarily a logical separation of network functions and traffic.
It does not mean there are two separate physical cables:

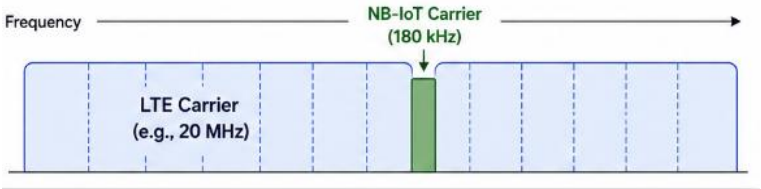


The actual implementation depends on the operator's network architecture. The control and user traffic may use different logical interfaces and network paths, even though they are part of the same overall cellular infrastructure.

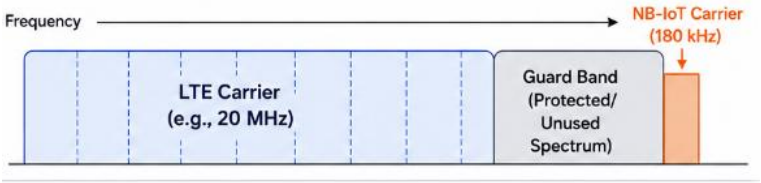
Easy way to remember
Control Plane = "Manage the connection"
User Plane = "Carry the data"

6.3 NB-IoT Deployment

1. In-Band Deployment
NB-IoT operates **inside an existing LTE carrier**, using one of the available LTE resource blocks.
Benefit: Reuses existing LTE spectrum and infrastructure



2. Guard-Band Deployment
NB-IoT operates in the **guard band at the edge of an LTE carrier**.
Benefit: Utilizes spectrum that may otherwise have limited use.



3. Standalone Deployment
NB-IoT operates using a **dedicated carrier**, independent of an LTE carrier.
Benefit: Suitable when dedicated spectrum is available for NB-IoT.

Why NB-IoT has 180KHz Bandwidth?

- 180 kHz is the standard NB-IoT carrier bandwidth for in-band and guard-band deployment.
- It is narrow because NB-IoT is designed for small data, low power, and IoT devices.
- 180 kHz is the bandwidth, not the operating frequency.
- The actual frequency can vary depending on the operator, country, and deployment band.
- In In-Band: the 180 kHz carrier is placed inside LTE.
- In Guard-Band: the 180 kHz carrier is placed at the edge of LTE.
- In Standalone: NB-IoT uses a separate carrier; approximately 200 kHz of spectrum is required.

6.4 NB-IoT Band

An **NB-IoT band** refers to the **radio frequency band** in which an NB-IoT device operates.
It is different from the **180 kHz bandwidth**.

- **Band** → Where the NB-IoT signal operates
- **Bandwidth** → How wide the NB-IoT carrier is
- **180 kHz** → Standard NB-IoT carrier bandwidth for in-band/guard-band operation

NB-IOT FREQUENCY BANDS

GLOBAL

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
1	2140	1950	FDD
3	1842.5	1747.5	FDD
8	962.5	897.5	FDD
31	465	455	FDD

EMEA

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
20	800	847	FDD
72	463.5	453.5	FDD

NORTH AM. REGION

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
2	1960	1860	FDD
4	2132.5	1732.5	FDD
5	881.5	836.5	FDD
12	737.5	707.5	FDD
13	751	782	FDD
14	763	793	FDD
17	740	710	FDD
25	1932.5	1852.5	FDD
26	876.5	831.5	FDD
66	2140	1745	FDD
70	2007.5	1707.5	FDD
71	634.5	608.5	FDD
74	1462.5	1442.5	FDD
85	737	707	FDD

EU

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
28	758.5	703.5	FDD

APAC

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
28	758.5	703.5	FDD
79	463.5	453.5	FDD

JAPAN

Band	Downlink MHz Center f_0	Uplink MHz Center f_0	Duplex Mode
11	1485.9	1437.9	FDD
18	867.5	822.5	FDD
19	1862.5	1777.5	FDD
21	1500.4	1455.4	FDD

https://haltian.com/resources/frequency-bands-for-ten-most-popular-iot-protocols/

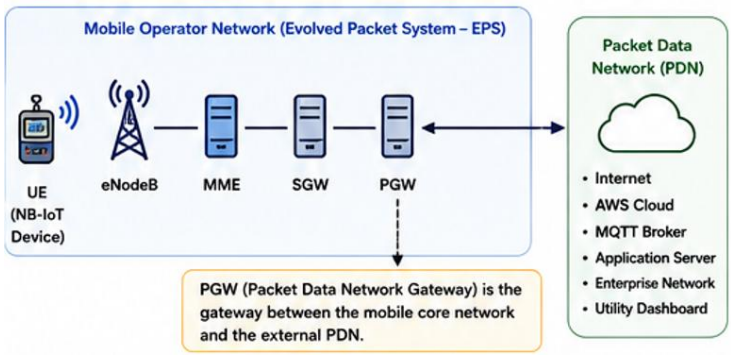
NB-IoT FREQUENCY PLAN – DETAILS

Item	Description
Bandwidth	NB-IoT uses 180 kHz (1 Resource Block)
Subcarrier Spacing	15 kHz
Modulation	BPSK / QPSK
Duplex Mode	FDD (most common) & TDD (supported in some bands)
Transmission Modes	Standalone, Guard-Band, In-Band
Data Rates	Uplink up to ~60 kbps Downlink up to ~250 kbps
Coverage Gain	Up to 20 dB better than LTE
Link Budget	~164 dB (Excellent for deep indoor & rural coverage)

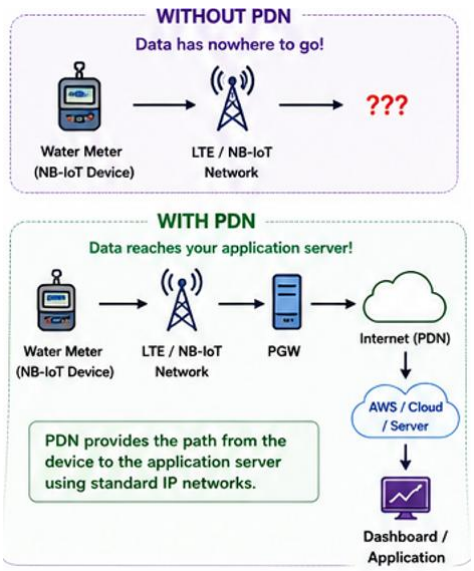
6.5 PDN

PDN (Packet Data Network) is the external data network that connects the cellular core network to the Internet, cloud, IoT platform, or enterprise application. The PDN is basically the destination network for the device's IP data.

PDN (Packet Data Network) is the external IP network, such as the Internet, private enterprise network, or cloud, to which the cellular core connects the IoT device for data communication.



Without PDN and with PDN:



Without PDN, the application can use cellular network-level services to interact with the NB-IoT device without establishing a conventional IP data connection.

A PDN connection is needed whenever the device must exchange IP-based data with an application server, cloud platform, or IoT service.

Without a PDN connection, the device cannot use normal IP-based applications such as MQTT, HTTP, or HTTPS over that IP data path.

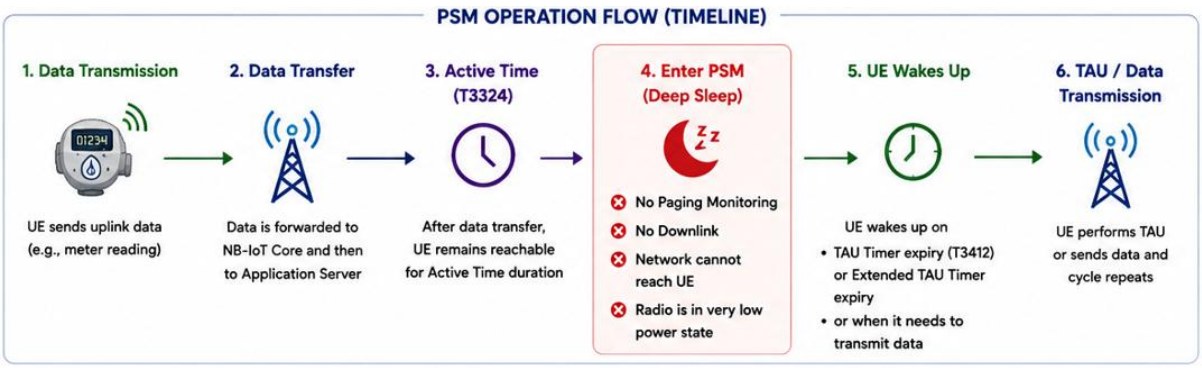
6.6 PSM and eDRX

Both **PSM (Power Saving Mode)** and **eDRX (Extended Discontinuous Reception)** reduce battery consumption, but the key difference is whether the network can reach the device while it is sleeping.

1. PSM — Power Saving Mode

In PSM, the NB-IoT UE first completes its data transmission and remains reachable during the configured Active Time (T3324). After the Active Time expires, the UE enters a deep power-saving state. While in PSM, the UE does not monitor paging, so the network cannot initiate downlink communication with it. The UE wakes up later according to its configured TAU timing or when it needs to transmit data, providing maximum battery savings.

Flow:
Data Transmission → Active Time → PSM Deep Sleep → No Paging/Downlink → UE Wakes → TAU/Data Transmission



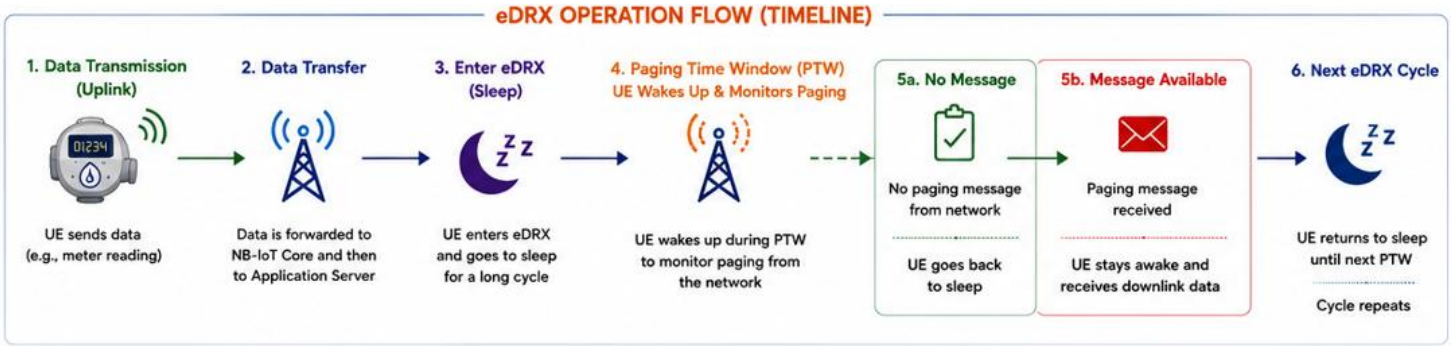
2. eDRX — Extended Discontinuous Reception

In eDRX, the NB-IoT UE remains in an extended sleep state for a long period but periodically wakes during the Paging Time Window (PTW) to check whether the network has a paging message. If there is no message, the UE returns to sleep; if a message is available, the UE remains active and receives the downlink data. This provides a balance between battery saving and network reachability.

Flow:
Data Transmission → eDRX Sleep → Paging Time Window → Check Paging → No Message: Sleep Again / Message: Receive Data

Key difference

PSM: Maximum battery saving → UE is unreachable during PSM.
eDRX: High battery saving → UE is periodically reachable during paging windows.



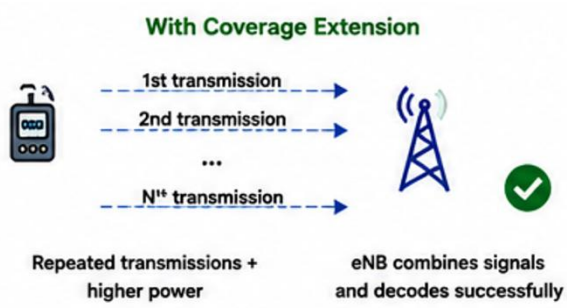
6.7 Coverage Extension and Power Class

1. Coverage Extension

Coverage Extension allows NB-IoT devices to operate in locations where the normal radio signal is weak, such as basements, indoor utility rooms, underground areas, and locations far from the eNB.

The network improves the probability of receiving the signal by using techniques such as:

- **Signal repetitions**
- More robust transmission
- Repeated NPDCCH/NPDSCH transmissions
- Lower data rate when required

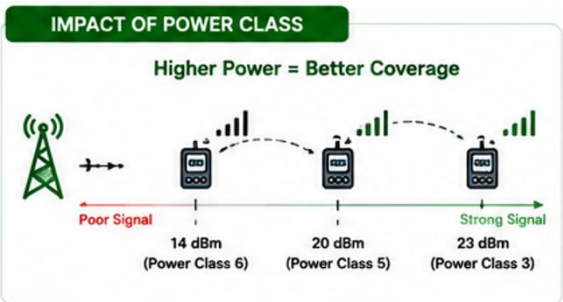


2. Power Class

Power Class defines the **maximum transmit power that an NB-IoT UE is allowed to use**.

The UE's power class affects:

- Maximum transmit power
- Coverage capability
- Battery consumption
- Device thermal/load requirements



Type of Power Class

1. Class 3 — 23 dBm

Highest of the three NB-IoT power classes.

- Maximum TX power: **23 dBm**
- Better uplink link budget
- Suitable where coverage is important
- Higher potential battery consumption during transmission

2. Class 5 — 20 dBm

A **lower-power alternative** to Class 3.

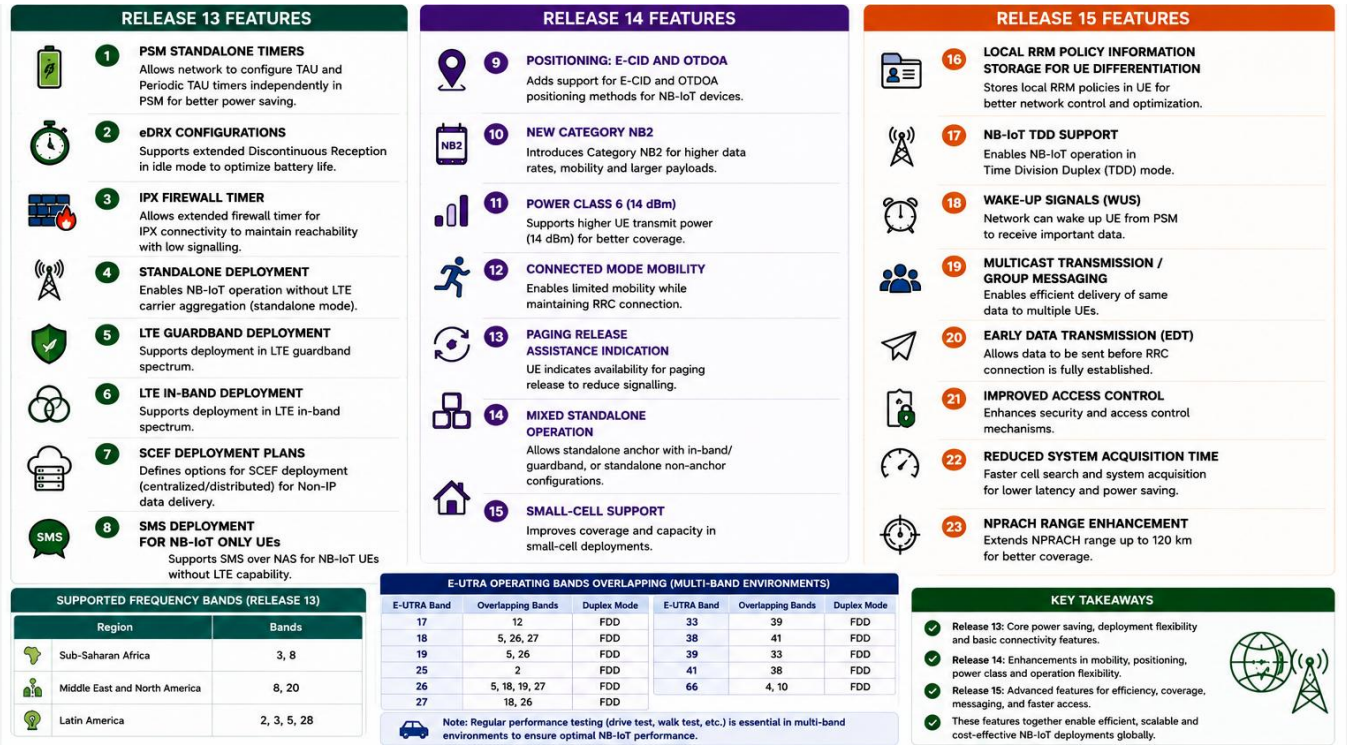
- Maximum TX power: **20 dBm**
- Lower TX power than Class 3
- Can reduce device power requirements
- Some reduction in uplink link budget compared with Class 3

3. Class 6 — 14 dBm

Introduced with **Release 14** for very low-power devices. The reduction from 23 dBm to 14 dBm can significantly reduce peak current requirements, making it suitable for devices using smaller batteries.

- Maximum TX power: **14 dBm**
- Very low transmit power
- Lower peak current
- Smaller/simpler battery options can be used
- Reduced coverage/link budget compared with higher-power classes

6.8 Release 13,14,15 Features



6.9 NB-IoT Device Integration with Cloud

If we are integrating an **NB-IoT device/SIM with an application server**, we should treat the ISP/MNO as providing the **cellular connectivity + data transport**, while **we provide the application endpoint/server and device-side configuration**.

The exact information depends on whether the ISP gives you **public Internet APN, private APN + VPN, or an IoT platform/API**. Private APN/VPN is commonly used when the device traffic needs to reach infrastructure securely.

1. Information you need to collect from the NB-IoT ISP

A. SIM / Subscription information

- SIM type: **NB-IoT only / NB-IoT + LTE-M / NB-IoT + 4G**
- IMSI

- ICCID
- MSISDN, if applicable
- SIM activation status
- IMEI binding requirement
- Number of SIMs
- Data plan / monthly data allowance
- Data overage charges
- SIM lifecycle management
- Roaming support, if required
- Supported NB-IoT bands in your deployment area

The ISP should also confirm that the SIM is provisioned for NB-IoT. APN configuration is a key part of cellular IoT connectivity.

B. APN information — very important

What APN should be configured in the NB-IoT device?

For example:

- APN: xyz.iot
- Username: xxx
- Password: xxx

Authentication: PAP / CHAP / None

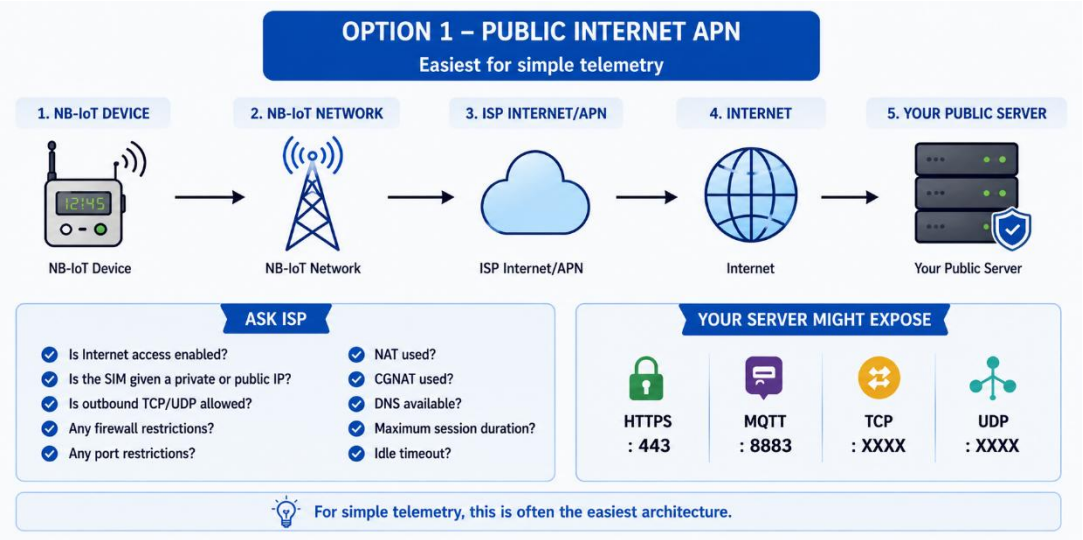
Also ask:

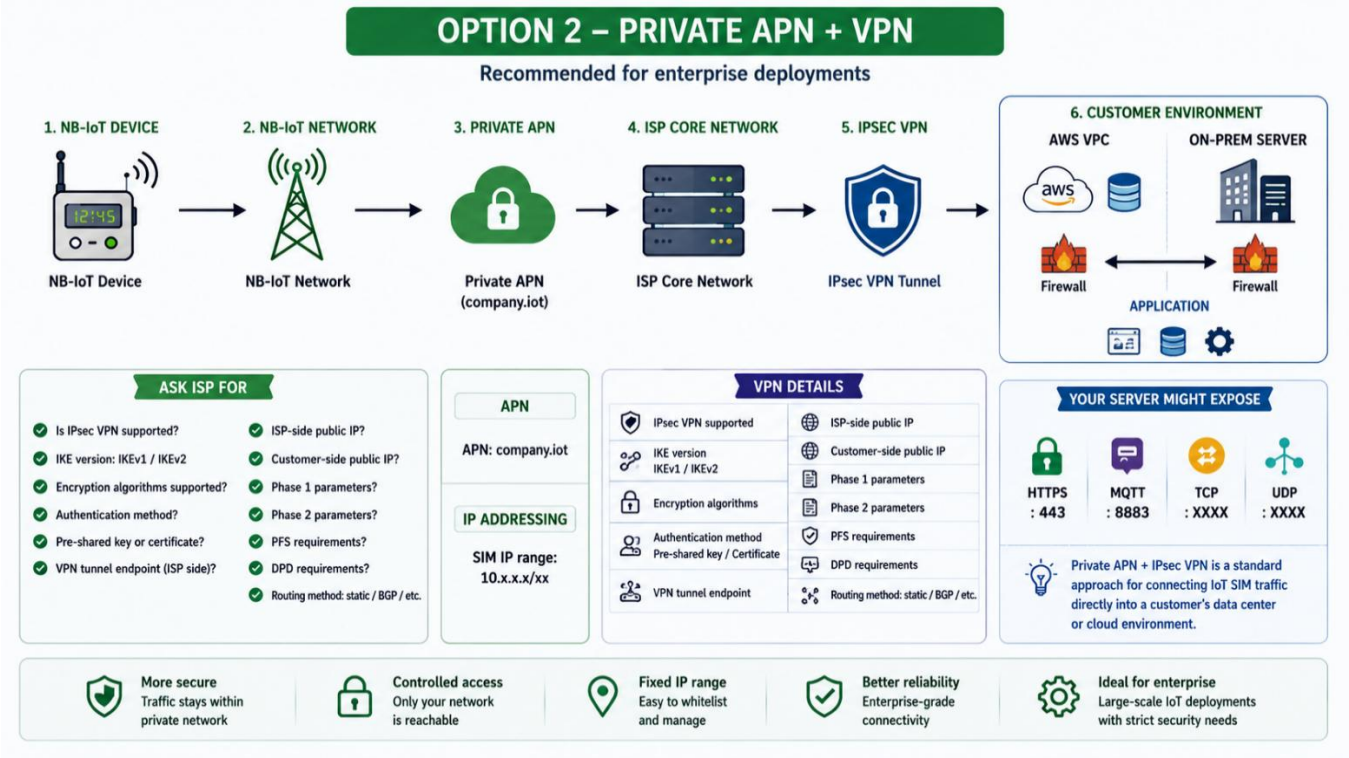
- Public APN or Private APN?
- Dynamic IP or Static IP?
- Private IP range?
- IPv4 / IPv6 / dual stack?
- Can devices communicate directly with our server?
- Is inbound communication to the device supported?
- Is device-to-device communication allowed?

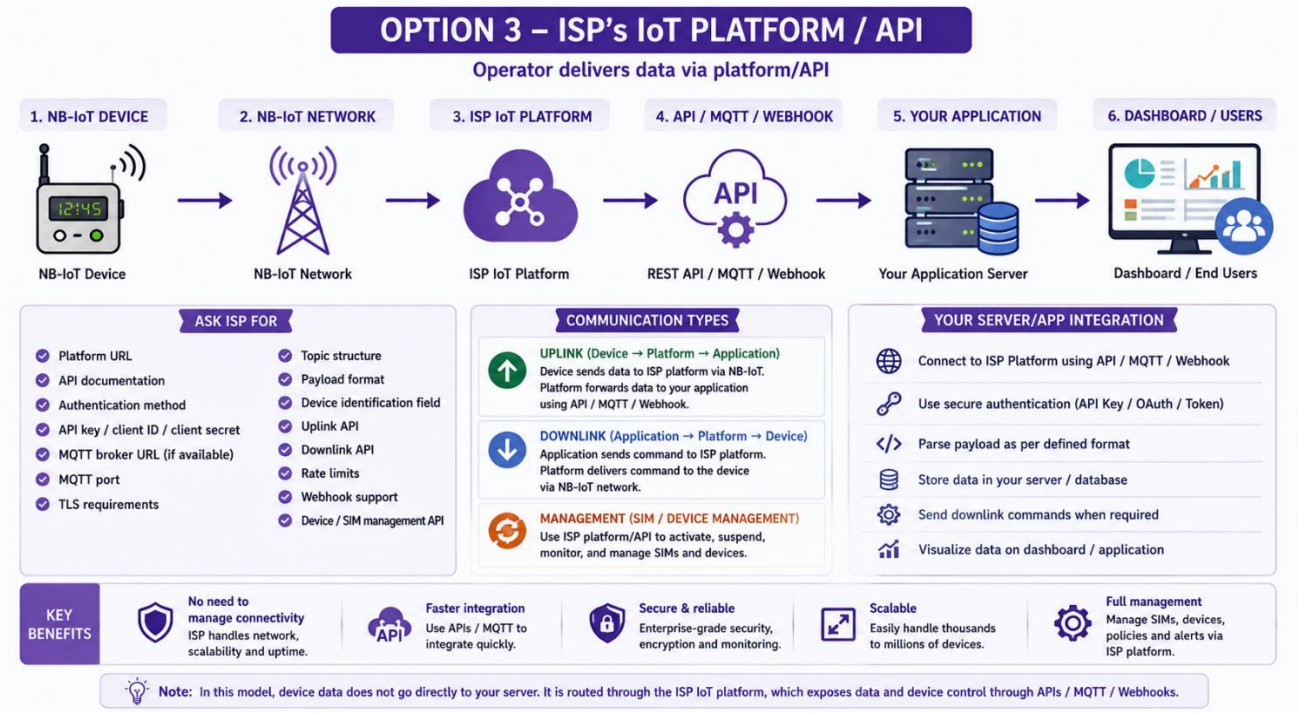
Some IoT providers use private IP ranges and route SIM traffic through a customer VPN rather than exposing devices directly to the Internet.

2. Most important question: How will NB-IoT data reach our server?

There are typically **3 architectures**.







3. Information you need to provide to ISP

Your side should provide the ISP with:

Server information

- Server type: AWS / Azure / On-Prem
- Server IP: x.x.x.x
- Server hostname: xxx.company.com

Required ports and protocols

Communication	Transport Protocol	Port
HTTPS	TCP	443
MQTT over TLS	TCP	8883
MQTT	TCP	1883
CoAP	UDP	5683
CoAP over DTLS	UDP	5684
Custom TCP	TCP	XXXX
Custom UDP	UDP	XXXX

Server-side security

Provide:

- Public IP
- Domain/FQDN
- Firewall rules
- Allowed source IP range
- TLS certificate requirements
- VPN endpoint
- Your VPN public IP
- Required routing
- Required ports

4. Device information you should provide

- Give the ISP/device vendor:
- Device Manufacturer:
- Device Model:
- NB-IoT Module:
- Module IMEI:
- SIM ICCID:
- Firmware Version:
- Supported Bands:
- Protocol:
- Data Payload:

5. NB-IoT radio/network parameters

PARAMETER	ASK ISP	DETAILS / EXAMPLES
NB-IoT supported	Yes/No	Confirm if NB-IoT is supported in the deployed network.
NB-IoT bands	Which bands?	e.g., B1/B3/B5/B8/B20/B28 etc.
LTE Band	Which LTE bands are used?	e.g., B3/B5/B8/B28 etc.
NB-IoT mode	In-band / guard-band / standalone?	Specify the deployed mode.
Coverage	Your deployment locations?	Confirm coverage at all required locations.
Roaming	Required/available?	Domestic / International roaming support?
NB-IoT release	NB-IoT Release?	e.g., Release 13 / 14 / 15
PSM	Supported?	Power Saving Mode support
eDRX	Supported?	Extended Discontinuous Reception support
Extended TAU	Supported?	Extended Tracking Area Update support
Maximum payload	Limits?	Maximum uplink payload size (bytes)
Downlink	Supported?	Downlink (mobile terminated) support?
SMS	Supported?	SMS over NB-IoT support?
IP connectivity	Yes/No?	Does the SIM have IP connectivity?
Non-IP Data Delivery	Supported?	Support for Non-IP Data Delivery (NIDD)?

NB-IoT can use IP data delivery, while some deployments can also use non-IP data delivery; therefore you should explicitly confirm which connectivity model your ISP supports.

6. Data/payload information

- This is often missed.
- Your device team should define:
- Device ID
- Timestamp
- Meter reading
- Battery
- Signal strength
- Alarm
- Temperature
- etc.

Example:

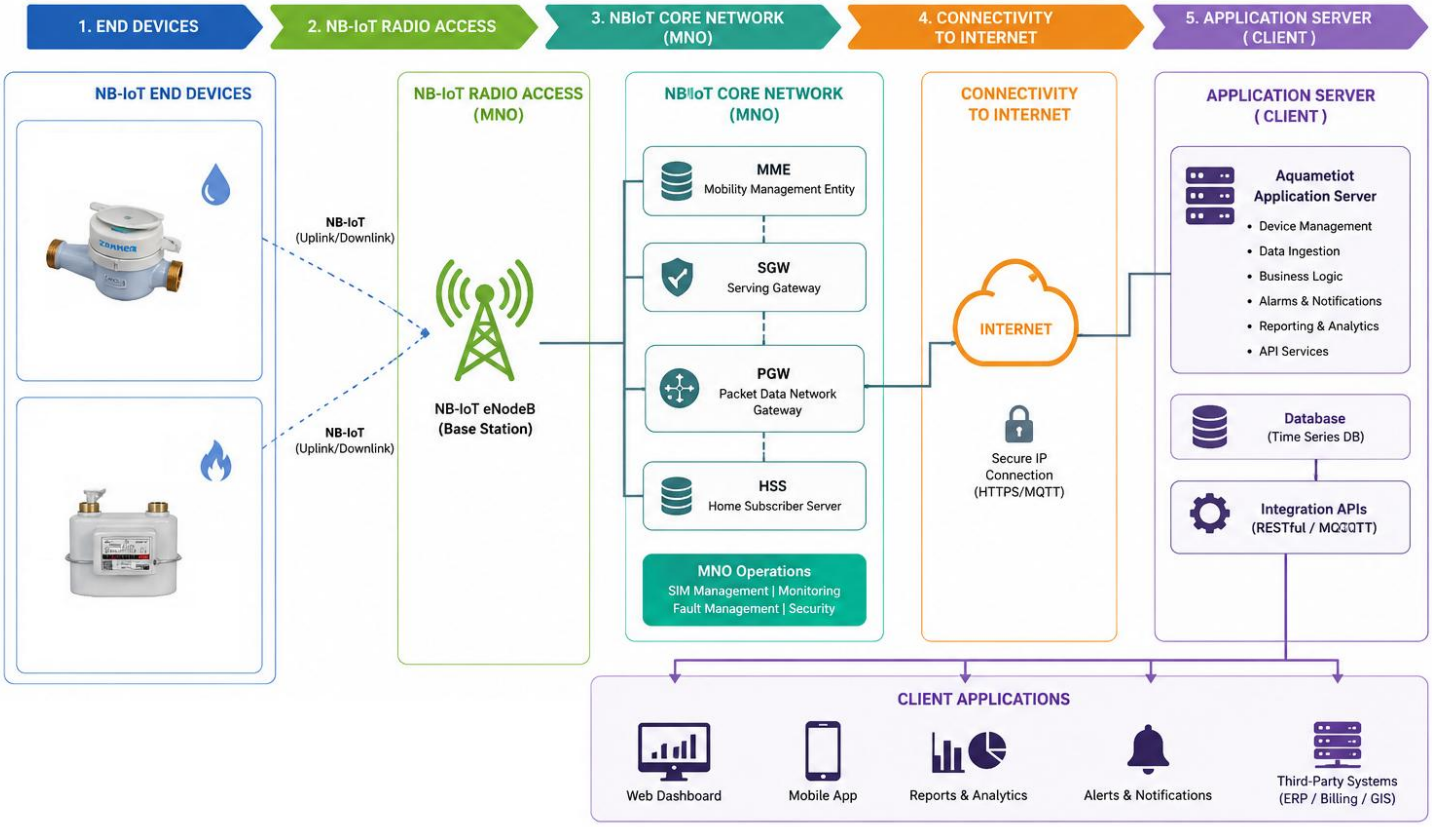
```
{
 "device_id": "WM000123",
 "timestamp": "2026-09-03T10:30:00Z",
 "meter_reading": 125.6,
 "battery": 82,
 "rssi": -85
}
```

But **the ISP normally doesn't need to understand your application JSON** if they are simply providing IP connectivity. They mainly need to provide the network path.

7. Don't forget downlink

Ask ISP: Does your NB-IoT service support **mobile-terminated/downlink IP communication**?

NB-IoT ARCHITECTURE



Chapter 7

IoT Application Protocols

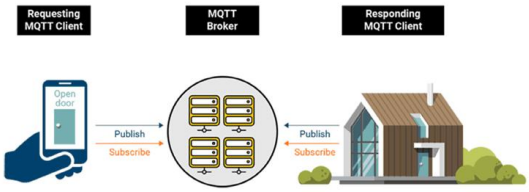
IoT application protocols define **how an IoT device, gateway, application server, or cloud platform exchanges application-level information**. They operate above the underlying connectivity technologies such as Ethernet, Wi-Fi, LoRaWAN, NB-IoT, or cellular IP.
Different applications require different communication models. MQTT is well suited to telemetry and event driven IoT, HTTP/HTTPS is common for APIs and web services, WebSocket supports continuous bidirectional communication.

7.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight publish/subscribe messaging protocol widely used for IoT telemetry.
Instead of a device directly sending data to every application, devices communicate through an MQTT Broker.
The main components are:

- **Publisher** – sends messages
- **Broker** – receives and distributes messages
- **Subscriber** – receives messages
- **Topic** – identifies the message channel

Example:
Water Meter → Publish → Broker → Subscribe → IoT Platform



MQTT Publish/Subscribe Architecture

For additional technical information and practical examples on MQTT, refer to HiveMQ's MQTT Essentials guide: https://www.hivemq.com/mqtt/

7.1.1 Key Features

1. MQTT Clients

MQTT clients are devices or applications that publish messages to an MQTT broker or subscribe to messages they want to receive. MQTT clients have a lightweight design, making them suitable for resource-constrained devices and networks with limited bandwidth.

2. MQTT Broker

The MQTT broker receives messages published by clients and forwards them to the appropriate subscribed clients. It uses MQTT topics and subscription information to determine which clients should receive each message.

3. MQTT QoS Levels

MQTT provides three Quality of Service (QoS) levels that define the message-delivery guarantee between the sender and receiver:

- QoS 0 – At Most Once: Message is delivered without an acknowledgement.
- QoS 1 – At Least Once: Message is acknowledged, but duplicates may occur.
- QoS 2 – Exactly Once: Provides the highest delivery guarantee and prevents duplicate delivery.

4. Persistent Sessions

MQTT can maintain a persistent session between the client and broker. Session information can remain available even when the client temporarily loses network connectivity. When the client reconnects, the broker can use the stored session information to continue communication.

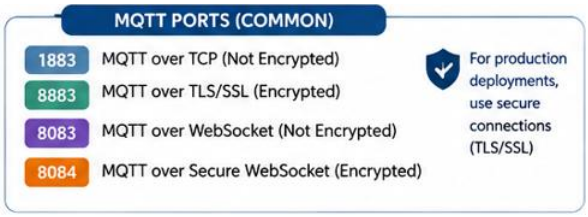
5. Retained Messages

An MQTT broker can store a retained message for a topic. When a new client subscribes to that topic, the broker can immediately send the latest retained message.

6. Last Will and Testament (LWT)

MQTT provides a Last Will and Testament (LWT) mechanism. A client can configure a message that the broker publishes if the client disconnects unexpectedly.
For example:
Device unexpectedly disconnects → Broker publishes → Device Offline
This allows the IoT platform to detect unexpected device disconnections and update the device status accordingly.

7.1.2 MQTT Ports



7.1.3 Implementation of MQTT Protocols

1. MQTT Broker Details

First, you need an MQTT broker that the device can connect, you need to provide the below details:

- **Broker Hostname/IP**
- **MQTT Port**
- **Protocol** — MQTT/TCP or MQTT/TLS
- **TLS enabled/disabled**

2. Device Authentication

Configure how the device will authenticate with the MQTT broker.

Common options:

- Username + Password
- Client Certificate
- Username + Password + TLS
- Mutual TLS (client + server certificates)

3. Client ID

Give each device a **unique MQTT Client ID**.
This is important when you have many devices.

For example:

- WATERMETER_001
- WATERMETER_002
- WATERMETER_003

4. Publish Topic

You need to know **which MQTT topic the device publishes its data to**.

- For example: watermeter/001/telemetry

5. Subscribe Topic

If your application needs to **send commands/configuration back to the device**, you also need the device's subscribe topic.

- Example: watermeter/001/command

6. QoS

Confirm which MQTT QoS the device uses.

Typical options:

- **QoS 0** – At most once
- **QoS 1** – At least once
- **QoS 2** – Exactly once

For meter data, **QoS 1** is commonly considered when reliable delivery is important, but you should use whatever the device/application requirements specify.

7. Payload Format

This is one of the **most important integration parameters**. You need to know exactly what the device sends. Example:

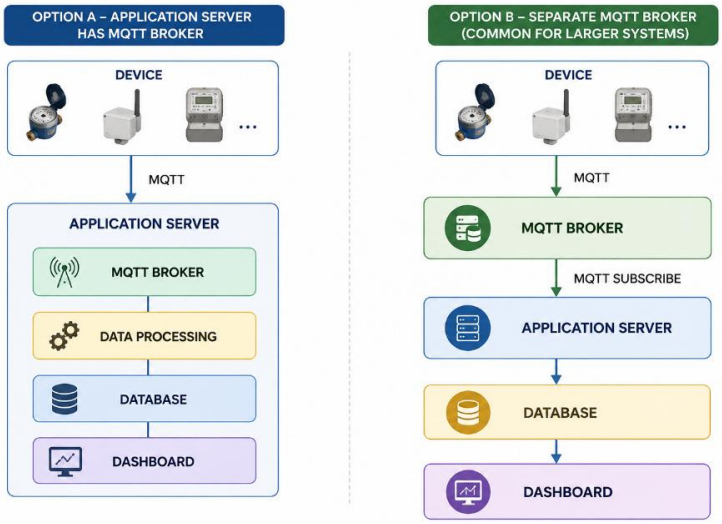
```
{
 "deviceId": "WM001",
 "reading": 125.6,
 "battery": 87
}
```

Or the device may send a Hex payload such as: 01027D6057. If it is Hex payload, you need the **payload decoding specification from the device manufacturer**.

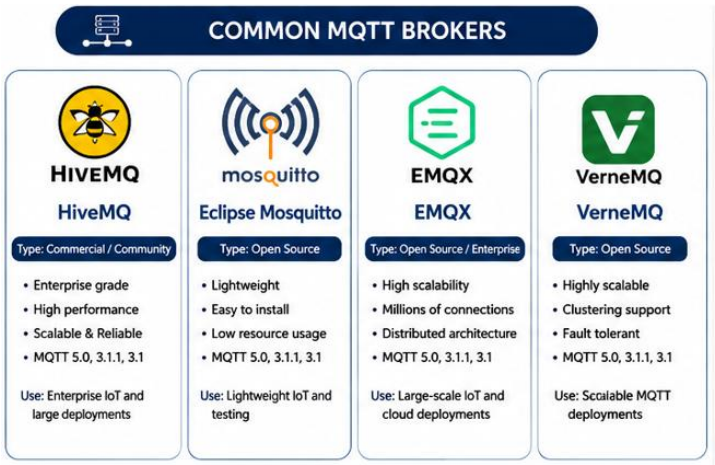
8. Application Server / MQTT Broker Architecture

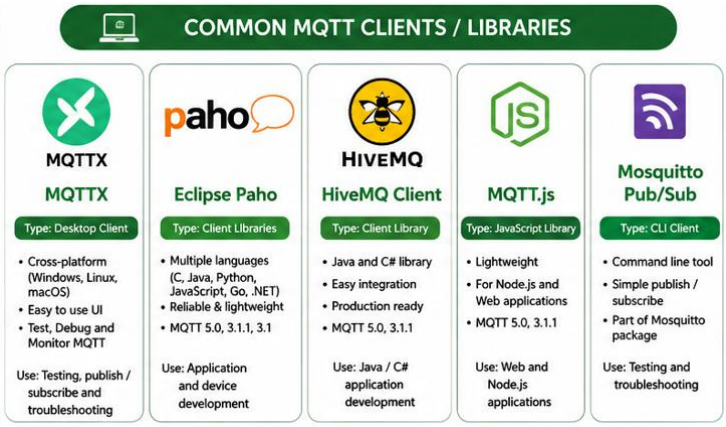
There are two common approaches.
Option A – Application Server Has MQTT Broker
Option B – Separate MQTT Broker

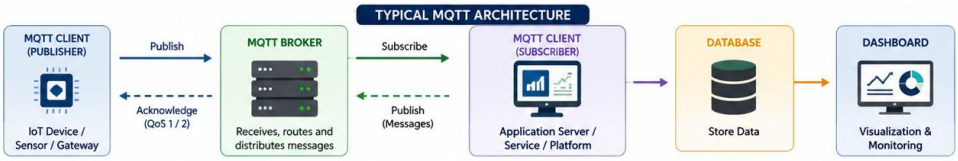
- In this case, your application server does **not necessarily connect directly to the device**. It subscribes to the MQTT broker.



7.1.4 List of MQTT Brokers and Clients







7.2 HTTP/HTTPS

HTTP and HTTPS are application-layer protocols commonly used for communication between IoT devices, gateways, application servers, and cloud platforms. They are especially useful when an IoT device needs to send data to a web service or REST API.

A typical IoT communication flow is:
IoT Device/Gateway → IP Network → HTTP/HTTPS → Application Server → Database/Dashboard

7.2.1 HTTP – Hypertext Transfer Protocol

HTTP is a **request-response protocol** in which a client sends a request to a server and the server returns a response.

In an IoT system:

IoT Device/Gateway → HTTP Request → Application Server
Application Server → HTTP Response → IoT Device/Gateway
HTTP is commonly used when a device or gateway needs to send telemetry, retrieve configuration, or communicate with an application API.

HTTP Request

An HTTP request generally contains:

- HTTP method
- Server URL
- Endpoint
- Headers
- Payload/body, when required

Example:

POST /api/v1/device/data
The device can send sensor data in the request body.
Example JSON payload:

```
{
 "deviceId": "WM001",
 "temperature": 28.5,
 "battery": 87
}
```

Common HTTP Methods

Method	Purpose	IoT Example
GET	Retrieve information	Get device configuration
POST	Send/create data	Send sensor readings
PUT	Update information	Update device configuration
DELETE	Delete a resource	Delete a device record

HTTP Response

After receiving a request, the server returns a response containing a status code.

Status Code	Meaning
200 OK	Request successful
201 Created	Resource successfully created
400 Bad Request	Invalid request or payload
401 Unauthorized	Authentication failed/missing
403 Forbidden	Access not permitted
404 Not Found	Endpoint/resource not found
500 Internal Server Error	Server-side problem

HTTP Port

The conventional HTTP port is **Port 80**
Example: http://api.example.com:80
HTTP does **not provide TLS encryption by itself**, so sensitive IoT data should generally not be transmitted using plain HTTP in production environments.

7.2.2 HTTPS – HTTP Secure

HTTPS is **HTTP protected using TLS (Transport Layer Security)**.

It provides an encrypted communication channel between the IoT device/gateway and the application server.

IoT Device → TLS → HTTPS → Application Server

The conventional HTTPS port is **Port 443**

Example: https://api.example.com:443

HTTPS Security

HTTPS can provide:

- **Encryption** – Protects data during transmission
- **Server authentication** – Helps the device verify that it is communicating with the intended server
- **Data integrity** – Helps detect modification of data during transmission

A typical HTTPS connection involves verification of the server's TLS certificate before secure application communication takes place.

HTTPS Certificates

For device integration, the following may need to be configured:

- Server certificate
- CA certificate
- Certificate validation
- TLS version
- Hostname verification

For higher-security deployments, **mutual TLS (mTLS)** can be used, where both the server and device authenticate using certificates.

7.2.3 HTTP vs HTTPS

Parameter	HTTP	HTTPS
Encryption	❌ No TLS	✅ TLS
Typical Port	**80**	**443**
Data Protection	Low	High
Typical Use	Testing/internal systems	Production/API communication
Certificate	Not required	Required for normal TLS validation
IoT Production	Generally not preferred	**Recommended**

7.2.4 HTTP/HTTPS in IoT Integration

When onboarding an HTTP/HTTPS-enabled device to an application server, the System Integrator normally needs to obtain and configure:

Parameter	Example
Protocol	HTTPS
Server URL	`https://api.example.com`
Port	443
API Endpoint	`/api/v1/device/data`
HTTP Method	POST
Authentication	API Key / Token
TLS	Enabled
CA Certificate	Required
Content-Type	`application/json`
Payload	JSON
Device ID	WM001

The complete communication can be:

IoT Device → 4G/Wi-Fi → IP Network → HTTPS:443 → REST API → Application Server → Database → Dashboard

7.3 REST API

7.3.1 What is REST API?

REST (Representational State Transfer) is an architectural style used to design APIs for communication between different systems. In IoT, REST APIs are commonly implemented using **HTTP/HTTPS**.

A REST API allows an IoT device, gateway, application, or cloud platform to exchange information using defined endpoints and HTTP methods.

Typical flow:
IoT Device/Gateway → HTTPS → REST API → Application Server → Database

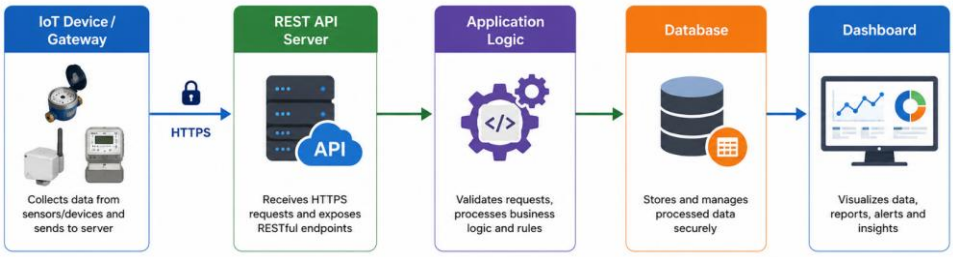
REST is particularly useful when a device or gateway needs to send data to an application server or retrieve configuration from it.

7.3.2 REST API Architecture

A REST-based IoT architecture generally consists of:

- **Client** – Device, gateway, application, or dashboard
- **API Server** – Receives and processes requests
- **Endpoint** – Defines the specific resource or operation
- **Database** – Stores device and application data
- **Authentication mechanism** – Controls access to the API

Example:



The client does not normally need to know how the application internally processes or stores the data. It only needs to follow the API specification.

7.3.3 HTTP Methods

REST APIs commonly use HTTP methods to define the required operation.

Method	Purpose	IoT Example
GET	Retrieve information	Read device configuration
POST	Send/create information	Send sensor telemetry
PUT	Update/replace information	Update device configuration
DELETE	Delete information	Remove a device

Example

A gateway sends meter data: POST /api/v1/device/data
The server receives the request and processes the device data.
A device configuration can be retrieved using: GET /api/v1/device/WM001/config

7.3.4 REST API Request and Response

A REST request normally contains:

- HTTP method
- URL
- Endpoint
- Headers
- Authentication information
- Request body/payload, where required

Example:

POST /api/v1/device/data

Content-Type: application/json

Authorization: Bearer <token>

Payload:

```
{
 "deviceId": "WM001",
 "reading": 125.6,
 "battery": 87
}
```

The server processes the request and returns a response.

Example: HTTP/1.1 200 OK

The response may also contain information such as an acknowledgement, error message, or requested data.

7.3.5 API Endpoint and URL

An **API endpoint** identifies a specific resource or operation provided by the server.

Example:

https://api.example.com/api/v1/device/data

Here:

- https:// → Communication scheme
- api.example.com → Server hostname
- /api/v1/ → API path/version
- /device/data → Endpoint

The exact endpoint structure is defined by the application or API provider.

For system integration, the **complete API specification** should be obtained from the platform or application owner.

7.3.6 Authentication and Authorization

REST APIs normally require mechanisms to control who can access the API.

Common methods include:

- API Key
- Username/password
- Bearer token
- OAuth-based authentication
- Client certificate/mTLS

7.3.7 REST API Payload

The payload contains the actual information being exchanged.

JSON is commonly used as the payload format.

Example:

```
{
 "deviceId": "WM001",
 "timestamp": "2026-09-03T10:30:00Z",
 "consumption": 125.6,
 "battery": 87
}
```

7.3.8 REST API Status Codes

The HTTP response status code helps identify the result of an API request.

Code	Meaning	Possible Integration Issue
200	Successful request	—
201	Resource created	—
400	Bad request	Incorrect payload/request
401	Unauthorized	Authentication problem
403	Forbidden	Permission problem
404	Not found	Incorrect endpoint
408	Request timeout	Server/network timeout
429	Too many requests	Rate limit
500	Server error	Application/server issue
503	Service unavailable	Server/service unavailable

For troubleshooting, the status code should be checked before assuming that the cellular or network connection is the problem.

7.4 WebSocket

7.4.1 What is WebSocket?

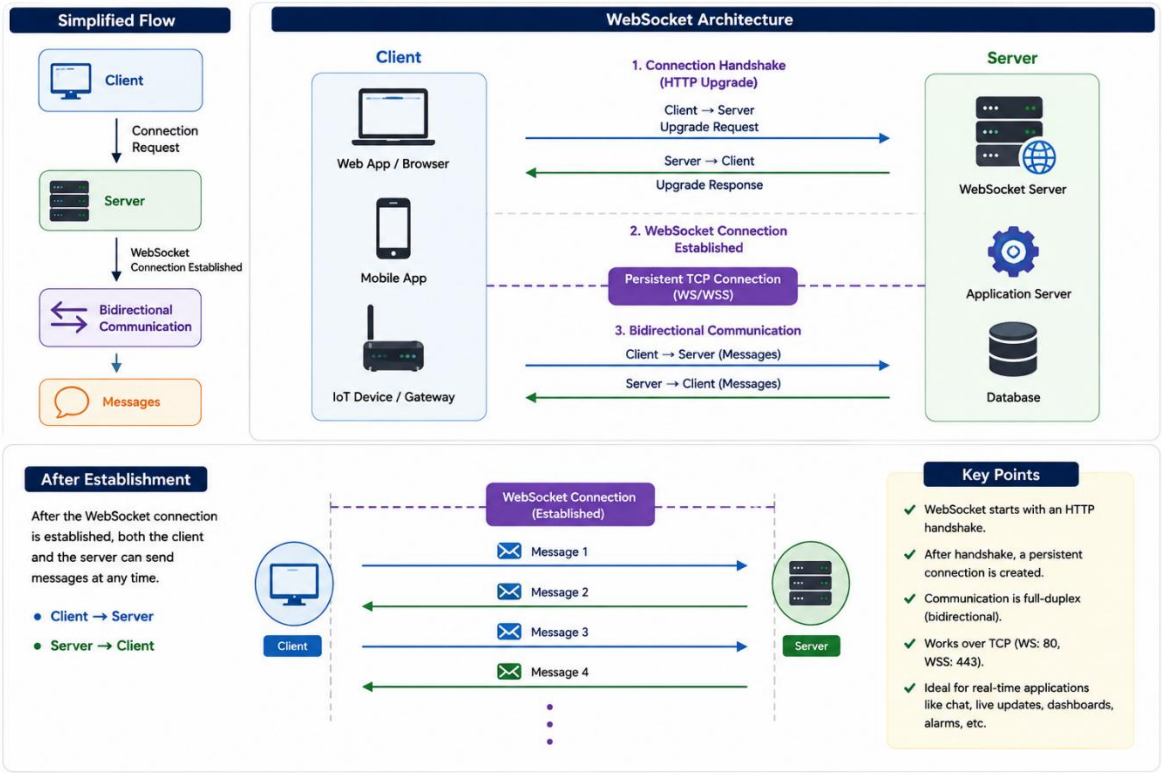
WebSocket is a communication protocol that provides a **persistent, bidirectional connection** between a client and server.

Once the connection is established, both the client and server can send messages without creating a separate request for every message.
Typical architecture: IoT Platform →WebSocket →Dashboard / Application

It is particularly useful for **real-time applications**.

7.4.2 WebSocket Communication

WebSocket communication starts with an initial HTTP-based handshake. After the connection is successfully established, communication continues using the WebSocket protocol.



7.4.3 WebSocket Connection Establishment

The client first connects to the WebSocket server. The connection then transitions from the initial HTTP handshake to a WebSocket connection. The connection remains active until the client or server closes it or the network connection is lost.

7.4.4 Bidirectional Communication

The major advantage of WebSocket is that the **server can send information to the client without waiting for a new request**.

For example: **Dashboard ↔ IoT Platform**

When a new alarm is generated:

IoT Platform → WebSocket → Dashboard

The dashboard can immediately display the alarm.

This is different from continuously polling the server using HTTP.

7.4.5 WebSocket Security

Two common WebSocket schemes are:

- **WS → WebSocket - 80**
- **WSS → WebSocket Secure - 443**

WSS uses TLS to protect the connection.

For production applications, **WSS/TLS** is generally preferred.

7.5 JSON

7.5.1 What is JSON?

JSON (JavaScript Object Notation) is a lightweight **data representation format** used to structure information exchanged between systems.

JSON is **not a communication protocol**.

It can be transported using:

- **MQTT + JSON**
- **HTTPS + JSON**
- **WebSocket + JSON**

7.5.2 JSON Structure

A JSON object consists of **key-value pairs**.

Example:

```
{
  "deviceId": "WM001",
  "temperature": 28.5,
  "battery": 87
}
```

Here:

- deviceId → Key
- "WM001" → String value
- temperature → Key
- 28.5 → Numeric value
- battery → Key
- 87 → Numeric value

7.5.3 JSON Data Types

Common JSON data types are:

- String
- Number
- Boolean
- Object
- Array
- Null

Example:

```
{
  "deviceId": "WM001",
  "active": true,
  "temperature": 28.5,
  "alarms": ["lowBattery", "leak"]
}
```

7.5.4 JSON Payload in IoT

JSON is frequently used for IoT telemetry because it is easy for both humans and software applications to understand.

Example:

```
{
  "deviceId": "WM001",
  "timestamp": "2026-09-03T10:30:00Z",
  "consumption": 125.6,
  "battery": 87,
  "status": "normal"
}
```

The application server can parse the payload and store individual values in the database.

7.5.5 JSON with MQTT and HTTP

JSON does not replace MQTT or HTTP.

Instead, JSON is carried **inside** the communication protocol.

MQTT + JSON

Device →MQTT → Broker →JSON Payload →Application

HTTPS + JSON

Device → HTTPS →REST API →JSON Payload →Application Server

This distinction is important when troubleshooting.

If MQTT connectivity is successful but the JSON format is incorrect, the problem is at the **payload/application layer**, not necessarily the MQTT connection.

7.6 TCP and UDP

7.6.1 What is TCP?

TCP (Transmission Control Protocol) is a **connection-oriented transport-layer protocol** that provides reliable and ordered data delivery between endpoints.

TCP is used underneath several common IoT application protocols.

Examples:

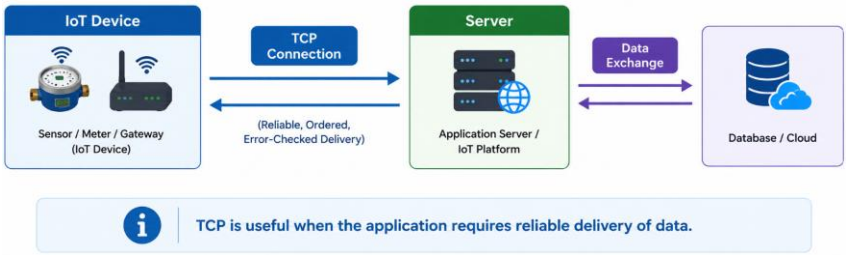
- **MQTT → TCP**
- **HTTP/HTTPS → TCP**
- **AMQP → TCP**

7.6.2 TCP Characteristics

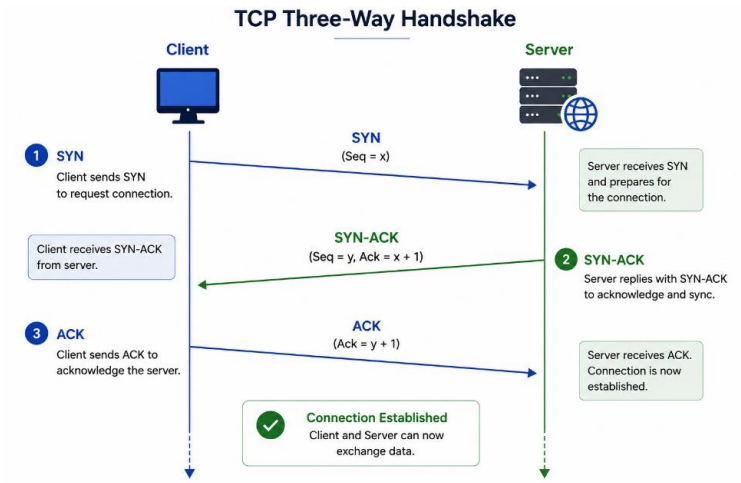
TCP provides:

- Connection establishment
- Reliable delivery
- Ordered data
- Error detection
- Retransmission
- Flow control
- Congestion control

A simplified communication flow is:



7.6.3 TCP Connection Establishment



After the connection is established, application data can be exchanged.

For a System Integrator, this is useful when diagnosing whether a device can establish a transport connection to the server.

7.6.4 TCP Reliability and Retransmission

TCP detects lost or incorrectly received data and can retransmit data when required.

It also maintains the correct order of the byte stream delivered to the application.

7.6.5 What is UDP?

UDP (User Datagram Protocol) is a **connectionless transport-layer protocol** that sends independent datagrams without establishing a TCP-style connection.

A simplified flow is:

IoT Device →UDP Datagram →Server

UDP itself does not guarantee:

- Delivery
- Ordering
- Retransmission
- Duplicate prevention

If an application requires these capabilities, they must be implemented by the application protocol or application itself.

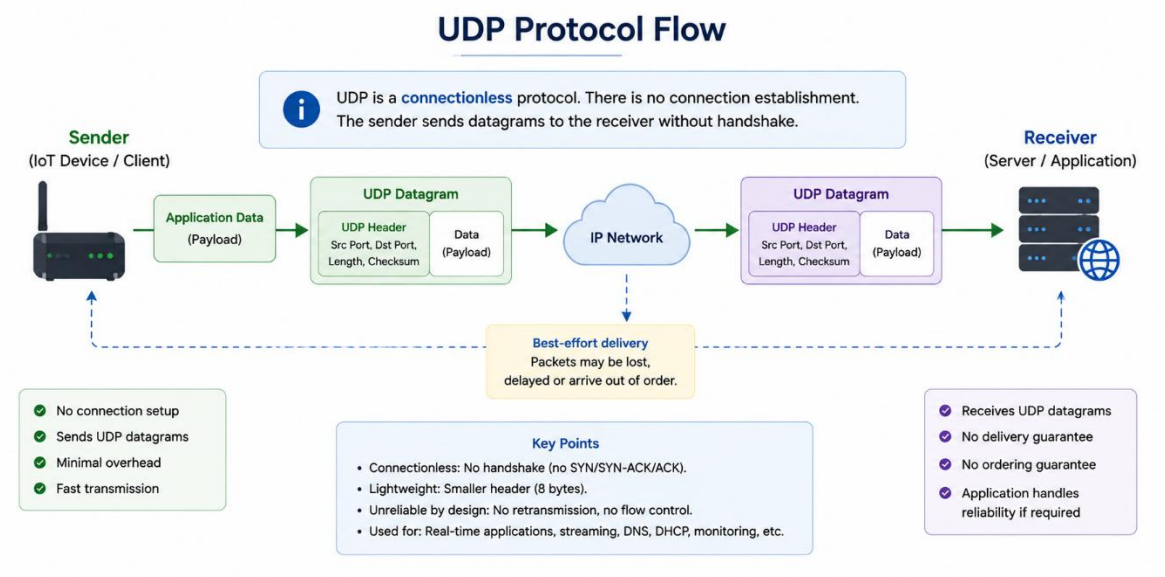
7.6.6 UDP Characteristics

Key characteristics include:

- Connectionless
- Low protocol overhead

- No TCP-style handshake
- Datagram-based communication
- No built-in delivery guarantee
- No built-in ordering guarantee

UDP can be useful where low overhead, simple communication, or application-controlled reliability is required.



7.6.7 TCP vs UDP

Feature	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Connection Type	Connection-oriented	Connectionless
Connection Establishment	Requires three-way handshake (SYN, SYN-ACK, ACK)	No handshake
Reliability	Reliable – ensures delivery	Unreliable – no delivery guarantee
Ordering	Maintains order of data	No ordering guarantee
Retransmission	Yes – lost data is retransmitted	No retransmission
Flow Control	Yes	No
Congestion Control	Yes	No
Header Size	Larger (20–60 bytes)	Smaller (8 bytes)
Speed / Latency	Slower due to connection, ACKs, retransmissions	Faster – low latency
Overhead	Higher	Lower
Use Case	When reliability and complete data delivery is important	When speed and low overhead are important (reliability handled by application if needed)
Typical IoT Protocols	MQTT, HTTP/HTTPS, AMQP, CoAP (over TCP)	DNS, NTP, CoAP (over UDP), Custom IoT protocols, Video/Audio streaming, Discovery protocols
Common Ports	HTTP (80), HTTPS (443), MQTT (1883), MQTTS (8883), AMQP (5672)	DNS (53), NTP (123), CoAP (5683), DHCP (67/68)
Summary	Use TCP when you need **reliable, ordered** and **complete** data delivery.	Use UDP when you need **faster communication** and can tolerate some data loss.

Chapter 8

Networking

Networking for System Integrators

Networking is the foundation of an IoT system. A System Integrator must understand how devices obtain network connectivity, receive IP addresses, communicate through ports, resolve server names, and securely reach application or cloud servers.

A typical IoT communication path is:

IoT Device / Gateway →Ethernet / Wi-Fi / Cellular →IP Network →Router / Gateway →Firewall / VPN →Internet / Private Network →Application / Cloud Server

The objective is not to become a network engineer, but to understand enough networking to **design connectivity, collect network requirements, commission devices, and troubleshoot communication failures**.

8.1 OSI and TCP/IP Models

OSI Model

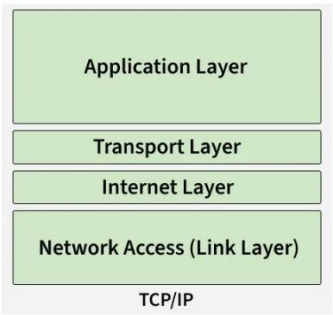
The **OSI (Open Systems Interconnection)** model divides network communication into seven layers.

Layer	Name	Examples
7	Application	HTTP, MQTT, DNS
6	Presentation	Data format, encryption
5	Session	Session management
4	Transport	TCP, UDP
3	Network	IP, Routing
2	Data Link	Ethernet, MAC
1	Physical	Cable, radio, connectors

For IoT integration, the most frequently encountered layers are:

Physical → Ethernet/Wi-Fi → IP → TCP/UDP → MQTT/HTTP → Application Data

TCP/IP Model



8.2 Ethernet and IP Addressing

Ethernet

Ethernet is a common wired LAN technology used to connect IoT gateways, controllers, servers, switches and other network equipment.

When commissioning an Ethernet device, check:

- Ethernet cable
- Link/activity status
- IP address
- Subnet mask
- Default gateway
- DNS server
- DHCP/static configuration
- VLAN requirements
- Firewall restrictions

Important: A device having an Ethernet link does not necessarily mean that it has Internet connectivity.

IP Addressing

An **IP address** identifies a device/interface on an IP network.

Example: 192.168.1.100

A typical IPv4 configuration contains:

Parameter	Example
IP Address	192.168.1.100
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.1
DNS Server	8.8.8.8

Private IPv4 Address Ranges

Common private networks include:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

Private IP addresses are widely used for devices inside enterprise and IoT networks.

8.3 Subnetting

Subnetting divides a larger IP network into smaller logical networks.

For example: 192.168.1.0/24

can represent a conventional IPv4 subnet with 256 total addresses and typically 254 usable host addresses.

The /24 represents the **CIDR prefix length**.

For an IoT deployment, the System Integrator should obtain the required:

- Network address
- Subnet/prefix
- Device IP range

- Default gateway

8.4 TCP and UDP

TCP and UDP operate at the **transport layer**.

TCP

- TCP is **connection-oriented** and provides reliable and ordered delivery.
- TCP establishes a connection before normal data exchange.

UDP

- UDP is **connectionless**.
- UDP does not perform a TCP-style handshake and does not provide built-in guarantees for delivery, ordering or retransmission.

Integrator rule:

TCP is normally selected when reliable, ordered delivery is required; UDP is useful when low overhead and application-controlled delivery are acceptable.

8.5 Ports

An **IP address identifies the host**, while a **port identifies a service/application endpoint** on that host.

Example: <IP-Address:Port > - 192.168.1.100:443

Common Ports

Service	Port	Transport
SSH	22	TCP
HTTP	80	TCP
HTTPS	443	TCP
MQTT	1883	TCP
MQTT over TLS	8883	TCP
DNS	53	TCP/UDP

If the gateway has Internet access but TCP 8883 is blocked by a firewall, MQTT will not connect.

Therefore, when requesting firewall access, specify:

Source → Destination → Protocol → Port → Direction → Purpose

8.6 DNS and DHCP

DNS (Domain Name System) translates a hostname into an IP address.

```
mqtt.example.com
        |
        | DNS Query
        ▼
   DNS Server
        |
        ▼
203.0.113.10
```

DHCP

DHCP (Dynamic Host Configuration Protocol) automatically provides network configuration to a device.

Simplified DHCP flow:

```
Device                    DHCP Server

   |
   | DHCP Discover
   ├──────────────────────►
   |
   | DHCP Offer
   ◄──────────────────────┤
   |
   | DHCP Request
   ├──────────────────────►
   |
   | DHCP ACK
   ◄──────────────────────┤
   |
   ▼
IP Configuration
```

DHCP can provide:

- IP address
- Subnet mask
- Default gateway
- DNS server

DHCP vs Static IP

DHCP	Static IP
Automatically assigned	Manually configured
Easy to deploy	Predictable address
Address may change	Normally remains fixed
Good for large deployments	Useful for critical equipment

The addressing method should be agreed with the customer's IT/network team before deployment.

8.7 Firewall, Routing and VPN

Routing: Routing determines how packets move between different networks.

```
IoT Gateway
192.168.1.100
      |
      ▼
Default Gateway
192.168.1.1
      |
      ▼
    Router
      |
      ▼
 Internet
      |
      ▼
Cloud / Application Server
```

If the device can communicate within its local subnet but cannot reach the cloud, check:

IP → Subnet → Gateway → Routing → Firewall → NAT

Firewall

A firewall controls which network traffic is allowed or blocked.

```
IoT Gateway
     |
     | TCP 8883
     ▼
  Firewall
     |
     | ALLOW
     ▼
 MQTT Broker
```

A firewall rule may contain:

Parameter	Example
Source	IoT Gateway subnet
Destination	MQTT Server
Protocol	TCP
Destination Port	8883
Direction	Outbound
Action	Allow
Purpose	MQTT over TLS

For security, avoid requesting unrestricted access when only a specific server and port are required.

VPN

A **VPN (Virtual Private Network)** creates a secure logical connection between networks or endpoints over an underlying network.

Typical IoT architecture:

```
IoT Site
   |
   |
   ▼
VPN Gateway
   ================
      VPN Tunnel
   ================
           |
           ▼
    Enterprise Network
           |
           ▼
    Application Server
```

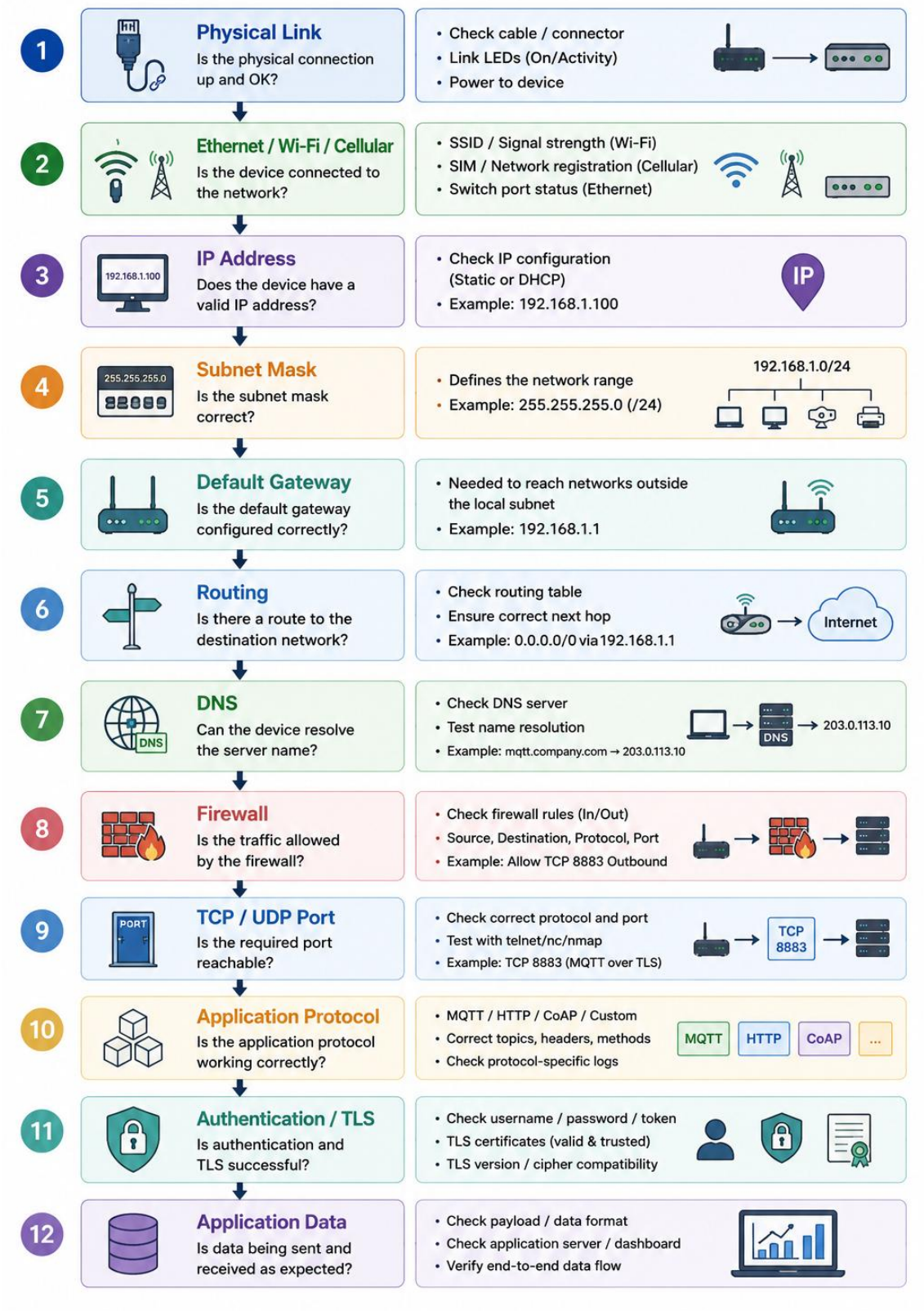
VPNs are commonly used when an IoT site needs secure access to:

- Enterprise applications
- BMS systems
- SCADA systems
- Private servers
- Databases
- Remote management systems

Common VPN technologies include **IPsec, OpenVPN and WireGuard**, depending on the organization's architecture.

8.8 Practical Network Troubleshooting

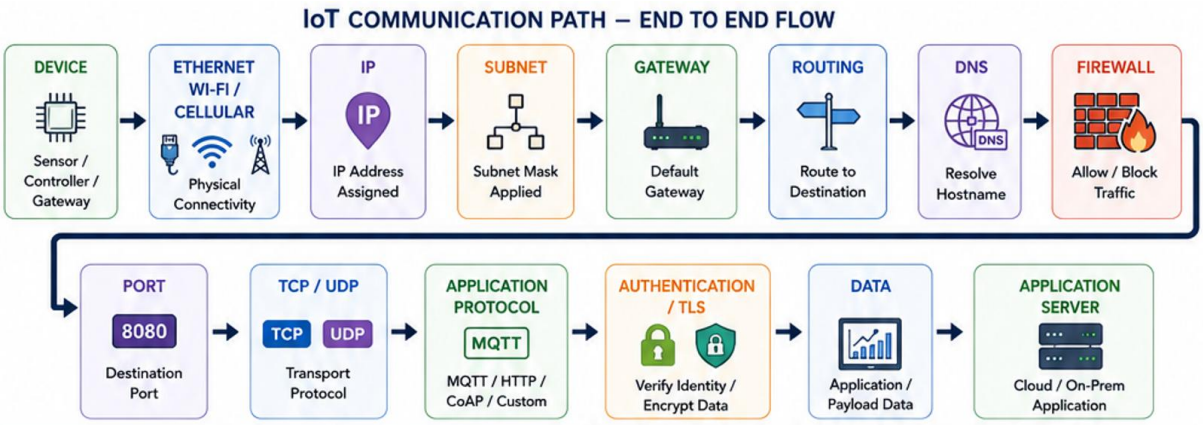
For a System Integrator, troubleshooting should be performed **layer by layer** rather than immediately assuming that the application is faulty.



Note: To identify IP addresses of devices on an authorized network, a network/IP scanning tool approved by the customer's IT or security team may be used.

8.9 System Integrator Network Checklist

PARAMETER	REQUIRED INFORMATION	
Network Type	Ethernet / Wi-Fi / Cellular	
IP Version	IPv4 / IPv6	IPv4 IPv6
IP Assignment	DHCP / Static	DHCP Static IP
IP Address	Address/range	e.g., 192.168.1.10 or 192.168.1.10 – 192.168.1.50
Subnet	CIDR/prefix	e.g., 255.255.255.0 or /24
Default Gateway	Gateway IP	e.g., 192.168.1.1
DNS	Primary/Secondary	e.g., 8.8.8.8 / 8.8.4.4
Destination	Server hostname/IP	e.g., mqtt.example.com or 203.0.113.10
Protocol	TCP / UDP	TCP UDP
Port	Destination port	e.g., 1983, 8883, 443, 5683
Firewall	Required rules	e.g., Allow Outbound TCP 8883 to 203.0.113.10
Routing	Required routes	e.g., 0.0.0.0/0 via 192.168.1.1
VPN	Required/not required	e.g., Site-to-Site VPN / Client VPN
VLAN	VLAN ID if applicable	e.g., VLAN 10, 20, 30
Authentication	Credentials/certificates	e.g., Username/Password, Client Cert, API Key
TLS	Required/not required	e.g., TLS 1.2/1.3, Certificate validation



A System Integrator should be able to trace an IoT communication path from the device all the way to the application:

Device → Ethernet/Wi-Fi/Cellular → IP → Subnet → Gateway → Routing → DNS → Firewall → Port → TCP/UDP → Application Protocol → Data

Understanding this flow allows the integrator to distinguish between a **device problem, network problem, firewall problem, server problem, or application/payload problem**

Chapter 9

System Integration Solutions

System integration in IoT is the process of connecting **field devices, sensors, controllers, communication networks, protocols, gateways, servers, and applications** so that data can move reliably from the physical device to the application layer.

For a System Integrator, the objective is not simply to establish communication. The complete solution must ensure that data is:

Collected → Transmitted → Decoded → Validated → Converted → Stored → Visualized

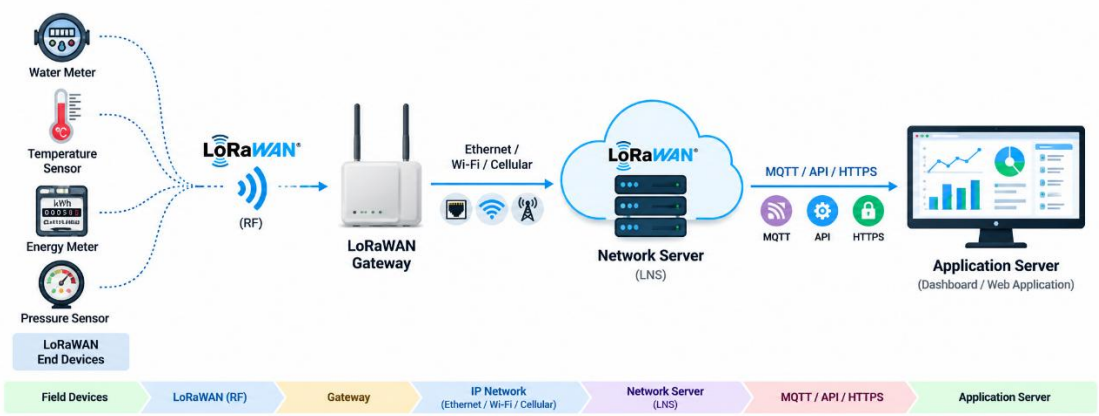
A typical integration architecture is:

Field Devices → Field Protocol → Gateway / Protocol Converter →IoT Network → Network Server / LNS → Application / Integration Layer → Database / BMS / Dashboard

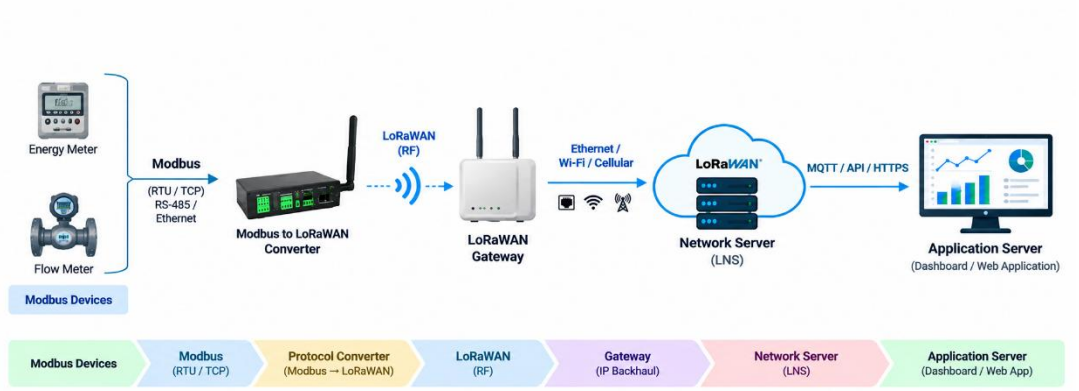
System Integration Solutions

The following sections cover various IoT solution architectures with flow diagrams, designed to help System Integrators understand the end-to-end solution and identify the appropriate devices, gateways, communication technologies, and platforms required for a particular application.

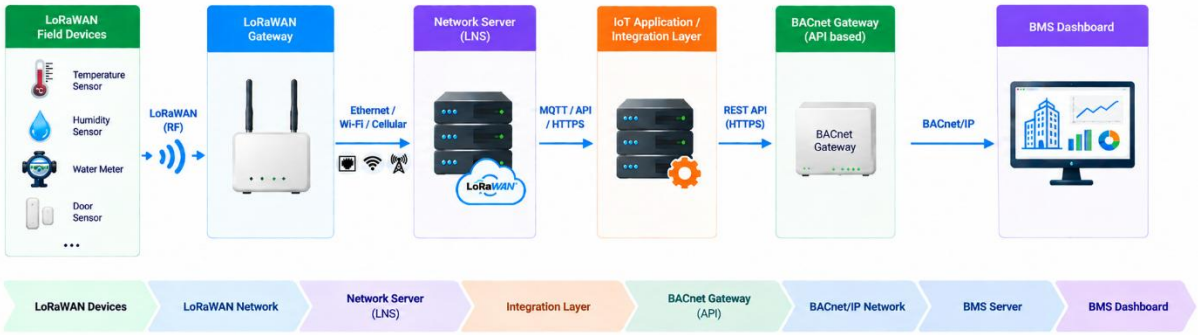
9.1 LoRaWAN Solution



9.2 Modbus to LoRaWAN Solution



9.3 LoRaWAN to BACnet Solution



9.4 Wireless M-Bus Solution

Disclaimer

This guide has been prepared for educational, informational, and technical reference purposes from an IoT System Integration perspective. It is intended to help readers understand devices, interfaces, protocols, gateways, networks, communication technologies, system architectures, and application integration concepts.

The information presented in this guide is based on a combination of technical references, industry documentation, practical system integration experience, and AI-assisted content organization and drafting. While reasonable care has been taken to present the information clearly and accurately, no guarantee is made that every technical detail is complete, current, or applicable to every device, product, network, protocol version, or implementation.

Technology standards, specifications, software, hardware, and vendor implementations may change over time. Readers should always verify implementation-specific information with the latest official standards, specifications, manufacturer documentation, product manuals, and applicable regulatory requirements.

The architectures, examples, configurations, and recommendations presented in this guide are intended to illustrate general system integration concepts. They should not be considered a substitute for professional engineering assessment, vendor-specific documentation, security review, or project-specific design and commissioning procedures.

The author and publisher assume no responsibility for any loss, damage, service interruption, configuration error, security issue, or other consequence resulting from the use or interpretation of the information contained in this guide.

Use this guide as a technical reference and starting point for understanding IoT system integration—not as a replacement for official technical documentation or project-specific engineering practices.