Technology Arts Sciences
TH Köln

M.Eng Automation and IT

PROFIBUS and MQTT Communication Protocol: Technology, Applications, and Security Analysis

Maryam Talebi – 1137469726
Mohammad Nikkhoo – 11404215

November 24, 2025

Contents

List of Figures

List of Tables

Submitted by: Maryam Talebi

1 Introduction and Protocol Overview-Profibus

PROFIBUS (Process Field Bus) was developed in 1987–1989 as a joint project between the German Federal Ministry of Research and Technology (BMFT), 13 industrial companies, and 5 research institutes. The original goal was to create a single serial fieldbus that could replace expensive, parallel 4–20 mA wiring and proprietary protocols in both factory and process automation plants. The first public demonstration took place in 1989, and the technology was officially transferred to manufacturers in 1993.

In 1999–2000, PROFIBUS became an international standard under IEC 61158 and IEC 61784 as Communication Profile Family 3 (CPF 3). This standardization guaranteed long-term availability and worldwide interoperability of devices from different vendors. Today it is maintained by PROFIBUS & PROFINET International (PI), the world's largest fieldbus user organization with more than 1,400 member companies.

PROFIBUS exists in two main variants that are optimized for different industries:

- **PROFIBUS DP** (Decentralized Peripherals) – high-speed version (up to 12 Mbit/s) using RS-485 or fiber optics, primarily used in factory automation for drives, I/O modules, and PLCs.
- **PROFIBUS PA** (Process Automation) – runs at fixed 31.25 kbit/s on Manchester Bus Powered (MBP) two-wire cable that simultaneously supplies power and transmits data. It dominates chemical, pharmaceutical, and oil & gas industries.

The following section will examine the technical details of the PROFIBUS protocol in depth.

2 Profiebus Technical Details

2.1 Reference Model and Protocol Structure

The PROFIBUS protocol is one of the few fieldbuses that was designed from the beginning according to the ISO/OSI 7-layer reference model, although it intentionally omits layers 3–6 to achieve maximum speed and determinism. This simplified structure is officially defined in IEC 61158 and IEC 61784 as Communication Profile Family 3 (CPF 3).

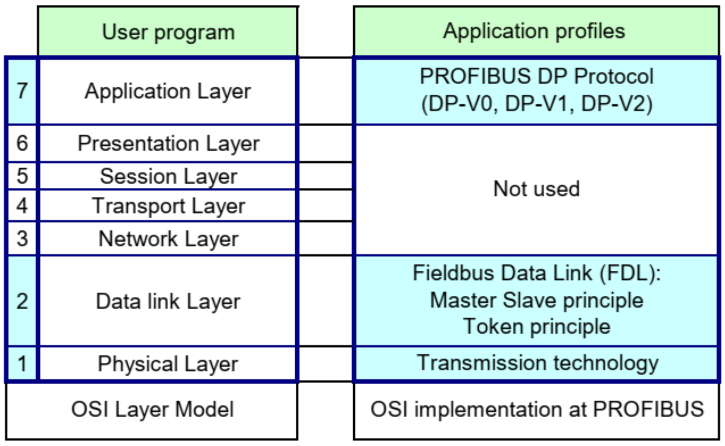
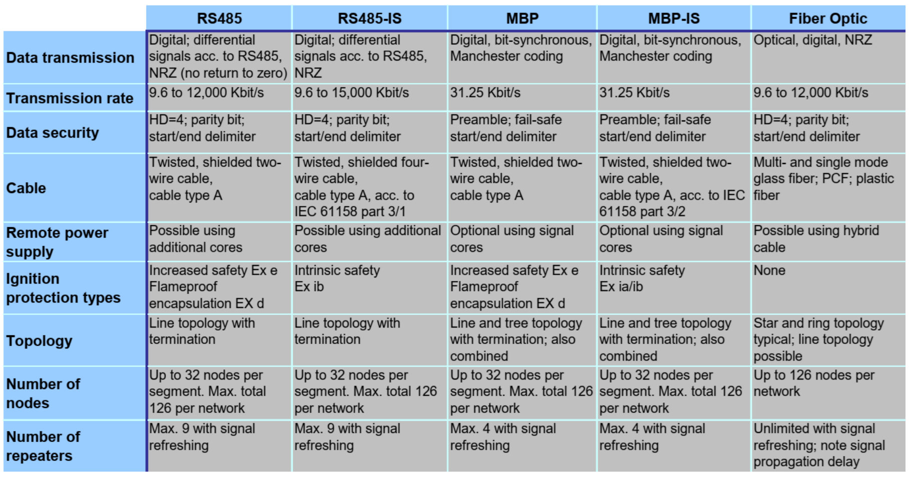


Figure 1: References between OSI model and PROFIBUS

2.2 Transmission Technology (Physical Layer)

PROFIBUS supports five standardized physical layers that can be combined in the same plant using repeaters, couplers, and optical link modules. The complete comparison is shown in Table 2.1, which is the official reference table used worldwide by planners and certification bodies.



	RS485	RS485-IS	MBP	MBP-IS	Fiber Optic
Data transmission	Digital; differential signals acc. to RS485, NRZ (no return to zero)	Digital; differential signals acc. to RS485, NRZ	Digital, bit-synchronous, Manchester coding	Digital, bit-synchronous, Manchester coding	Optical, digital, NRZ
Transmission rate	9.6 to 12,000 Kbit/s	9.6 to 15,000 Kbit/s	31.25 Kbit/s	31.25 Kbit/s	9.6 to 12,000 Kbit/s
Data security	HD=4; parity bit; start/end delimiter	HD=4; parity bit; start/end delimiter	Preamble; fail-safe start/end delimiter	Preamble; fail-safe start/end delimiter	HD=4; parity bit; start/end delimiter
Cable	Twisted, shielded two-wire cable, cable type A	Twisted, shielded four-wire cable, cable type A, acc. to IEC 61158 part 3/1	Twisted, shielded two-wire cable, cable type A	Twisted, shielded two-wire cable, cable type A, acc. to IEC 61158 part 3/2	Multi- and single mode glass fiber; PCF; plastic fiber
Remote power supply	Possible using additional cores	Possible using additional cores	Optional using signal cores	Optional using signal cores	Possible using hybrid cable
Ignition protection types	Increased safety Ex e Flameproof encapsulation EX d	Intrinsic safety Ex ib	Increased safety Ex e Flameproof encapsulation EX d	Intrinsic safety Ex ia/ib	None
Topology	Line topology with termination	Line topology with termination	Line and tree topology with termination; also combined	Line and tree topology with termination; also combined	Star and ring topology typical; line topology possible
Number of nodes	Up to 32 nodes per segment. Max. total 126 per network	Up to 32 nodes per segment. Max. total 126 per network	Up to 32 nodes per segment. Max. total 126 per network	Up to 32 nodes per segment. Max. total 126 per network	Up to 126 nodes per network
Number of repeaters	Max. 9 with signal refreshing	Max. 9 with signal refreshing	Max. 4 with signal refreshing	Max. 4 with signal refreshing	Unlimited with signal refreshing; note signal propagation delay

Figure 2: Overview of Transmission Values

The three dominant technologies in real installations are:

- **RS-485 / RS-485-IS** $\rightarrow$ used for $> 95\,\%$ of all PROFIBUS DP networks (factory automation, machine building, packaging lines). Standard RS-485 uses the well-known violet-sheathed two-wire cable (Type A). The intrinsically safe RS-485-IS variant (rarely used today) is identified by a blue cable sheath, with installations almost always employing 9-pin D-sub connectors in panels and M12 circular connectors (IP65/67) on the plant floor.

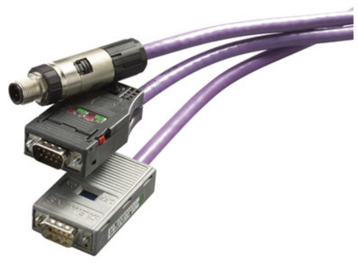


Figure 3: Profiebus DP Cable

- **MBP (Manchester Bus Powered)** $\rightarrow$ used for virtually all PROFIBUS PA installations in process industries (oil & gas, chemicals, pharmaceuticals, food & beverage) because it supplies power and data over the same pair and is intrinsically safe. MBP typically uses light blue (non-Ex) or black (preferred in Ex areas) two-wire cable, terminated with screw/cage-clamp terminals on field devices or dedicated M12 "PA" connectors.

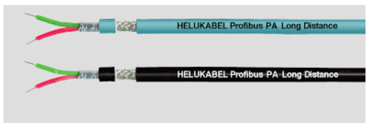

HELUKABEL Profibus PA Long Distance
HELUKABEL Profibus PA Long Distance

Figure 4: Profiebus PA Cable

- **Fiber Optic (multimode or single-mode)** → used when strong electromagnetic interference exists or distances exceed 10 km (steel mills, paper plants, railways, large chemical complexes). Typically installed with green or orange jacketed glass or POF/PCF fibers and SC, ST, or duplex-LC connectors.

Figure 5: Profiebus Fiber Optic Cable

2.3 Network Topology and Segmentation

PROFIBUS networks are always designed as one single logical bus: all devices share the same communication medium without switches or routers. The logical token-passing mechanism operates independently of the physical wiring, ensuring deterministic real-time behaviour regardless of the chosen physical layer.

PROFIBUS DP networks using RS-485 physical layer are implemented as a linear bus with daisy-chain wiring. The cable runs sequentially from device to device, and active termination resistors must be enabled at both ends of each segment to prevent signal reflections. Only very short stubs (typically less than 0.3 m at high baud rates) are allowed unless active repeaters or special bus connectors are employed. One RS-485 segment supports up to 32 nodes, with maximum segment length depending on baud rate (e.g., 1200 m at 93.75 kbit/s down to 100 m at 12 Mbit/s).

PROFIBUS PA operates at a fixed 31.25 kbit/s using Manchester Bus Powered (MBP) transmission. Thanks to the lower speed and robust Manchester encoding, reflections are far less critical. Both linear bus and tree topologies are therefore fully supported and widely used. Field devices are commonly connected via junction boxes, field barriers, or multi-port spurs, resulting in practical tree or star-like structures while remaining part of the same logical bus.

Fibre-optic links are employed when electromagnetic immunity, electrical isolation, or distances beyond several kilometres are required. Optical Link Modules (OLMs) convert the electrical signal to light and back again. Fibre segments can be configured in linear, star, or — using OLMs with redundancy function — redundant ring topologies. In a redundant optical ring, a cable break or module failure is automatically bypassed by routing communication in the opposite direction, providing extremely high network availability without affecting token passing.

Through the intelligent combination of repeaters, DP/PA couplers, and optical link modules, large industrial plants can contain several hundred devices distributed across multiple physical segments while still functioning as one deterministic logical PROFIBUS network.

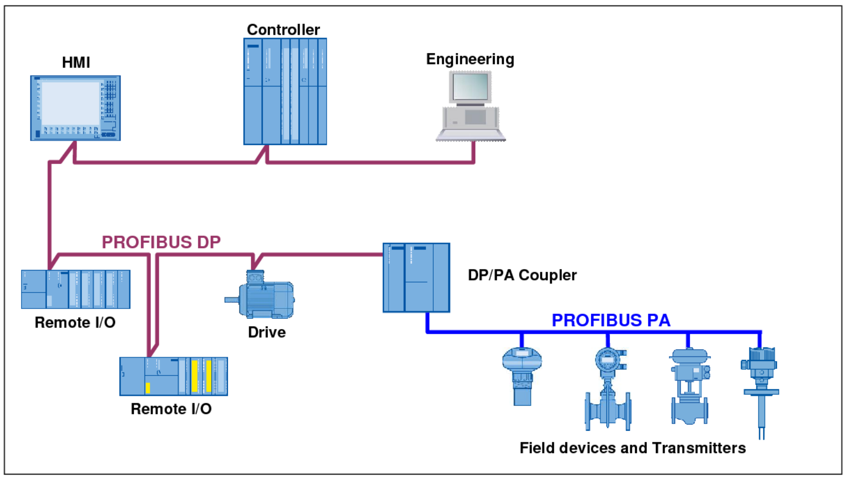


Figure 6: Profiebus Topology Sample

2.4 Data Link Layer (FDL)

The PROFIBUS data link layer – officially called **Fieldbus Data Link (FDL)** – acts as the "traffic policeman" of the bus. It ensures that all devices communicate in perfect order and that no two devices ever speak at the same time, even when the bus is 100 % loaded.

PROFIBUS devices are divided into three official classes based on their functions:

- **Class-1 Master** – the main controller (almost always a PLC or DCS) that runs the process or machine in real time.
- **Class-2 Master** – engineering tools, laptops, HMIs, or diagnostic devices that are connected only temporarily for configuration, monitoring, or maintenance.
- **Slave** – sensors, actuators, drives, valves, remote I/O – devices that never start communication on their own.

Communication follows two simple rules that work together:

- **Master–slave principle**: Only masters (Class-1 or Class-2) may initiate communication. Slaves remain silent and only reply when directly addressed – like students who only speak when the teacher asks them a question.
- **Token passing among masters**: All masters (both Class-1 and Class-2) form a logical ring and pass a virtual "spean right" called the **token**. Only the master currently holding the token is allowed to poll its slaves or perform acyclic services. As soon as it finishes, it immediately hands the token to the next master in the ring.

This hybrid mechanism (master–slave + token ring between masters) delivers two decisive advantages:

1. **Deterministic timing** – the bus cycle time is always exactly predictable (e.g., 2 ms, 5 ms, or 10 ms) regardless of load.
2. **Zero collisions** – unlike early Ethernet, data packets can never crash into each other because only one station transmits at any moment.

The token circulation and polling sequences are fully automatic and do not require manual settings. This extremely robust and reliable real-time behavior is the reason PROFIBUS has been used successfully in industrial plants for more than three decades.

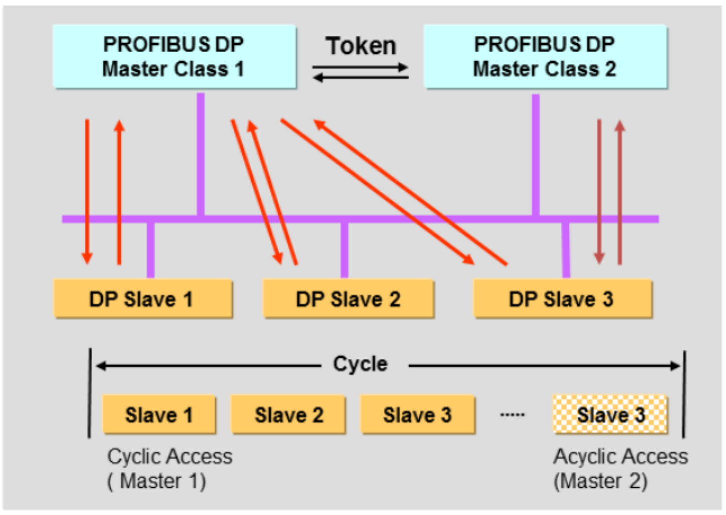

PROFIBUS DP Master Class 1
Token
PROFIBUS DP Master Class 2
DP Slave 1
DP Slave 2
DP Slave 3
Cycle
Slave 1
Slave 2
Slave 3
.....
Slave 3
Cyclic Access (Master 1)
Acyclic Access (Master 2)

Figure 7: Token passing between Class-1 and Class-2 masters

2.5 Cyclic and Acyclic Communication Services

PROFIBUS supports two fundamentally different types of communication services – cyclic and acyclic – which serve different purposes and follow a strict priority scheme.

Cyclic communication is the high-priority, real-time transfer of process values (e.g., temperature, pressure, motor speed, valve position, digital inputs/outputs). It occurs automatically in every single bus cycle – usually every 1–10 ms – using only a few fixed bytes per device. Because cyclic data are exchanged continuously and with absolute priority, the control loop is never delayed, even when the bus is loaded.

Acyclic communication, in contrast, is used only occasionally for reading or writing device parameters, diagnostic information, alarm messages, recipes, or configuration data. Acyclic services are executed only when spare time is left after all cyclic traffic has been completed in the current token cycle. This clear priority rule – **cyclic always before acyclic** – guarantees that real-time control is never disturbed by engineering or maintenance activities.

2.6 Device Integration and GSD Files

The key to PROFIBUS being truly open and multi-vendor is the **General Station Description (GSD)** file – a small, standardised, human-readable text file that every manufacturer must deliver with each PROFIBUS device. The GSD tells the engineering software everything needed for real-time (cyclic) communication: how many input and output bytes the device uses, which baud rates it supports, its diagnostic capabilities, timing parameters, and its unique identification numbers. When an engineer adds the device in tools such as TIA Portal, PCS 7, or ABB 800xA, the software simply reads the GSD and automatically configures the PLC – no special drivers or manual programming are required. This is why devices from completely different suppliers (e.g., Endress+Hauser transmitters, Samson valves, SEW drives) work together immediately on the same bus.

Although the GSD is sufficient for cyclic data exchange and basic setup, more complex devices have hundreds of additional parameters, menus, and graphics that cannot fit into a simple GSD. For these advanced functions, three complementary technologies are available and fully supported on PROFIBUS systems:

- **EDD** (Electronic Device Description) – text-based, standardised description language,
- **FDT/DTM** – software component with manufacturer-specific graphical interfaces,
- **FDI** (Field Device Integration, IEC 62769) – the newest unified solution combining the best of EDD and FDT/DTM using OPC UA.

Regardless of which advanced technology is used, the GSD file remains the mandatory foundation that allows every PROFIBUS device to join the cyclic real-time communication.

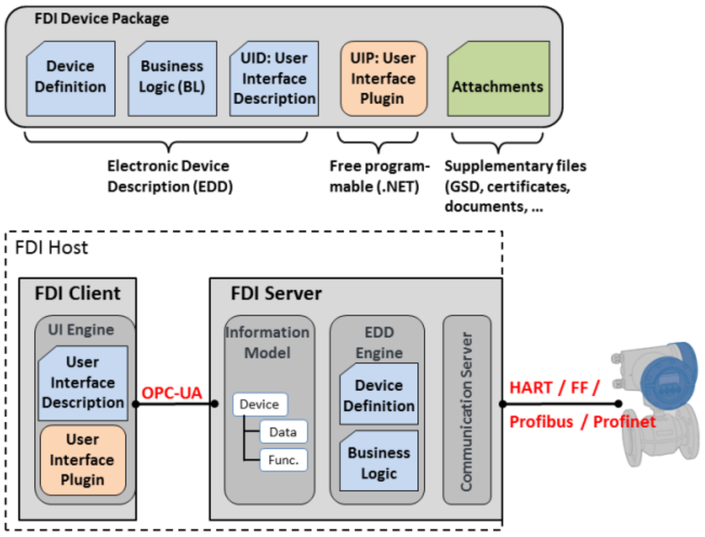


Figure 8: FDI Device Package and FDI host

2.7 PROFIBUS DP Versions and Extensions

PROFIBUS DP evolved in three main steps:

- **DP-V0** (1990s) – basic, fast cyclic data exchange and simple diagnostics.
- **DP-V1** ($\approx$ 2000) – added acyclic parameter reading/writing and alarms, essential for process automation (PA).
- **DP-V2** (later) – introduced high-speed drive control with **isochronous mode** (all devices synchronised within microseconds), publisher-subscriber communication, and support for the two most important extensions:
  - **PROFIdrive** – universal drive profile (speed, position, torque control of any motor brand),
  - **PROFIsafe** – safety-related communication (emergency stop, light curtains) over the same standard cable, certified up to SIL 3.

All versions are fully backward compatible – a modern PLC can still communicate without problems with 30-year-old DP-V0 slaves.

3 Security Analysis of PROFIBUS

PROFIBUS deliberately implements only OSI Layers 1, 2, and 7, omitting Layers 3–6 completely.This design was chosen in the 1990s to achieve maximum speed and low cost, but it removes every layer where modern networks normally add security.

At the application layer (Layer 7), PROFIBUS DP-V0/V1/V2 and all standard profiles send process data, setpoints, and alarms in clear text without any authentication or digital signatures. At the data-link layer (Layer 2), the only integrity protection is an 8-bit non-cryptographic Frame Check Sequence (FCS) that detects accidental bit errors but can be perfectly recalculated by an attacker in microseconds after any deliberate modification of addresses, function codes, or payload. Layers 3 to 6 are entirely absent, so there are no IP-style addresses, no sequence numbers, no replay protection, and no possibility of TLS-like encryption or session keys. Finally, the physical layer (Layer 1) uses open multi-drop RS-485 wiring, allowing any device to be connected in parallel without causing detectable errors.

OSI Layer	What is missing in PROFIBUS	Direct security consequence
Layer 7 (Application)	No authentication, no encryption, no digital signatures in DP-V0/V1/V2	All process values, setpoints, and alarms are sent in clear text inside the telegram Data field.
Layers 3–6	Completely absent (Network, Transport, Session, Presentation)	No IP-like addressing, no sequence numbers, no replay protection, no TLS-like encryption possible.
Layer 2 (FDL)	Only an 8-bit non-cryptographic checksum (FCS)	An attacker who changes any byte (address, function code, or data) can instantly recalculate the correct FCS – tampering is undetectable.
Layer 1	Open multi-drop RS-485	Anyone who reaches the cable can read or inject perfect telegrams.

Table 1: Security weaknesses of PROFIBUS across OSI layers

The **PROFIBUS telegram** directly mirrors the absence of security in the implemented OSI layers. It contains only a start delimiter, 1-byte source and destination addresses, a function code, the clear-text data payload, an 8-bit non-cryptographic Frame Check Sequence (FCS), and an end delimiter.

Byte	Field	Fixed/Variable	Description
1	SD	Fixed (68h)	Start Delimiter
2	LE	Variable	Length of telegram section
3	LEr	Variable (same as LE)	Length repeat
4	SD	Fixed (68h)	Start Delimiter repeat
5	DA	Variable (00–7Eh)	Destination Address
6	SA	Variable	Source Address
7	FC	Variable	Function Code
8–n	Data	Variable (0–244 bytes)	Cyclic/acyclic payload
n+1	FCS	Variable	8-bit checksum (calculated)
n+2	ED	Fixed (16h)	End Delimiter

Table 2: PROFIBUS Telegram Structure

Since no cryptographic binding, authentication, or replay protection exists **at any OSI layer**, an attacker who can transmit on the bus (physically or through a compromised gateway) can instantly:

- read all process data,
- forge or modify any command,
- replay previously captured telegrams,
- impersonate the legitimate master or any slave,

with **100 % success and no detection** by the protocol itself.

3.1 Real-World Exploitation: The Stuxnet Attack on PROFIBUS-Controlled Centrifuges

The Stuxnet worm, discovered in 2010, remains the most sophisticated and best-documented cyber-physical attack that directly exploited the absence of security mechanisms in PROFIBUS DP.

Stuxnet targeted the uranium enrichment facility at Natanz, Iran, where cascades of high-speed gas centrifuges were controlled by Siemens S7-315 and S7-417 PLCs connected via PROFIBUS DP to frequency converters (Valmet/Vacon and KSB models). After infecting Windows-based engineering stations through USB drives and four zero-day vulnerabilities, Stuxnet injected malicious code blocks into the PLCs using the proprietary S7comm protocol. Once resident in the PLC firmware, it performed the following actions entirely over the PROFIBUS DP network:

- Monitored clear-text process values (rotor frequency) received from the frequency converters.
- Periodically overwrote the speed setpoint telegrams transmitted to the converters (e.g., commanding 1 410 Hz followed by 2 Hz) to induce destructive mechanical resonance.
- Simultaneously replayed previously recorded “normal” frequency values ($\approx 1\,070$ Hz) back to the SCADA system, concealing the sabotage from operators.

These manipulations were possible because PROFIBUS DP telegrams contain no sender authentication, no cryptographic integrity protection, and no replay prevention. The 8-bit Frame Check Sequence (FCS) could be correctly recalculated after any modification, making forged or altered telegrams indistinguishable from legitimate ones.

The attack ultimately destroyed approximately 1 000 IR-1 centrifuges over several months while remaining undetected by the control system, demonstrating that the lack of message authentication and confidentiality in PROFIBUS enables complete compromise once an attacker gains code execution on a legitimate master device.

3.2 Two Practical Approaches for Securing PROFIBUS Communication

3.2.1 Approach 1: PROFIBUS with IPSec over Ethernet via Gateway

In the first approach, master and field devices continue to communicate using unchanged PROFIBUS DP/PA protocol on the factory floor, ensuring maximum real-time performance. A gateway or proxy device translates PROFIBUS telegrams into standard Ethernet/IP packets. From the gateway to the control room, SCADA system, or engineering workstation, the connection is protected by IPSec (the same protocol commonly used for VPNs).

Advantages

- Reuses mature, widely available Internet security technology (IPSec).
- Requires no modification of existing PROFIBUS devices or cabling.
- Provides confidentiality, integrity, and authentication outside the real-time segment.

Disadvantages

- Encryption and decryption introduce noticeable processing and latency overhead.
- Key management (Security Associations) becomes complex, potentially requiring n^2 keys for n nodes.
- Early implementations suggested weak algorithms (e.g., 56-bit DES) to reduce delay, which are no longer considered secure.
- Unsuitable for strict real-time requirements directly at the field level.

3.2.2 Approach 2: PROFIBUS with OPC UA via Gateway (Recommended Modern Solution)

The field-level communication remains pure PROFIBUS DP/PA, preserving the required cycle times of less than 10 ms. A modern gateway (typically an FDI Server or OPC UA wrapper) converts the PROFIBUS data into standardized OPC UA messages. From the gateway upward to SCADA, MES, engineering stations, or cloud systems, only the secure OPC UA protocol is used, which natively supports strong authentication, encryption, and certificate management.

Advantages

- Official industry-standard solution for secure interoperability (OPC UA).
- Employs up-to-date cryptography (AES-128/AES-256, SHA-256, proper certificates) without relying on outdated algorithms.
- Centralized user management, session security, and comprehensive auditing.
- Seamless integration with higher-level systems and Industry 4.0/IIoT environments.
- Leaves the real-time PROFIBUS segment completely untouched and fast.

Disadvantages

- Strong OPC UA security policies add overhead; they are therefore applied only above the field level (real-time segment remains unencrypted).
- Requires a capable gateway or device that supports OPC UA or FDI.
- Slightly higher initial implementation and configuration effort compared to simple Ethernet tunneling.

In current industrial practice, Approach 2 (OPC UA layered above the fieldbus) is the recommended and most widely adopted method, as it delivers robust security without compromising the critical real-time performance of PROFIBUS networks.

Submitted by: Mohammad Nikkhoo

4 Introduction and Protocol Overview — MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight publish–subscribe communication protocol originally designed for machine-to-machine (M2M) telemetry applications in constrained environments. The protocol was created in 1999 as a joint effort between Andy Stanford-Clark of IBM and Arlen Nipper of Arcom (later part of Eurotech). Their design goal was to develop an extremely bandwidth-efficient communication method suitable for high-latency or expensive satellite links, such as those used in remote oil and gas pipeline monitoring.

The protocol gained rapid adoption due to its simplicity, minimal header overhead, and suitability for battery-powered or low-resource embedded devices. In 2010, IBM released MQTT v3.1 to the public, enabling broader community adoption and implementation across vendors. MQTT subsequently underwent formal standardization through the Organization for the Advancement of Structured Information Standards (OASIS). In 2014, MQTT v3.1.1 was officially approved as an OASIS standard, providing clearer behavioral definitions and improved interoperability for industrial and IoT applications.

A major milestone occurred in March 2019, when MQTT v5.0 became an OASIS Standard. This revision introduced substantial enhancements—including user properties, improved error reporting, shared subscriptions, and request–response capabilities—making the protocol more scalable and enterprise-ready while maintaining backward compatibility with existing deployments.

MQTT operates on top of TCP/IP and is built around three core elements: publishers, subscribers, and a central broker. Clients publish messages to hierarchical topics, and subscribers receive messages for the topics they are interested in. The broker handles session management, message routing, quality of service (QoS) guarantees, and optional retained messages. This architecture enables highly decoupled communication, as publishers and subscribers do not need to know anything about each other's identities or availability.

One of MQTT's defining characteristics is its focus on lightweight communication. The protocol control packets are intentionally simple, allowing implementation even on microcontrollers with limited memory. Its design supports three QoS levels (0, 1, and 2), enabling applications to choose between latency, reliability, and network load. This efficiency makes MQTT the de-facto messaging protocol in modern IoT systems, ranging from consumer smart devices to industrial SCADA, process automation, and remote monitoring infrastructures.

The following sections will examine the technical details of the MQTT protocol in depth.

5 MQTT Technical Details

5.1 Protocol Architecture and Position in the OSI Model

MQTT is an application-layer messaging protocol built on top of the TCP/IP stack. It does not follow the full OSI 7-layer model as used in classical fieldbus systems; instead, it defines only application-layer semantics while relying entirely on TCP for transport-layer reliability, ordering, and retransmission.

The protocol is formally standardized in two major specifications:

- **ISO/IEC 20922:2016** — the international MQTT standard.

- **OASIS MQTT Specifications** — including versions 3.1.1 (2014) and 5.0 (2019), which define protocol behavior, control packets, session management, and extension mechanisms.

MQTT organizes functionality into three conceptual layers rather than adopting the full OSI stack:

1. **Transport Layer (TCP)**: Provides connection-oriented and reliable data transfer.
2. **Session Behavior**: Controlled through the CONNECT handshake, keep-alive timers, clean session flags, persistent sessions, and (in MQTT 5.0) session expiry settings.
3. **Application Messaging Semantics**: Based on hierarchical topic names, publish–subscribe routing, payload flexibility, retained messages, and configurable Quality of Service (QoS) levels.

Because reliability mechanisms are delegated to TCP, MQTT control packets are intentionally compact (typically requiring only a 2-byte fixed header). This design makes the protocol suitable for constrained devices, high-latency networks, and large-scale IoT deployments while maintaining interoperability through its standardized application-layer structure.

5.2 Communication Model (Publish–Subscribe & Broker Functions)

MQTT implements a topic-based publish–subscribe communication model, which provides loose coupling between data producers (publishers) and data consumers (subscribers). Unlike traditional request–response or bus-oriented architectures, MQTT clients do not communicate directly with each other. Instead, all interactions are mediated by a central entity known as the **broker**.

Publish–Subscribe Paradigm

In the MQTT model, publishers send application messages to hierarchical *topics*, which act as logical addresses rather than device identifiers. Subscribers register interest in specific topics or topic patterns using wildcards, allowing flexible message filtering and dynamic data distribution.

This decoupling occurs in three dimensions:

- **Time decoupling**: Publishers and subscribers do not need to be online simultaneously.
- **Space decoupling**: Endpoints do not require knowledge of each other's IP address or identity.
- **Synchronization decoupling**: Communication is asynchronous; subscribers receive updates automatically.

Role of the Broker

The broker is responsible for the core routing and management functionality of the MQTT ecosystem:

- **Message routing** based on topic subscriptions.

- **Session management**, including persistent sessions and stored subscriptions.
- **Quality of Service (QoS)** handling and retransmission.
- **Retained messages** to deliver the last known value to newly connected clients.
- **Will messages**, allowing clients to define last-will notifications on unexpected disconnections.
- **Subscription filtering**, including hierarchical matching and wildcard expansion.

MQTT 5.0 further extends broker functionality with shared subscriptions for load balancing, negative acknowledgments, and enhanced metadata propagation through user properties.

Advantages in IoT/IIoT Deployments

The publish–subscribe model significantly improves scalability and robustness for distributed systems. It enables efficient many-to-many communication patterns, which are particularly valuable in industrial IoT (IIoT), distributed sensing, and remote monitoring applications where device availability and network conditions vary over time.

5.3 Transport Binding (TCP, WebSocket, MQTT-SN)

MQTT relies on a simple and well-defined transport binding, with the primary objective of providing reliable delivery while minimizing protocol overhead. Unlike traditional industrial fieldbuses, MQTT does not define any physical or data link layers; instead, it inherits all reliability, framing, ordering, and retransmission behavior from its underlying transport protocols.

TCP Transport (Default Binding)

The standard and mandatory transport binding for MQTT is **TCP**. A typical MQTT connection uses:

- Port **1883** for unencrypted transport.
- Port **8883** for MQTT over TLS (MQTTS).

TCP ensures:

- reliable, in-order delivery,
- retransmission on packet loss,
- flow control and congestion control,
- persistent, stateful connections.

These characteristics allow MQTT to keep its control packets extremely small and simple, as the protocol does not need to implement its own reliability mechanisms beyond QoS handshakes.

WebSocket Transport

MQTT may also operate over **WebSockets**, typically in environments where direct TCP connections are restricted or when browser-based clients must participate in MQTT messaging. In such cases, messages are encapsulated in WebSocket frames, while the MQTT packet structure remains unchanged.

WebSocket-based MQTT is widely used in:

- browser dashboards and HMIs,
- cloud IoT platforms,
- secure tunneling through proxies and load balancers.

MQTT-SN for Wireless Sensor Networks

MQTT for Sensor Networks (MQTT-SN) is a companion protocol designed for extremely low-power or non-TCP wireless networks such as IEEE 802.15.4 or UDP-based mesh networks. MQTT-SN does not require TCP; instead, it introduces:

- short **topic IDs** instead of UTF-8 strings,
- a **gateway** that bridges the sensor network to a standard MQTT broker,
- optional broadcast-based discovery mechanisms.

These optimizations make MQTT-SN suitable for battery-operated devices and high-density sensor deployments where TCP overhead is not acceptable.

So, MQTT intentionally avoids defining lower-layer communication behavior. By running on top of TCP, WebSockets, or lightweight wireless transports (via MQTT-SN), the protocol can operate across heterogeneous networks while maintaining the same standardized application-layer behaviors and semantics.

5.4 Message Structure and Control Packet Format

MQTT defines a compact and efficient control packet structure that minimizes protocol overhead and enables implementation on constrained embedded devices. All MQTT interactions between a client and a broker consist of well-defined **control packets**. These packets encapsulate connection setup, message publication, subscription management, keep-alive mechanisms, and graceful disconnection.

Control Packet Types

The MQTT specification defines a fixed set of packet types, including:

- **CONNECT / CONNACK** — establish a session with the broker.
- **PUBLISH** — transmit an application message on a topic.
- **PUBACK, PUBREC, PUBREL, PUBCOMP** — QoS 1 and QoS 2 delivery flows.
- **SUBSCRIBE / SUBACK** — register topic filters.
- **UNSUBSCRIBE / UNSUBACK** — remove subscriptions.

- **PINGREQ / PINGRESP** — keep-alive and connection liveness checks.
- **DISCONNECT** — terminate the session.

In MQTT 5.0, several packet types include extended metadata through *reason codes* and *user properties*, enabling richer communication between clients and brokers.

Control Packet Structure

Each MQTT control packet consists of up to three components:

1. **Fixed Header** Every control packet includes a fixed header of at least 2 bytes. It contains the packet type, flags, and the remaining length field (encoded using a variable-length integer representation).
2. **Variable Header** Present in many packet types. Examples:
    - CONNECT: protocol name, protocol level, connect flags, keep-alive timer.
    - PUBLISH: topic name and packet identifier (for QoS1/QoS2).
    - SUBSCRIBE: packet identifier and topic filter list.
3. **Payload** Appears only for certain packets. Examples:
    - CONNECT: client identifier, will message parameters, username and password.
    - PUBLISH: application message payload.
    - SUBSCRIBE: list of topic filters and requested QoS levels.

Packet Flow for Reliable Delivery

MQTT uses different control packet sequences for each Quality of Service level:

- **QoS 0**: PUBLISH only (no acknowledgments).
- **QoS 1**: PUBLISH $\rightarrow$ PUBACK.
- **QoS 2**: PUBLISH $\rightarrow$ PUBREC $\rightarrow$ PUBREL $\rightarrow$ PUBCOMP.

These flows ensure reliable or exactly-once delivery semantics while keeping the encoding minimal. The separation of QoS mechanisms from message payloads enables lightweight and consistent behavior across diverse applications.

Design Considerations

The compact packet structure is one of MQTT's defining characteristics. It reduces bandwidth consumption and processing overhead, making the protocol highly suitable for:

- embedded microcontrollers,
- high-latency satellite links,
- cellular IoT networks,
- large-scale cloud messaging infrastructures.

Because MQTT delegates framing, retransmission, and ordering to the transport layer (TCP), the control packet format remains simple, deterministic, and easy to parse even for resource-constrained devices.

5.5 Quality of Service, Sessions, and Message Reliability

MQTT provides several mechanisms to ensure predictable message delivery in distributed systems with varying network reliability. These mechanisms span Quality of Service (QoS) levels, session behavior, and explicit message lifecycle controls. The combination of these features enables MQTT to adapt to resource-constrained devices as well as large-scale enterprise deployments.

Quality of Service Levels

MQTT defines three QoS levels, each representing a different reliability guarantee:

- **QoS 0 – At most once** The sender transmits the message without expecting acknowledgment. Delivery is not guaranteed, making this level suitable for non-critical sensor updates or highly latency-sensitive applications.
- **QoS 1 – At least once** The sender retries until it receives a `PUBACK`. Duplicate messages are possible, and application logic must be able to handle them.
- **QoS 2 – Exactly once** Ensures that each message is received only once through a four-step handshake involving `PUBREC`, `PUBREL`, and `PUBCOMP`. This level provides the strongest guarantee and is often used for financial or command operations where duplicates cannot be tolerated.

The QoS model is asymmetric: publishers and subscribers may each request different levels, and the broker enforces the most restrictive QoS applicable to the message path.

Session Behavior

Sessions determine how state is maintained between clients and brokers. MQTT supports two primary session modes:

- **Clean Session (Stateless)** No session state is stored after disconnection. Subscriptions and pending messages are discarded, allowing lightweight and transient connections.
- **Persistent Session (Stateful)** The broker stores subscriptions, unacknowledged QoS messages, and queued retained messages. Clients reconnect without losing state, providing stronger guarantees for intermittent or mobile devices.

MQTT 5.0 introduces **session expiry intervals**, enabling fine-grained control over how long session data remains stored after disconnection.

Keep-Alive and Liveness Detection

MQTT uses an application-level keep-alive timer. If no control packets are exchanged within the configured interval, the client sends a `PINGREQ`, to which the broker responds with `PINGRESP`. Failure to receive a response causes both sides to assume the session is no longer valid, triggering disconnection and optional Last Will publication.

Last Will and Testament

The **Last Will** mechanism allows a client to specify a message that the broker should publish on its behalf if the client disconnects unexpectedly. This is particularly useful in:

- machine health monitoring,
- device presence detection,
- supervisory control applications.

Message Expiry and Flow Control (MQTT 5.0)

MQTT 5.0 introduces additional features to enhance reliability under heavy load:

- **Message Expiry Interval**: discards stale messages automatically.
- **Receive Maximum**: limits the number of in-flight QoS 1 and QoS 2 messages.
- **Topic Alias**: optimizes repeated topic names to reduce bandwidth.

These enhancements provide greater flexibility for industrial and cloud-based MQTT systems that require strict message lifecycle management.

So, MQTT's reliability model combines transport-layer guarantees (TCP) with protocol-level QoS flows, persistent sessions, and message lifecycle controls. This layered design allows the protocol to function efficiently across diverse network environments ranging from resource-limited embedded devices to large-scale IIoT infrastructures.

5.6 Topic Namespace, Payload, and Metadata Conventions

MQTT uses a flexible and hierarchical topic namespace to route messages between clients. Unlike traditional industrial fieldbus systems, MQTT does not prescribe fixed data models, device profiles, or communication slots. Instead, it provides a general-purpose messaging framework upon which application-specific conventions can be built. This flexibility enables MQTT to support a wide range of IoT and IIoT use cases, from simple telemetry to structured industrial data exchange.

Topic Namespace

Topics in MQTT are UTF-8 encoded hierarchical strings separated by forward slashes. They act as logical routing paths rather than device addresses. Examples include:

- `factory/line1/robot3/position`
- `building/room42/temperature`
- `device/1234/status`

Two wildcard mechanisms support flexible subscription patterns:

- **Single-level wildcard** (+): matches one level in the hierarchy. Example: `factory/+/robot3/status`
- **Multi-level wildcard** (#): matches multiple remaining levels. Example: `factory/#`

The namespace is application-defined, offering high adaptability but also requiring careful design in industrial systems to ensure consistency and maintainability.

Payload Encoding

MQTT is **payload-agnostic**: the broker treats the payload as an opaque binary blob. Common encoding formats include:

- JSON or CBOR for structured sensor data,
- binary formats for optimized embedded communication,
- OPC UA mappings for industrial interoperability,
- Protobuf or Avro for cloud-oriented deployments.

Because the protocol imposes no schema, interoperability in industrial contexts depends on external conventions or standards.

Retained Messages

MQTT supports **retained messages**, which store the last published value of a topic on the broker. New subscribers immediately receive this retained value, enabling state synchronization for:

- dashboards and HMIs,
- device status monitoring,
- system initialization after reboot.

Retained messages are especially valuable in IIoT environments where many devices periodically disconnect and reconnect.

Metadata and Industrial Conventions

Since MQTT itself does not specify metadata models, several higher-level frameworks introduce structured conventions:

- **Sparkplug B**: defines a standardized topic namespace, typed payloads, device birth/death certificates and edge-node modeling for industrial MQTT deployments.
- **Digital Twins / Device Shadows**: used by cloud platforms such as AWS IoT and Azure IoT Hub to maintain virtual device states using JSON schemas.
- **Custom Metadata Topics**: many systems publish parameters and diagnostics under reserved paths such as `device/<id>/meta`.

These conventions provide interoperability and lifecycle management without increasing the complexity of the underlying MQTT protocol.

So, MQTT's topic and payload model is intentionally open and lightweight. The protocol offers powerful routing through hierarchical topics and wildcards while leaving semantic modeling to application layers or industrial frameworks such as Sparkplug B. This design enables MQTT to scale from simple telemetry to complex multi-vendor IIoT systems.

5.7 MQTT Versions and Extensions

MQTT protocol has evolved over time to accommodate the growing demands of Internet of Things (IoT) applications and other lightweight messaging systems. The primary versions of MQTT include:

- **MQTT 3.1:** The initial widely adopted version, designed to enable lightweight publish/subscribe communication for constrained devices and low-bandwidth networks.
- **MQTT 3.1.1:** An enhancement of MQTT 3.1, standardized by OASIS in 2014, which introduced improvements in protocol clarity, connection handling, and error reporting.
- **MQTT 5.0:** The latest major version, standardized in 2019, offering enhanced features such as reason codes, user properties, improved error handling, message expiry intervals, and shared subscriptions. MQTT 5.0 is designed to support more complex IoT deployments and scalable messaging architectures.

In addition to the core versions, MQTT supports various extensions and enhancements to improve interoperability and functionality:

- **MQTT-SN (MQTT for Sensor Networks):** A variant optimized for sensor networks, allowing communication over non-TCP/IP protocols such as Zigbee or LoRaWAN.
- **Security Extensions:** Integration with TLS/SSL for encrypted communication, along with support for authentication mechanisms including username/password, token-based, and certificate-based authentication.
- **Quality of Service Extensions:** Beyond the standard QoS levels (0, 1, 2), extensions may include application-specific delivery guarantees and message persistence mechanisms.
- **Bridging and Clustering:** Enhancements for connecting multiple MQTT brokers, enabling load balancing, fault tolerance, and distributed messaging architectures.

These versions and extensions ensure that MQTT remains a versatile and robust protocol, capable of addressing the evolving requirements of IoT systems, edge computing, and other lightweight messaging scenarios.

6 Security Analysis of MQTT (Protocol-Level Assessment)

MQTT was not originally designed as a security-focused protocol. As stated in the official OASIS standards (MQTT 3.1.1 and MQTT 5.0), the protocol defines only application-layer messaging semantics and delegates all confidentiality, integrity, and authentication mechanisms to the underlying transport. Therefore, MQTT provides minimal built-in protection unless implementers explicitly enable secure configurations.

6.1 Lack of Native Security Mechanisms

MQTT does not include cryptographic services in its core specification. Specifically:

- **No mandatory encryption** — MQTT over TCP (default port 1883) transmits all control packets, metadata, and payloads in cleartext.
- **No built-in message integrity** — MQTT contains no digital signatures, MACs, or integrity codes. Any bit-level modification remains undetected unless TLS is used.
- **No native replay protection** — MQTT QoS flows ensure delivery semantics, not cryptographic freshness. Attackers can replay previously captured PUBLISH packets if TLS or application-level counters are not used.

Thus, MQTT is inherently "secure-by-configuration," not secure by itself.

6.2 Vulnerabilities in Session Establishment (CONNECT/CONNACK)

The CONNECT packet is particularly sensitive:

- Username and password are sent in plain text when TLS is disabled.
- ClientID spoofing allows an attacker to disconnect or impersonate legitimate clients.
- Last Will misuse enables attackers to generate false device-offline alarms.

MQTT 5.0 introduces enhanced reason codes and authentication exchange fields (AUTH packet), but these still rely entirely on a secure transport layer.

6.3 Topic Namespace Vulnerabilities

Since MQTT uses hierarchical topic strings without access control by default:

- An attacker who connects to a misconfigured broker can subscribe using the # wildcard and read all system data.
- Attackers can publish commands to control actuators, alarms, or configuration topics in industrial and IIoT deployments.
- Retained messages can be overwritten to store malicious states (e.g., false sensor readings).

MQTT brokers must implement ACLs, as described in Mosquitto documentation.

6.4 Broker-Level Risks

Broker compromise is critical because the broker is the message router for the entire system. Common broker risks include:

- Default configurations allowing anonymous connections.
- Missing per-topic permissions.
- Weak password files stored in clear text.
- Susceptibility to resource exhaustion attacks (malicious clients sending rapid SUBSCRIBE/PUBLISH operations).

Industrial vendors such as Siemens and Schneider Electric explicitly warn about these risks in their MQTT deployment guidelines.

6.5 Transport-Layer Exposure

MQTT's use of TCP makes it vulnerable to:

- Man-in-the-middle attacks (if TLS not enforced),
- Session hijacking,
- Shodan/Censys enumeration of public brokers,
- Brute-force authentication attempts.

Many large-scale MQTT exposures reported in industry occurred due to unsecured TCP port 1883 being open to the Internet.

7 Real-World Attacks on MQTT (Documented Incidents)

The following real-world incidents represent some of the most widely cited and academically validated categories of MQTT security failures observed in practice.

7.1 Internet-Wide Scans Exposing Thousands of Unsecured MQTT Brokers

Multiple academic studies, security researchers, and industry reports have conducted Internet-wide scans targeting port 1883 (unencrypted MQTT). These investigations consistently revealed severe global misconfigurations, including:

- Tens of thousands of MQTT brokers directly exposed to the public Internet without authentication or TLS.
- Anonymous or weak/default credentials, allowing anyone to authenticate as a valid client.
- Unrestricted topic access, enabling attackers to subscribe to arbitrary topics or publish unauthorized commands.

As a result, attackers were able to retrieve highly sensitive information, such as:

- Real-time GPS locations of vehicles, delivery fleets, and drones,
- Smart-building automation states, including door status, motion sensors, fire alarms,
- Industrial process values (e.g., temperature, pressure, flow rate).

This form of exposure represents one of the most common real-world MQTT security failures and is repeatedly highlighted in both academic surveys and Internet-measurement research.

7.2 Active Hijacking of MQTT-Connected Smart-Home Devices

Researchers analyzing consumer IoT ecosystems found that many devices—including smart locks, thermostats, HVAC systems, lighting controllers, and energy monitors—communicated using MQTT without TLS or access control.

In these systems, attackers were able to:

- Subscribe to all device topics using the wildcard character #, gaining full visibility into home sensor data and activity patterns.
- Send unauthorized actuator commands, such as unlocking smart locks, altering thermostat settings, or switching appliances on/off.
- Access privacy-sensitive information, including occupancy status and daily user routines.

These attacks have been documented in peer-reviewed studies, demonstrating how insecure MQTT deployments in consumer environments can directly compromise user safety and privacy.

7.3 Industrial MQTT Attacks via Compromised Gateways

In industrial environments, MQTT is commonly used to relay telemetry from field devices through gateways into supervisory or cloud platforms. Several real-world reports from major industrial vendors describe incidents in which misconfigured industrial MQTT deployments were exploited.

Key observations include:

- Compromised or misconfigured field gateways allowed unauthenticated external attackers to publish forged control commands to industrial systems.
- Manipulation of operational data, where attackers injected false sensor readings into dashboards, HMIs, or analytics tools.
- Suppression of safety-critical alarms by overwriting retained alarm topics with benign or misleading values.
- Lateral movement opportunities, where MQTT brokers were directly connected to internal OT or IT networks, enabling attackers to pivot deeper into sensitive infrastructure.

These incidents show that insecure MQTT deployments in industrial control systems can cause not only data leakage but also direct manipulation of physical processes, potentially leading to operational disruption or safety hazards.

8 Hardening and Security Best Practices for MQTT

These recommendations follow official specifications (OASIS), industry whitepapers (Siemens, Schneider Electric), and security surveys (Gupta & Jain).

8.1 Enforce TLS for All MQTT Communications

- Use MQTT over TLS (port 8883).
- Reject unencrypted connections (disable port 1883).
- Use X.509 certificates for client authentication.
- Configure strong cipher suites.

(Supported in OASIS MQTT 5.0 specification and Siemens IIoT architecture whitepaper.)

8.2 Implement Strong Authentication & Authorization

- Disable anonymous login.
- Enforce username/password or mutual TLS certificates.
- Use per-topic ACLs (Mosquitto supports granular ACL rules).
- Enforce role-based access (publisher vs subscriber privileges).

8.3 Protect Topic Namespace

- Restrict wildcard subscriptions.
- Avoid publishing control or configuration commands under public namespaces.
- Separate telemetry topics (`tele/`) from command topics (`cmd/`), a best practice in IIoT standards like Sparkplug B.

8.4 Harden the Broker

- Keep the broker updated (Mosquitto, EMQX, HiveMQ).
- Limit connection rate to mitigate DoS.
- Use persistent storage with safe permissions.
- Disable unauthenticated WebSocket connections.

Schneider Electric recommends placing MQTT brokers behind firewalls or DMZs.

8.5 Network Segmentation

- Place MQTT brokers in DMZ zones between IT and OT.
- Do not expose the broker directly to the Internet.
- Limit inbound traffic using firewall rules.

Siemens Industrial Edge documentation explicitly recommends isolating MQTT traffic from critical control networks.

8.6 Logging, Monitoring, Anomaly Detection

- Enable broker audit logs (connect/disconnect, SUBSCRIBE/PUBLISH).
- Detect repeated ClientID conflicts (sign of impersonation).
- Monitor for scanning behavior and unusual wildcard subscriptions.
- Use IDS/IPS (Suricata, Zeek) with MQTT rulesets.

8.7 Secure Device Lifecycle

- Avoid hard-coded credentials in IoT devices.
- Implement secure boot and firmware integrity validation.
- Enforce key/certificate rotation.
- Limit retained messages for sensitive state information.

9 References

References

[1] PROFIBUS & PROFINET International (PI). PROFIBUS Technology and Application – System Description. Version 1.0, Karlsruhe, Germany: PI, 2020.

[2] A. Treytl and T. Sauter. A Review of PROFIBUS Protocol Vulnerabilities: Considerations for Implementing Authentication and Authorization Controls. In *Proceedings of the 9th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2004.

[3] PROFIBUS & PROFINET International (PI). PROFIBUS Design Guideline. Version 1.29, Karlsruhe, Germany: PI, 2020.

[4] PROFIBUS & PROFINET International (PI). PROFIBUS Commissioning Guideline. Version 1.23, Karlsruhe, Germany: PI, 2018.

[5] K. Zetter. An Unprecedented Look at Stuxnet, the World's First Digital Weapon. *WIRED*, November 3, 2014. `https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/`.

[6] D. Albright, P. Brannan, and C. Walrond. Did Stuxnet Take Out 1,000 Centrifuges at the Natanz Enrichment Plant? *ISIS Reports*, Institute for Science and International Security, December 22, 2010. `https://isis-online.org/isis-reports/did-stuxnet-take-out-1000-centrifuges-at-the-natanz-enrichment-plant/`.

[7] Kaspersky Lab. Stuxnet Definition & Explanation. Moscow, Russia: Kaspersky, 2017. `https://www.kaspersky.com/resource-center/definitions/what-is-stuxnet`.

[8] OASIS. MQTT Version 3.1.1 Plus Errata 01. OASIS Standard, December 2015. `https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/`.

[9] OASIS. MQTT Version 5.0. OASIS Standard, March 2019. `https://docs.oasis-open.org/mqtt/mqtt/v5.0/`.

[10] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar. Secure MQTT for the Internet of Things (IoT). In *Proceedings of the 5th International Conference on Communication Systems and Network Technologies (CSNT)*, IEEE, 2015.

[11] Siemens AG. Industrial Edge and MQTT Communication Whitepaper. Munich, Germany: Siemens, 2021. `https://new.siemens.com/global/en/products/automation/topic-areas/industrial-edge.html`.

[12] Eclipse Foundation. Mosquitto MQTT Broker — Project Documentation. Eclipse IoT Working Group, 2020. `https://mosquitto.org/documentation/`.

[13] A. Gupta and R. Jain. A Survey of Security Vulnerabilities in MQTT Protocol for IoT. *Journal of Information Security and Applications*, Elsevier, 2020.

[14] Schneider Electric. IIoT Architecture Using MQTT and EcoStruxure. Technical Whitepaper, 2020. `https://www.se.com/ww/en/work/services/industrial-automation/`.

[15] OASIS. MQTT Version 5.0: Part 1 – Core Specification. OASIS Standard, 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[16] ISO/IEC. ISO/IEC 20922:2016 – Information technology – Message Queuing Telemetry Transport (MQTT). International Organization for Standardization, Geneva, 2016.

[17] Eclipse Foundation / Cirrus Link. Sparkplug B Specification v3.0. Eclipse IoT Working Group, 2020. https://projects.eclipse.org/projects/iot.sparkplug/specifications.

[18] International Telecommunication Union (ITU). MQTT for Sensor Networks (MQTT-SN) Protocol Specification. ITU-T Draft Standard, 2022.

[19] Inductive Automation. Designing Large-Scale IIoT Architectures with MQTT and Sparkplug. Technical Whitepaper, 2021. https://inductiveautomation.com/resources/white-papers.

[20] Avast Threat Labs. Exposed MQTT Services in the Wild: Security Risks in Smart Home and IIoT Systems. Security Research Report, 2019. https://blog.avast.com/mqtt-security-issues.

[21] Airbus CyberSecurity. Whispers of the Machines: Exposing MQTT Hidden Talks. Technical Blog Post, 2024. https://protect.airbus.com/blog/exposing-mqtt-hidden-talks/.