technologies

MDPI

Review

Transport and Application Layer Protocols for IoT: Comprehensive Review

Ionel Petrescu, Elisabeta Niculae, Viorel Vulturescu, Andrei Dimitrescu and Liviu Marian Ungureanu *

Department of Theory of Mechanisms and Robots, National University of Science and Technology Politechnica Bucharest, 060042 Bucharest, Romania

* Correspondence: ungureanu.liviu.marian@gmail.com or liviu.ungureanu@upb.ro

Abstract

The Internet of Things (IoT) connects billions of heterogeneous devices, necessitating lightweight, efficient, and secure communication protocols to support a diverse range of use cases. While physical and network-layer technologies enable connectivity, transport and application-layer protocols determine how IoT devices exchange, manage, and secure information. The diverse and constrained nature of IoT devices presents a challenge in selecting appropriate communication protocols, with no one-size-fits-all solution existing. This article provides a comprehensive review of key transport and application protocols in IoT, including MQTT, MQTT-SN, CoAP, LwM2M, AMQP, XMPP, WebSockets, HTTP/HTTPS, and OPC UA. Each protocol is examined in terms of its design principles, communication patterns, reliability mechanisms, and security features. The discussion highlights their suitability for different deployment scenarios, ranging from resource-constrained sensor networks to industrial automation and cloud-integrated consumer devices. By mapping protocol characteristics to IoT requirements, such as scalability, interoperability, power efficiency, and manageability, the article provides guidelines for selecting the optimal protocol stack to optimize IoT system performance and long-term sustainability. Our analysis reveals that while MQTT dominates cloud telemetry, CoAP and LwM2M are superior in IP-based constrained networks, and emerging solutions like OSCORE are critical for end-to-end security.

Keywords: IoT; application protocols; MQTT; CoAP; LwM2M; AMQP; OPC UA; security

check for updates

Academic Editor: Kyoung-Don (KD) Kang

Received: 23 October 2025
Revised: 26 November 2025
Accepted: 6 December 2025
Published: 11 December 2025

Citation: Petrescu, I.; Niculae, E.; Vulturescu, V.; Dimitrescu, A.; Ungureanu, L.M. Transport and Application Layer Protocols for IoT: Comprehensive Review. *Technologies* **2025**, *13*, 583. https://doi.org/10.3390/technologies13120583

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Internet of Things (IoT) [1,2] refers to physical objects embedded with sensors, actuators, and technologies to exchange data with other devices and systems on the network [3–5]. IoT is rapidly transforming industries, cities, and homes by embedding connectivity into billions of devices ranging from tiny sensors to complex industrial machinery. At the heart of this transformation lies the ability of devices to communicate seamlessly with each other and with cloud services. While lower-layer technologies—such as cellular, Wi-Fi, or low-power wide-area networks—provide the physical means of connectivity, it is the "transport and application protocols" that shape how data is exchanged, interpreted, and acted upon. Currently, no universally accepted reference architecture for IoT exists, although many architectures are available for IoT systems [3].

Transport and application layers define the "language" of IoT communication; they determine whether data is delivered reliably, how often devices synchronize, whether communication is synchronous or event-driven, and how secure the exchange is against

interception or tampering. Choosing the right protocol is a critical design decision that affects scalability, efficiency, interoperability, and long-term viability of IoT deployments.

Unlike traditional internet applications, IoT systems must often function under severe constraints—limited bandwidth, intermittent connectivity, low power budgets, and minimal processing capability. To meet these challenges, a range of specialized protocols has emerged, each optimized for particular device classes, network environments, and application requirements. From the publish/subscribe efficiency of MQTT to the RESTful simplicity of CoAP and the lifecycle management capabilities of LwM2M, these protocols provide the foundation for robust and secure IoT ecosystems.

This article explores the landscape of transport and application protocols for IoT, examining their technical foundations, communication models, and deployment scenarios. By understanding their strengths and trade-offs, system architects and developers can make informed choices to ensure their IoT solutions are functional, scalable, resilient, and secure.

2. Context

2.1. Quick Taxonomy

The Network layer of IoT architecture enables IoT devices to communicate with the Middleware layer by including several protocols [6].

Physical/Link/PHY + MAC: radio and link technologies (BLE, IEEE 802.15.4, LoRa/LoRaWAN, Sigfox, Z-Wave, cellular NB-IoT/LTE-M, 5G-NR, Wi-Fi).

Adaptation/Network: IPv6 adaptation for constrained networks (6LoWPAN), mesh routing (RPL), IP vs. non-IP [7].

Transport and Messaging/Application: UDP/TCP, HTTP(s), MQTT, CoAP, AMQP, MQTT-SN, LwM2M, WebSockets, XMPP, OPC UA.

Security and Management: TLS/DTLS/OSCORE/ACE, FOTA, device provisioning, authentication (OAuth/ACE), device management (OMA LwM2M, TR-069).

2.2. Physical and Link Layer: Choices and Trade-Offs

2.2.1. Short-Range, Low-Power (Home/Building/Personal-Area)

Bluetooth Low Energy (BLE/Bluetooth 5.x/BLE Mesh)

Range: short (tens of meters; Bluetooth 5 extended modes increase range).

Power: extremely low (good for coin-cell devices).

Topologies: star (classic BLE) or mesh (BLE Mesh).

Use cases: wearables, asset tags, smartphone-to-device interactions, short-burst sensors.

IEEE 802.15.4 family (Zigbee, Thread, Raw 802.15.4)

Low power, mesh-friendly; Thread supports native IPv6 (via 6LoWPAN). Good for home automation. Zigbee is mature in home/building automation and has an ecosystem of devices.

2.2.2. Long-Range, Low-Rate (LPWAN)

LoRa/LoRaWAN (unlicensed ISM bands). Very long range (km scale—tens of km in rural conditions; a few km in dense urban). Very low data rates (≈0.3–50 kbps), excellent battery life for infrequent uplinks. Adaptive Data Rate (ADR) to balance range vs. time-on-air. Use for meters, agriculture, and remote sensors [8].

Sigfox (proprietary LPWAN in ISM; ultra-narrowband)—similar class to LoRaWAN but vendor/cloud driven.

NB-IoT (Narrowband-IoT) and LTE-M (eMTC) (cellular, licensed spectrum)

NB-IoT: optimized for deep indoor penetration and very low power; low throughput, narrowband.

LTE-M: higher bandwidth and lower latency than NB-IoT, supports mobility better (handovers), slightly higher power. Choose NB-IoT for extreme range/penetration and LTE-M for slightly richer data and mobile devices [9,10].

2.2.3. Wi-Fi and 5G

Wi-Fi (2.4/5/6/6E): high throughput, moderate power—good for devices with ample power or where infrastructure exists.

5G (NB-IoT/RedCap/URLLC slices): emerging IoT capabilities (URLLC for low latency, RedCap for reduced capability devices). Cellular options allow QoS and broad coverage, but at higher device and service costs.

In summary, BLE/802.15.4 is recommended for dense local networks and user-interaction devices; LoRaWAN/NB-IoT/LTE-M for wide-area low-data sensors; Wi-Fi/5G for high throughput/low latency or when power is available.

2.3. Network and Adaptation: IP for Constrained Devices

6LoWPAN: compression/fragmentation rules that let IPv6 run over IEEE 802.15.4 (small frames). Essential where you want "native IP" and RESTful stacks on tiny devices (e.g., Thread). See IETF 6Lo working group materials [11,12].

RPL (Routing Protocol for Low-Power and Lossy Networks): a routing protocol designed for multipoint-to-point traffic patterns common in IoT (sensors to collector). RPL builds DAGs (Directed Acyclic Graphs) and is widely referenced in constrained networks. RFC 6550 defines it [13].

IP vs. non-IP: IP (with 6LoWPAN) simplifies integration with cloud services and uses standard web protocols (CoAP/HTTP over IPv6). Non-IP stacks (some LPWANs or proprietary protocols) may be simpler/cheaper but require gateways that translate to IP.

2.4. Transport and Application Protocols—The Most Important Layer for Interoperability

2.4.1. MQTT (Message Queuing Telemetry Transport)

Pattern: publish/subscribe (brokered).

Transport: typically TCP (reliable), optional WebSocket transport. Lightweight headers; QoS levels (0/1/2) for reliability choices. MQTT 5.0 adds richer metadata and control features. Best for telemetry, real-time dashboards, and many-to-many pub/sub flows. OASIS/MQTT.org hosts the spec [14,15].

Pros: simple, low overhead, good for unreliable links with QoS control; fits well with cloud brokers (AWS IoT, Azure IoT, etc.).

Cons: relies on TCP (overhead, connection setup)—not ideal for extremely constrained radio stacks without TCP, or extremely low duty cycle LPWANs (where connection setup is costly).

2.4.2. CoAP (Constrained Application Protocol)

Pattern: request/response (RESTful), with observe (pub/sub-like) extension.

Transport: UDP (low overhead), optionally secured with DTLS or use OSCORE for end-to-end application layer security. RFC 7252 is the CoAP spec and is explicitly designed for constrained devices/networks [16,17].

Pros: very small header (4 bytes base), supports multicast, easy mapping to HTTP/REST, low overhead fits 6LoWPAN/802.15.4.

Cons: being UDP-based, reliability and ordering must be handled at the application layer where required; DTLS introduces state and memory usage that must be accounted for [18].

2.4.3. MQTT-SN (MQTT for Sensor Networks)

A variant of MQTT designed to run over non-TCP transports (e.g., UDP) or constrained networks. Consider when you like MQTT's semantics but cannot use TCP.

LwM2M (Lightweight M2M)

Device management + telemetry in one stack (from OMA). Uses CoAP for transport and adds resource modeling, firmware updates (FOTA), device management operations. Very commonly used in cellular IoT deployments for remote provisioning and management.

AMQP, XMPP, WebSockets, HTTP(S)

AMQP: heavier, feature-rich (transactions, routing), used where enterprise messaging features are needed.

HTTP/HTTPS: ubiquitous, but verbose; OK for devices with decent RAM/throughput, not ideal on very constrained radios.

WebSockets: useful for browser-to-broker or to tunnel MQTT over WebSocket when needed.

2.4.4. OPC UA

Rich industrial protocol (data models, security). Use in industrial automation contexts, sometimes combined with MQTT/AMQP [19] for cloud transport.

Practical comparison (very high level):

Most constrained (tiny RAM, 802.15.4) → CoAP + 6LoWPAN + RPL [16].

Telemetry/cloud/many clients interested → MQTT (brokered) [15].

Device management + FOTA → LwM2M (CoAP-based).

Very low duty-cycle LPWAN (LoRaWAN) → often send tiny uplinks to a cloud via gateway and use simple application payloads; MQTT commonly used on the backend between network server and application [20,21].

2.5. Security: Practical Stack and Pitfalls

Security is not optional [22,23]. The basis for secure communication in IoT systems is enrolling cryptographic material into end devices and binding their identities with those materials [24]. Important elements:

Transport/Session Security: TLS for TCP-based (MQTT over TLS), DTLS for UDP/CoAP. But DTLS can be heavy for constrained devices; consider OSCORE (object-security for CoAP) that secures at the application layer and works with intermediaries/gateways [25].

Authentication and Authorization: Use strong device identity and provisioning. OAuth2 profiles for constrained environments (ACE framework) and ACE profiles for MQTT exist (RFC 9431 for an ACE profile for MQTT). Avoid static default passwords [26,27].

Firmware updates (FOTA): must be authenticated and integrity-checked (signed). Device management standards like LwM2M include secure FOTA flows.

Key management and provisioning: one of the hardest operational problems. Use secure manufacturing provisioning, TPM/secure elements, and avoid ad hoc symmetric key sharing.

Gateway implications: when a non-IP technology (LoRaWAN, Sigfox) uses a gateway, end-to-end security must be designed carefully—gateways often terminate radio security and then forward to network servers.

2.6. Practical Deployment Considerations and Performance Trade-Offs

2.6.1. Message Size and Fragmentation

Constrained links (802.15.4) have small MTUs; CoAP/6LoWPAN minimize header overhead and avoid fragmentation. LoRaWAN has strict payload-size limits depending on region and data rate—design messages to fit a single uplink when possible [8].

2.6.2. Latency vs. Battery Life

Frequent keepalives or TCP handshakes kill battery life. MQTT over long-lived TCP sessions with keepalives can work if the network supports low-power modes, but if devices only wake rarely, a simpler single-shot UDP/LoRa uplink (with backend aggregation) is more efficient.

2.6.3. Reliability and QoS Levels

Use MQTT QoS to tune reliability (0/1/2). CoAP has confirmable vs. non-confirmable messages and retransmit rules. Design for the communication pattern (e.g., telemetry vs. command-and-control).

2.6.4. Scalability and Multi-Tenant Cloud Integration

MQTT scales well with broker clusters; CoAP fits well in microcontroller ecosystems, talking to border proxies that translate to HTTP/MQTT for cloud ingestion.

2.6.5. Interoperability

Prefer IP and standards (6LoWPAN + CoAP or MQTT) to reduce gateway translation complexity. Thread (IPv6 native) is often preferable to Zigbee where end-to-end IP is desired.

2.7. Typical Stacks and Recommended Choices (By Use Case)

2.7.1. Battery-Powered Sensors in a Wide Area (e.g., Agriculture, Asset Monitoring)

Physical: LoRaWAN or NB-IoT, depending on spectrum and operator availability.

Backend: LoRaWAN network server → application server → MQTT/Webhooks for cloud ingestion.

Why: LoRaWAN maximizes battery life (years), NB-IoT gives more reliability and licensed spectrum [9,20].

2.7.2. Home/Building Automation (Many Nodes, Mesh, IP)

Physical: 802.15.4 (Thread) or BLE Mesh.

Network: 6LoWPAN + Thread, RPL for routing if needed.

Application: CoAP (for constrained nodes) or MQTT if a broker is available at the edge/gateway [11,13].

2.7.3. Industrial IoT (Low Latency, Rich Models)

Physical: Ethernet/Wi-Fi/private 5G/industrial wireless.

Application: OPC UA for local OT, MQTT/AMQP for cloud telemetry; use TLS + mutual auth and local edge gateways (protocol translators).

2.7.4. Smartphones/Consumer Devices Interacting with Cloud

Physical: Wi-Fi/BLE.

Application: HTTPS or MQTT over WebSockets; use OAuth2/OpenID Connect for user auth.

2.8. Management, Provisioning, and Lifecycle

Device management: OMA LwM2M is the de facto for remote management + FOTA in cellular IoT.

Provisioning: Use secure enrollment (establish unique keys; use signed device certificates or secure elements).

Monitoring: telemetry of radio link metrics (RSSI, SNR), battery, and application-level health.

2.9. Interoperability and Gateways (Practical Notes)

Many deployments use heterogeneous stacks and gateways that translate from the field protocol to an IP/cloud protocol. Gateways are [28] a pragmatic choice but add complexity: they are failure points, and they often break end-to-end security unless designed for it (examples: LoRaWAN gateways vs. network servers). Consider edge processing to reduce cloud bandwidth and latency.

2.10. Implementation and Testing Tools (Short List)

MQTT brokers/tests: Mosquitto, EMQX, HiveMQ (broker software and test clients).

CoAP tooling: libcoap, CoAP clients (coap-client), Californium.

6LoWPAN/RPL stacks and simulators: Contiki-NG, RIOT OS, Cooja emulator.

LPWAN testing: LoRaWAN network/server testbeds, Semtech tools.

Security testing: TLS/DTLS scanners, Fuzzing frameworks for constrained implementations.

2.11. Key Standards and Readings (Authoritative Sources)—The Five Most Load-Bearing References I Used

CoAP (RFC 7252)—foundational spec for constrained RESTful devices [12].

MQTT (OASIS/MQTT 5.0)—modern publish/subscribe standard for IoT telemetry [9].

LoRaWAN Specification (LoRa Alliance)—LPWAN spec: data rates, ADR, long-range trade-offs [20].

RPL (RFC 6550)—routing protocol for low-power and lossy networks used with 6LoWPAN [11].

3GPP documentation and vendor whitepapers for NB-IoT and LTE-M—comparison of NB-IoT vs. LTE-M trade-offs (coverage vs. throughput vs. mobility) [9,29].

Figure 1 illustrates the representative protocol stacks used in the Internet of Things (IoT), highlighting the layered relationship between network, transport, and application protocols. Each stack demonstrates how specific technologies are combined to support different deployment environments, ranging from cellular IoT with CoAP/LwM2M to industrial gateways with OPC UA, and consumer devices using Wi-Fi with MQTT or HTTPS [30,31]. By visualizing these stacks, the diagram emphasizes how design choices at each layer influence power consumption, reliability, scalability, and integration with cloud or enterprise platforms.

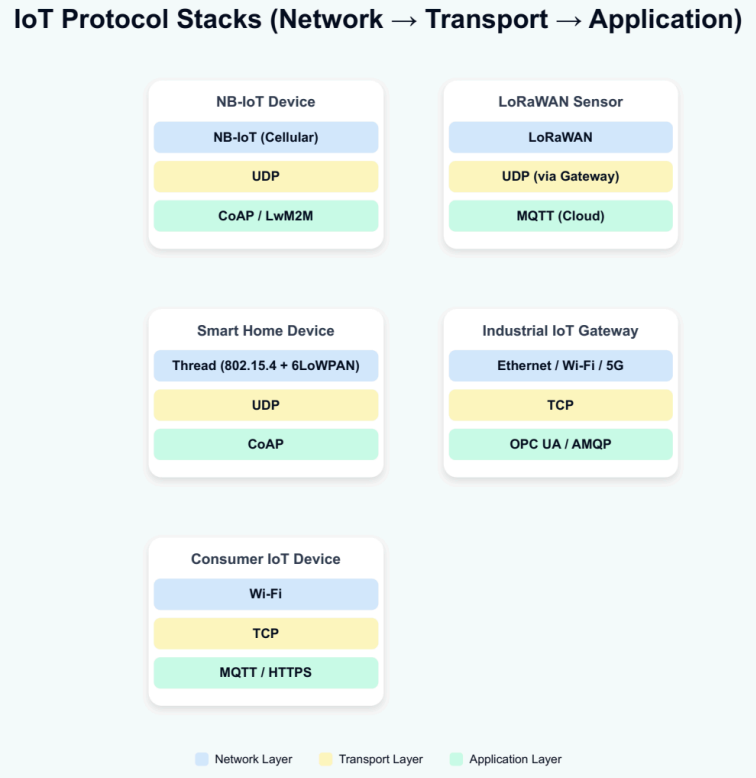


Figure 1. Protocol stacks used in the Internet of Things.

2.12. Common Mistakes and Pitfalls

- Relying on default/unprotected devices or hardcoded passwords.
- Ignoring MTU/payload limits (LoRaWAN and IEEE 802.15.4) [20]
- Using TCP everywhere on extremely constrained radios (connection setup leads to rapid battery depletion).
- Designing for the "lab" radio environment, not the reality (penetration, multi-path, duty cycle limits).

3. Transport and Application Protocols

3.1. MQTT and MQTT-SN

3.1.1. Overview for MQTT and MQTT-SN

MQTT (Message Queuing Telemetry Transport) is one of the most widely used IoT messaging protocols. It is a lightweight and flexible messaging protocol originally developed by IBM in the late 1990s and has since become an OASIS standard due to its simplicity, efficiency, and suitability for constrained devices and unreliable networks [4,32–34]. Its core design revolves around a publish/subscribe model, where devices (clients) exchange information via a central broker. This decoupling of message producers and consumers allows for highly scalable and flexible IoT deployments.

MQTT-SN (MQTT for Sensor Networks) is an adaptation of MQTT specifically for low-power, low-bandwidth, or non-TCP networks such as Zigbee, LoRaWAN, and UDP-based links. It retains MQTT's publish/subscribe semantics while reducing message overhead and accommodating constrained environments [10].

3.1.2. Communication Model for MQTT and MQTT-SN

Publish/Subscribe Architecture: Devices publish messages to topics; interested subscribers receive updates via the broker. This allows many-to-many communication without

direct device-to-device connections. The asynchronous nature of message exchange simplifies the communication model [6].

Broker Role: The broker manages subscriptions, routes messages, and enforces access controls, simplifying client logic.

Topic Hierarchy: Topics are structured hierarchically (e.g., 'home/livingroom/temperature') to organize data streams efficiently.

3.1.3. Transport Layer for MQTT and MQTT-SN

MQTT: Typically uses TCP, providing reliable, ordered delivery. TCP ensures message integrity and is widely supported in IoT devices connected via Wi-Fi, Ethernet, or cellular networks [8].

MQTT-SN: Designed for non-TCP transports, such as UDP, Zigbee, or proprietary links. Topic names can be replaced with short numeric IDs to reduce message size and header overhead [10].

3.1.4. Quality of Service (QoS)

MQTT supports three levels of message delivery reliability:

QoS 0—At most once. Messages are sent without acknowledgment (fire-and-forget).

QoS 1—At least once. Messages are acknowledged by the broker; duplicates are possible.

QoS 2—Exactly once. Uses a two-phase handshake to ensure a message is delivered only once, at the cost of higher latency and processing overhead.

MQTT-SN supports similar QoS semantics, adapted for constrained transports [10].

3.1.5. Security for MQTT and MQTT-SN

Transport Security: TLS is commonly used to protect data in transit.

Authentication: Username/password, client certificates, or token-based authentication can be applied.

Lightweight Authorization: ACE/OAuth2 profiles are increasingly used for constrained devices and cloud-integrated deployments [9].

3.1.6. Strengths for MQTT and MQTT-SN

- Low message overhead and minimal protocol complexity.
- Efficient for battery-powered devices and networks with variable connectivity.
- Scalable from small home deployments to enterprise-grade IoT platforms.
- Supports asynchronous and decoupled communication, making it ideal for distributed telemetry and event-driven architectures.

3.1.7. Limitations for MQTT and MQTT-SN

- MQTT relies on a broker, creating a single point of failure unless high-availability brokers are deployed.
- Not inherently RESTful; integration with web services may require adapters or bridges.
- QoS 2 may be too heavy for highly constrained networks or ultra-low-power devices.

3.1.8. Typical Use Cases for MQTT and MQTT-SN

- Telemetry reporting for sensors and actuators.
- Smart home ecosystems where multiple devices share real-time status.
- Industrial IoT for data aggregation and monitoring.
- Cloud IoT platforms where scalable, many-to-many messaging is required.

3.2. CoAP

3.2.1. Overview for CoAP

CoAP, standardized by the IETF (RFC 7252), is a RESTful protocol designed for constrained devices and networks [11]. It mirrors HTTP's request/response model but is optimized for low-power sensors, low-bandwidth links, and intermittent connectivity. CoAP enables IoT devices to interact seamlessly using familiar methods such as GET, POST, PUT, and DELETE, while minimizing message overhead and power consumption [12]. CoAP is mainly used in constrained environments with limited devices and network resources [6].

Unlike MQTT, which follows a publish/subscribe model, CoAP implements a client/server architecture, making it suitable for direct device-to-device or device-to-gateway communication in mesh or star networks [12].

3.2.2. Transport Layer for CoAP

Primary Transport: UDP, chosen for lightweight communication and minimal protocol overhead.

Optional Transports: TCP or SMS in some deployments; CoAP over TCP allows compatibility with networks requiring reliable transport.

Header Overhead: Minimal—4-byte base header, significantly smaller than HTTP, reducing transmission costs for constrained devices.

3.2.3. Reliability and Messaging

Confirmable (CON) Messages: Reliable messages with acknowledgment (ACK). Retransmission is automatic in case of packet loss.

Non-Confirmable (NON) Messages: Fire-and-forget messages for low-latency or periodic updates.

Separate Responses: For long processing times, the server can send an ACK immediately and deliver the response later.

Duplicate Detection: Built-in message ID tracking prevents duplicate processing.

3.2.4. Extensions and Features

Observe: Enables subscription-like functionality, allowing clients to receive asynchronous updates from resources (similar to pub/sub).

Blockwise Transfers: Supports large payloads segmented into smaller blocks to accommodate constrained MTU sizes.

Multicast Support: Efficient group communication for simultaneous device updates.

Proxy and Caching Support: Interoperates with HTTP through proxies, allowing integration with existing web services [12].

3.2.5. Security for CoAP

DTLS (Datagram Transport Layer Security): Provides encryption, authentication, and integrity at the transport level. In 2016, the IETF published RFC 7925 that defines TLS/DTLS IoT profiles for IoT. The main goal of creating that document was to protect CoAP messages using DTLS [24,35].

OSCORE (Object Security for Constrained RESTful Environments): Provides end-to-end application-layer security, enabling secure messages even through untrusted proxies or gateways.

Lightweight Authentication: PSK (pre-shared keys) are common in constrained networks; certificate-based authentication is optional.

3.2.6. Strengths for CoAP

- Extremely lightweight, suitable for constrained devices and low-power networks.
- RESTful model allows easy integration with existing web infrastructure.
- Supports asynchronous notifications, multicast, and proxying.
- Flexible security options with DTLS and OSCORE.

3.2.7. Limitations for CoAP

- Less suited for many-to-many communication without additional observe subscriptions.
- Relies on UDP, which may be blocked or restricted in some networks.
- Requires more complex logic for retransmissions and duplicate detection compared to TCP-based protocols.

3.2.8. Typical Use Cases for CoAP

- Sensor and actuator networks in home or industrial IoT.
- IPv6-based networks, especially 6LoWPAN and Thread deployments.
- Energy-efficient monitoring systems, where low bandwidth and power consumption are critical.
- RESTful IoT APIs for direct device control or data collection.
- This section highlights CoAP as the ideal protocol for constrained, RESTful IoT applications, contrasting with MQTT's publish/subscribe model.
- Provides a detailed technical analysis of MQTT, CoAP, and others, specifically dedicated to application-level protocols [36].

3.3. LwM2M (Lightweight M2M)

3.3.1. Overview for LwM2M

LwM2M, developed by the Open Mobile Alliance (OMA), is a lightweight, scalable client-server protocol tailored for IoT device management [6]. It extends CoAP to provide not only data exchange but also remote configuration, monitoring, firmware update, and lifecycle management of devices. Designed specifically for constrained devices and low-power networks, LwM2M is well-suited for large-scale deployments, especially in cellular IoT applications such as NB-IoT and LTE-M [13].

Unlike CoAP, which focuses primarily on resource access, LwM2M defines a standardized object model for device management operations, including telemetry reporting, provisioning, and firmware update mechanisms [14].

3.3.2. Transport Layer for LwM2M

Primary Transport: UDP via CoAP.

Optional Transport: TCP in some extensions, offering improved reliability for high latency or lossy networks.

Security: Built on DTLS (Datagram Transport Layer Security) or OSCORE for end-to-end encryption, authentication, and integrity.

3.3.3. Features and Functionality

Resource Model: Devices expose a hierarchical structure of objects, instances, and resources with standardized IDs. Examples include device information, connectivity statistics, or sensor readings.

Device Management: Supports remote configuration, provisioning, monitoring, and diagnostics.

Firmware Over-The-Air (FOTA): Enables secure updates of device firmware, critical for long-lived IoT devices.

Bootstrapping: Facilitates secure initial provisioning of devices onto a network.

Telemetry Reporting: Efficient reporting of sensor data using CoAP observe or scheduled updates.

The LwM2M is a standard device and service management built on top of CoAP to ensure remote management and configuration of constrained and powerful devices [6].

3.3.4. Security for LwM2M

DTLS: Ensures confidentiality, integrity, and authentication at the transport level.

OSCORE: Provides end-to-end security, particularly useful when devices communicate through intermediate gateways.

Lightweight Keys and Certificates: Optimized for constrained devices, with support for pre-shared keys, raw public keys, or certificates.

3.3.5. Strengths for LwM2M

- Standardized object and resource model ensures interoperability across vendors and devices.
- Combines telemetry and device management in a single protocol, reducing implementation complexity.
- Efficient for constrained networks and devices with limited power and memory.
- Supports secure provisioning and firmware updates, enabling scalable IoT deployments.

3.3.6. Limitations for LwM2M

- Slightly more complex than CoAP alone, requiring device models to be implemented correctly.
- Performance may be impacted in extremely low-power or high-latency networks due to security and management overhead.
- Less suited for many-to-many pub/sub communication without additional brokers or gateways.

3.3.7. Typical Use Cases for LwM2M

- Large-scale cellular IoT deployments with NB-IoT or LTE-M.
- Remote device provisioning, configuration, and monitoring for smart meters, wearables, and industrial sensors.
- Secure firmware update management for devices deployed in the field for years.
- Any IoT scenario requiring both telemetry and lifecycle management.
- This section establishes LwM2M as the preferred protocol for managing constrained IoT devices at scale, building on CoAP while adding standardized device lifecycle capabilities.

3.4. AMQP (Advanced Message Queuing Protocol)

3.4.1. Overview for AMQP

AMQP is an open standard messaging protocol designed for reliable, enterprise-grade communication. It follows a broker-based, publish-subscribe model and uses TCP as the transport layer protocol [6]. Unlike lightweight IoT protocols such as MQTT or CoAP, AMQP emphasizes robustness, guaranteed message delivery, transactional messaging, and fine-grained routing capabilities. It is widely used in industrial and enterprise IoT systems where high reliability, ordered processing, and seamless integration with backend systems are critical.

AMQP's architecture supports queues, sophisticated routing, and a variety of messaging patterns, making it suitable for applications that require assured message delivery and processing even under failure conditions. Due to its relatively heavier protocol overhead

compared to MQTT or CoAP, AMQP is less suited for resource-constrained IoT devices but excels in scenarios demanding enterprise-grade messaging reliability and flexibility.

3.4.2. Transport Layer for AMQP

Transport: TCP, providing reliable, ordered delivery.

Message Framing: AMQP defines its own binary protocol over TCP to efficiently encode messages, headers, and delivery instructions.

Broker Role: Brokers handle routing, queuing, acknowledgments, and transactions, ensuring reliable delivery across distributed systems.

3.4.3. Communication Model for AMQP

Point-to-Point Queues: Messages are delivered to a single consumer with reliability guarantees.

Publish/Subscribe: Multiple consumers can subscribe to topics or exchanges, supporting flexible routing.

Transactions: AMQP supports transactional message delivery, ensuring atomic operations.

Acknowledgments: Consumers send acknowledgments to confirm successful processing of messages.

3.4.4. Security for AMQP

Transport Security: TLS provides encryption and integrity.

Authentication: SASL (Simple Authentication and Security Layer) mechanisms, including username/password, OAuth, and certificate-based authentication.

Access Control: Role-based access control is enforced at the broker level, allowing fine-grained permissions.

3.4.5. Strengths for AMQP

- Reliable and robust: Guarantees delivery, order, and transactions.
- Enterprise integration: Bridges IoT devices with IT systems, enterprise message buses, and industrial platforms.
- Flexible routing: Supports direct messaging, publish/subscribe, and complex routing topologies.
- Scalable: Suitable for high-throughput environments with multiple producers and consumers.

3.4.6. Limitations for AMQP

- Resource-intensive: Higher memory and processing requirements compared to lightweight protocols like MQTT or CoAP.
- Not ideal for constrained devices: TCP overhead and broker complexity make it unsuitable for very low-power or ultra-constrained IoT devices.
- Deployment complexity: Requires configuration of brokers and routing logic.

3.4.7. Typical Use Cases for AMQP

- Industrial IoT systems for monitoring and control, where message reliability is critical.
- Enterprise IoT deployments integrating sensors, actuators, and business applications.
- Scenarios requiring complex routing, guaranteed delivery, or transactional messaging, such as supply chain tracking and asset management.
- This section positions AMQP as a robust, enterprise-focused protocol, suitable for industrial IoT and large-scale deployments requiring high reliability and complex messaging patterns.

3.5. XMPP (Extensible Messaging and Presence Protocol)

3.5.1. Overview for XMPP

XMPP is an XML-based messaging protocol also known as Jabber. It is a standard initially designed for instant messaging and exchange of messages between applications, no matter which operating system they are using in IoT [6]. Over time, it has been extended for IoT and machine-to-machine (M2M) communication. XMPP provides a decentralized, extensible architecture with features such as addressing, presence notifications, and real-time message exchange. Its extensibility allows developers to define custom XML stanzas tailored to IoT applications.

Unlike lightweight IoT protocols like MQTT or CoAP, XMPP is verbose due to its XML encoding, making it less suitable for highly constrained devices but valuable where flexibility and extensibility are important.

3.5.2. Transport Layer for XMPP

Transport: TCP is the standard transport, often secured with TLS (XMPP over TLS is commonly referred to as XMPP + TLS).

Persistent Connection: Clients maintain long-lived TCP connections with the server to enable real-time messaging and presence tracking.

Message Framing: XML streams are used to structure messages, commands, and presence notifications.

3.5.3. Communication Model for XMPP

Client-Server Architecture: Devices connect to an XMPP server, which handles routing and addressing.

Publish/Subscribe Extensions (XEP-0060): Supports pub/sub patterns suitable for IoT data distribution.

Presence and Status: Devices can communicate availability, state, or operational status, enhancing network coordination.

Extensibility: Custom XML namespaces allow the addition of new message types and features without changing the core protocol.

3.5.4. Security for XMPP

TLS/SSL: Encrypts communication between clients and servers.

SASL Authentication: Supports username/password, certificate, or token-based authentication.

Access Control: Fine-grained permissions for pub/sub topics and message routing.

3.5.5. Strengths for XMPP

- Highly extensible, allowing custom features without protocol redesign.
- Supports presence, notifications, and structured messaging, useful for interactive IoT applications.
- A decentralized server network allows distributed deployments and federation.
- Mature ecosystem with libraries and tools for many platforms.

3.5.6. Limitations for XMPP

- XML verbosity increases bandwidth usage, making it less efficient for constrained networks.
- Long-lived TCP connections can be challenging for battery-powered devices.
- Higher memory and CPU requirements compared to lightweight protocols.
- Not ideal for extremely low-power or low-bandwidth IoT deployments.

3.5.7. Typical Use Cases for XMPP

- Smart home ecosystems where devices need to report presence and status.
- Interactive IoT applications requiring real-time notifications.
- Federated IoT networks where extensibility and interoperability are key.
- Scenarios where structured messaging, flexibility, and extensibility outweigh bandwidth efficiency concerns.

This section establishes XMPP as a flexible and extensible protocol for IoT applications that require structured messaging, presence awareness, and real-time interactions, while highlighting its trade-offs for constrained devices.

3.6. WebSockets

3.6.1. Overview for WebSockets

WebSockets is a full-duplex communication protocol that enables persistent, bidirectional connections between clients and servers over a single TCP connection. Originally designed for real-time web applications, WebSockets have become useful in IoT scenarios where low-latency, continuous communication is required, particularly for integrating IoT devices with web dashboards, mobile apps, or cloud platforms.

Unlike traditional HTTP, which follows a request/response model, WebSockets allows both client and server to send messages asynchronously without the overhead of repeated handshakes, making it more efficient for real-time data streaming.

3.6.2. Transport Layer for WebSockets

Transport: TCP, typically initiated via an HTTP/HTTPS handshake (‘ws://’ or ‘wss://’) and then upgraded to a WebSocket connection.

Persistent Connection: Once established, the TCP connection remains open, allowing continuous data flow.

Message Framing: Binary or text frames are exchanged efficiently over the established connection.

3.6.3. Communication Model for WebSockets

Full-Duplex Channel: Both endpoints can send and receive data at any time without waiting for a request or response.

Event-Driven Messaging: Supports real-time updates, notifications, and streaming telemetry.

Integration with IoT Protocols: Often used to tunnel MQTT or CoAP messages to web applications, bridging constrained devices with dashboards or cloud platforms.

3.6.4. Security for WebSockets

TLS Support: Secure WebSockets (‘wss://’) ensures encryption and integrity.

Authentication: Can leverage existing HTTP authentication mechanisms (cookies, tokens, OAuth).

Firewall-Friendly: Runs over standard ports (80/443), simplifying network traversal.

3.6.5. Strengths for WebSockets

- Enables real-time, low-latency communication between IoT devices and web/cloud applications.
- Persistent connection reduces overhead compared to repeated HTTP requests.
- Works seamlessly with existing web technologies and platforms.
- Supports both text and binary messaging.

3.6.6. Limitations for WebSockets

- TCP connection required: May not be ideal for extremely constrained or low-power devices.
- No native QoS or message reliability guarantees: Must be implemented at the application layer.
- Persistent connections may impact server scalability if many devices are connected simultaneously.

3.6.7. Typical Use Cases for WebSockets

- Streaming telemetry from IoT devices to web dashboards.
- Real-time control of smart home or industrial devices via web interfaces.
- Bridging IoT protocols (MQTT, CoAP) to cloud or web applications.
- Mobile applications that require instant updates from IoT networks.

This section positions WebSockets as an ideal protocol for real-time integration between IoT devices and web/cloud platforms, complementing lightweight IoT protocols like MQTT and CoAP for interactive applications.

3.7. HTTP/HTTPS

3.7.1. Overview for HTTP/HTTPS

HTTP (Hypertext Transfer Protocol) is the foundational protocol of the web, widely used for communication between clients and servers. In IoT, HTTP and its secure variant, HTTPS, are often employed for devices that are resource-capable, such as Wi-Fi appliances or gateways, to interact with cloud services, RESTful APIs, and web dashboards.

While HTTP is more verbose and heavyweight compared to IoT-optimized protocols like MQTT or CoAP, its ubiquity, compatibility, and support for standard web infrastructure make it a practical choice for many IoT deployments.

3.7.2. Transport Layer for HTTP/HTTPS

Transport: TCP, providing reliable and ordered delivery.

Security: HTTPS leverages TLS to ensure encryption, integrity, and server authentication.

Message Format: Text-based, with headers, cookies, and JSON/XML payloads, which adds overhead but supports rich metadata and web integration.

3.7.3. Communication Model for HTTP/HTTPS

Request/Response: Clients initiate requests (GET, POST, PUT, DELETE) to servers, which return responses.

Stateless: Each HTTP request is independent, which simplifies server design but may increase connection overhead for frequent messaging.

Integration: Works seamlessly with RESTful APIs, enabling standardized interaction with cloud services and other web-based systems.

3.7.4. Security for HTTP/HTTPS

HTTPS/TLS: Protects data in transit from eavesdropping and tampering.

Authentication: Supports a wide range of methods, including Basic, Digest, OAuth2, API keys, and certificates.

Authorization: Can leverage existing web access control mechanisms.

3.7.5. Strengths for HTTP/HTTPS

- Universally supported by web servers, cloud platforms, and IoT gateways.
- Simplifies integration with REST APIs, dashboards, and enterprise systems.

- Mature ecosystem with libraries, tools, and standards.
- Supports rich metadata, content negotiation, and web-friendly formats like JSON.

3.7.6. Limitations for HTTP/HTTPS

- Verbose headers and text-based format result in higher bandwidth usage.
- The stateless request/response model is less efficient for frequent or real-time messaging.
- Less suitable for battery-powered, low-bandwidth, or highly constrained devices.
- Requires TCP, which may not be ideal for some low-power networks.

3.7.7. Typical Use Cases for HTTP/HTTPS

- Cloud-connected devices with adequate resources (Wi-Fi appliances, gateways, edge devices).
- RESTful APIs for device telemetry, configuration, and control.
- Integration with web dashboards, cloud analytics platforms, and enterprise applications.
- Firmware and software update delivery for capable devices.

This section positions HTTP/HTTPS as the web-native protocol for IoT, suitable for capable devices and systems that prioritize interoperability with cloud and enterprise platforms over bandwidth or power efficiency.

3.8. OPC UA (Open Platform Communications Unified Architecture)

3.8.1. Overview for OPC UA

OPC UA is an industrial communication standard designed to provide secure, reliable, and interoperable data exchange in automation, manufacturing, and Industry 4.0 applications. Unlike lightweight IoT protocols like MQTT or CoAP, OPC UA emphasizes semantic interoperability, rich data modeling, and robust security, making it a preferred choice for industrial IoT (IIoT) deployments.

OPC UA is not just a messaging protocol—it defines a framework for data representation, allowing devices, sensors, machines, and control systems to expose structured data in a consistent, vendor-neutral format.

3.8.2. Transport Layer for OPC UA

Primary Transport: TCP with a binary encoding for efficiency.

Optional Transport: HTTPS for web-based integration or MQTT/AMQP for pub/sub scenarios.

Message Framing: Binary and XML encodings supported; binary preferred for performance-critical environments.

3.8.3. Communication Model for OPC UA

Client-Server Architecture: Standard OPC UA interactions include reading, writing, and subscribing to nodes in the server's address space.

Pub/Sub Extensions: Supports many-to-many messaging, event-driven data, and multicast communication for industrial networks.

Data Modeling: Devices expose an object-oriented information model, including nodes, variables, methods, and events, ensuring semantic consistency across heterogeneous systems.

3.8.4. Security for OPC UA

Transport Security: TLS provides encryption, integrity, and authentication.

Authentication: Certificates, username/password, or token-based methods.

Authorization: Role-based access controls for fine-grained permissions.

Integrity and Non-Repudiation: OPC UA ensures secure and traceable industrial communications.

3.8.5. Strengths for OPC UA

- Semantic interoperability: Standardized data models allow integration across vendors and systems.
- Flexible communication: Supports client-server and pub/sub patterns.
- Robust security: Comprehensive authentication, encryption, and authorization mechanisms.
- Industrial readiness: Suitable for mission-critical automation, control systems, and IIoT networks.

3.8.6. Limitations for OPC UA

- Higher resource requirements compared to lightweight IoT protocols.
- Implementation complexity can be significant, requiring specialized development tools.
- Less suitable for low-power, constrained devices typical in consumer IoT applications.

3.8.7. Typical Use Cases for OPC UA

- Industrial IoT deployments in manufacturing, energy, and process control.
- Integration of sensors, machines, and SCADA systems.
- Real-time monitoring and control in factory automation.
- Bridging IT and OT systems in Industry 4.0 initiatives.

This section positions OPC UA as the standard for industrial IoT, providing semantic interoperability, strong security, and reliable communication for complex automation and manufacturing environments.

Table 1 is a comparison table summarizing all the protocols, including range, throughput, power consumption, topology, security, and best use cases. This table summarizes the trade-offs of each protocol in terms of range, throughput, power efficiency, communication topology, security features, and practical applications, providing a clear guide for protocol selection.

Table 1. Summarizes of IoT Protocols.

Protocol	Range/Connectivity	Throughput	Power Consumption	Topology/Architecture	Security Options	Best Use Case
MQTT	Wi-Fi, Ethernet, Cellular, LPWAN	Low–Moderate	Low (suitable for battery devices)	Publish/Subscribe (Broker-based)	TLS, user-name/password, OAuth2	Telemetry, remote monitoring, cloud-integrated IoT
MQTT-SN	Zigbee, LoRaWAN, UDP-based networks	Low	Very Low	Publish/Subscribe (Broker-based)	DTLS, pre-shared keys	Constrained sensor networks and low-power devices
CoAP	6LoWPAN, Thread, Wi-Fi, LPWAN	Low–Moderate	Very Low	Client/Server, optional Observe (pub/sub-like)	DTLS, OSCORE	Low-power sensor/actuator networks, RESTful APIs
LwM2M	NB-IoT, LTE-M, 6LoWPAN, Thread	Low–Moderate	Low	Client/Server, Object-oriented resource model	DTLS, OSCORE	Device management, provisioning, firmware updates
AMQP	Ethernet, Wi-Fi, Cellular	Moderate–High	Moderate	Broker-based, point-to-point, or pub/sub	TLS, SASL, role-based access control	Industrial IoT, enterprise integration, reliable messaging
XMPP	Ethernet, Wi-Fi, Cellular	Low–Moderate	Moderate	Client/Server, pub/sub extensions	TLS, SASL, access control	Real-time notifications, presence-aware IoT applications
WebSockets	Wi-Fi, Ethernet, Cellular	Moderate–High	Moderate	Full-duplex persistent connection	TLS, HTTP authentication	Real-time dashboards, mobile apps, and cloud IoT integration
HTTP/HTTPS	Wi-Fi, Ethernet, Cellular	Moderate–High	Moderate–High	Client/Server (Request/Response)	TLS, OAuth2, API keys, certificates	RESTful APIs, cloud integration, web dashboards

Table 1. *Cont.*

Protocol	Range/Connectivity	Throughput	Power Consumption	Topology/Architecture	Security Options	Best Use Case
OPC UA	Ethernet, Wi-Fi, Industrial networks	Moderate–High	Moderate–High	Client/Server, pub/sub, object-oriented model	TLS, certificates, user-name/password, RBAC	Industrial automation, factory monitoring, IIoT networks

4. Discussion, Comparative Analysis, and Research Synthesis

IoT protocols find applications in many areas. In smart cities, for example, end-to-end systems have been developed that use MQTT for monitoring and alerting traffic or urban infrastructure [36]. Thanks to the pub/sub model, MQTT decouples transmitters from receivers, facilitating communication between millions of sensors and urban management platforms [37]. In industry (IIoT), IoT platforms with MQTT and CoAP support are used in automation; in fact, Things Board is frequently used for applications in smart agriculture, utility networks, smart cities, and industrial automation [37].

In precision agriculture, lightweight protocols are essential: MQTT and CoAP meet the requirements of field sensors with efficient traffic and low power. Furthermore, specialized extensions such as MQTT-SN (Sensor Networks) are ideal for resource-constrained sensor networks on farms, offering efficient coding and minimal overhead. Also, LPWAN technologies (LoRaWAN, SigFox) are widespread in agriculture due to their long-distance coverage and low consumption [38].

In the healthcare sector, IoT protocols enable continuous patient monitoring: MQTT is appreciated for its reliability and scalability, including features such as "Last Will", which alerts when a medical device loses connection. However, the need to secure medical data is highlighted, as studies note that MQTT, although effective, is not inherently a strong protection mechanism [39].

Overall, the mentioned protocols are integrated into real systems in all these domains, demonstrating their flexibility and efficiency in IoT. Table 2 presents the performances of the protocols by device class.

Table 2. Performances of the protocols.

Protocol	Mode	Device Class	Latency (ms)	Throughput	Packet Loss (%)	Reliability	Notes/References
MQTT	QoS 0	MCU	50–70	120–150 msg/s	2%	Moderate	Very lightweight [40]
MQTT	QoS 0	Edge	45–60	200 msg/s	1.5%	High	Handles higher loads [40]
MQTT	QoS 1	MCU	65	100–130 msg/s	1%	Very High	ACK adds overhead [41]
MQTT	QoS 1	Edge	60	180–200 msg/s	0.8%	Very High	Minimal latency impact [41]
MQTT	QoS 2	MCU	90–110	80–120 msg/s	0%	Highest	CPU-heavy for MCUs, QoS2 overhead [42]
MQTT	QoS 2	Edge	90	160 msg/s	0%	Highest	Good for critical logs [42]
CoAP	Non-confirmable	MCU	40	150–200 msg/s	5%	Moderate	Best-effort mode [43]
CoAP	Non-confirmable	Edge	35–40	200–220 msg/s	3%	Moderate	Efficient for telemetry [43]
CoAP	Confirmable	MCU	60–80	100–130 msg/s	2%	High	Based on retransmission logic [44,45]
CoAP	Confirmable	Edge	50–65	140–160 msg/s	1%	Very High	Still lightweight [44]
HTTP	HTTP/1.1	MCU	100–150	50–100 req/s	0.5%	High	TCP overhead [46];
HTTP	HTTP/1.1	Edge	90–120	100–150 req/s	0.5%	High	Suitable for dashboards [46]
HTTPS	TLS-enabled	MCU	200–250	40–80 req/s	0%	Very High	TLS cost; TLS handshake expensive [47]
HTTPS	TLS-enabled	Edge	180–210	80–120 req/s	0%	Very High	Easily handled [47]

Recently, the main communication protocols for IoT (e.g., MQTT, CoAP, LwM2M, AMQP) have been analyzed in detail in the context of energy efficiency and performance. Comparative studies show that publish/subscribe protocols such as MQTT can reduce energy consumption compared to HTTP. For example, in cold chain monitoring applications, the use of MQTT (QoS 0 and 1) led to energy savings of 6.03–8.33% compared to HTTP [48–51], as shown in Table 3 and Figure 2.

Table 3. Comparison of the energy saving of protocols.

Protocol/Variant	Average Power (mW, Full System)	σ (mW)	Device Class	Relative to HTTP (%)	Energy Savings vs. HTTP (%)	Notes
HTTP/HTTPS	670	±16	NodeMCU (microcontroller)	100% (baseline)	0%	Verbose headers/connections; highest for request/response.
MQTT QoS 2	650–660	±20	NodeMCU (microcontroller)	97–98%	2–3%	Estimated (QoS handshakes add ~5–8% vs. QoS 1); avoid in battery ops.
MQTT QoS 1	614	±22	NodeMCU (microcontroller)	92%	8%	At least once; lowest MQTT draw in this workload (optimized ACKs).
MQTT QoS 0	630	±19	NodeMCU (microcontroller)	94%	6%	Fire-and-forget; scalable but slightly higher than QoS 1 here.
CoAP CON (Confirmable)	600–620	±18	Z1 mote (microcontroller)	90–93%	7–10%	ACK equiv. to QoS 1; RESTful, good for edge.
CoAP NON (Non-Confirmable)	590–610	±17	Z1 mote (microcontroller)	88–91%	9–12%	Stateless UDP; top for loss-tolerant telemetry.
MQTT-SN (over UDP)	550–580	±15	Z1 mote (microcontroller)	82–87%	13–18%	Best overall; topic-ID efficiency for non-IP nets like LPWAN.

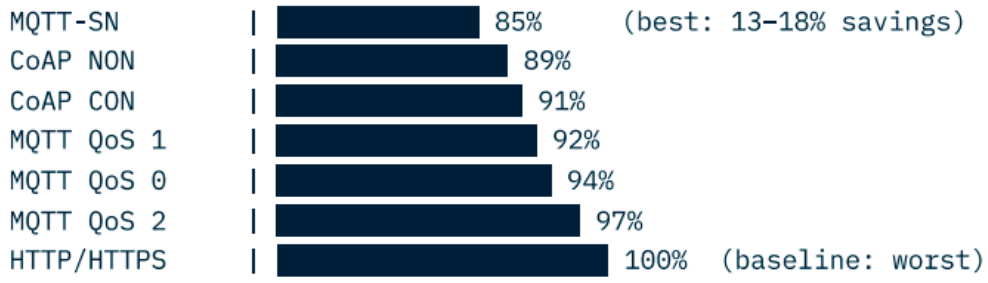


Figure 2. Relative Energy Consumption (% of HTTP Baseline).

In general, on Raspberry Pi and microcontroller IoT devices, MQTT has been observed to provide lower power consumption than HTTP, especially at high QoS. More recent analyses indicate that CoAP, due to its RESTful nature and UDP transport, offers the highest power efficiency (with lower power consumption than MQTT or HTTP). In simulations comparing MQTT, CoAP, MQTT-SN, AMQP, and HTTP, the MQTT-SN and CoAP protocols (unconfirmed mode) showed the best power efficiency, while TCP-based protocols (HTTP, AMQP) consume significantly more. For example, MQTT-SN was found to consume even less than CoAP, and HTTP was the most powerful inefficient [51]

Performance in terms of latency and throughput depends on the configurations. For example, in TLS 1.3 secured MQTT communication, increases of up to ~23% in execution time and energy compared to unsecured communication for the authentication phase have been measured [52]. At the same time, evaluation of different cipher suites shows that ChaCha20_Poly1305 is more efficient than AES_256_GCM on ARM hardware, requiring fewer CPU cycles and consuming less energy.

IoT protocols include standardized security solutions such as DTLS (for CoAP) and TLS (for MQTT) and dedicated protocols for constrained devices such as OSCORE (Object Security for Constrained Environments). These mechanisms provide confidentiality and integrity but also involve processing costs. In IoT environments, OSCORE and DTLS are

widely adopted and provide standard security mechanisms [53] but require pre-configured trusted infrastructure. In experimental analyses, introducing TLS 1.3 over MQTT was feasible, but generated significant overhead (e.g., +6–23% in processing and energy in the key exchange and authentication phases) [52].

In addition, proper encryption and authentication are essential: although MQTT has mechanisms such as Last Will to detect disconnections, the lack of robust security features makes the protocol vulnerable to cyberattacks. Frequently, traditional authentication methods (name/password) are not sufficient in IoT, being susceptible to MitM or replay attacks [54,55]. Thus, lightweight authentication mechanisms (e.g., based on symmetric cryptography, certificate networks, or pre-distributed key solutions) are needed to protect resource-constrained environments.

The complexity of IoT networks requires interoperability between various protocols and standards. Management protocols, such as oneM2M or OGC SensorThings, provide integration frameworks that can encompass MQTT, CoAP, LwM2M, data harmonization, etc. Such interconnection models have significantly improved transmission efficiency (e.g., by ~52.9% and reduced latencies) and enabled unified device management via LwM2M or MQTT [28].

The main problem is that LPWAN (e.g., LoRaWAN, SigFox) is now available without IP, and it is difficult to integrate into an existing IP architecture. In the current state, these non-IP technologies are now natively interoperable and require ad hoc integration solutions.

In the IoT context, data is often processed either at the edge or in the cloud, and IoT protocols facilitate both scenarios. Due to their lightweight features, MQTT or CoAP can be used in edge computing gateways to preprocess local data, reducing latency. For example, modern architecture distributes tasks between the cloud and the edge based on data urgency, leaving critical computations at the edge [56]. Cloud IoT platforms (e.g., AWS IoT, Azure IoT Hub, ThingsBoard, etc.) offer dedicated edge modules that support popular protocols and allow computing logic to run at the data source. For example, the open-source ThingsBoard platform supports MQTT, HTTP, and CoAP and includes an "Edge" component for distributed deployments close to the sensor source. Additionally, using an MQTT broker in the cloud can reduce energy consumption at terminals (for maintaining the broker-side connection) and allows for increased scalability and redundancy.

The comparative assessment in this study concentrates on the most widely implemented IoT communication protocols, namely MQTT, CoAP, LwM2M, AMQP, XMPP, HTTP/HTTPS, WebSockets, and OPC UA. Figure 3 shows the estimated energy per message.

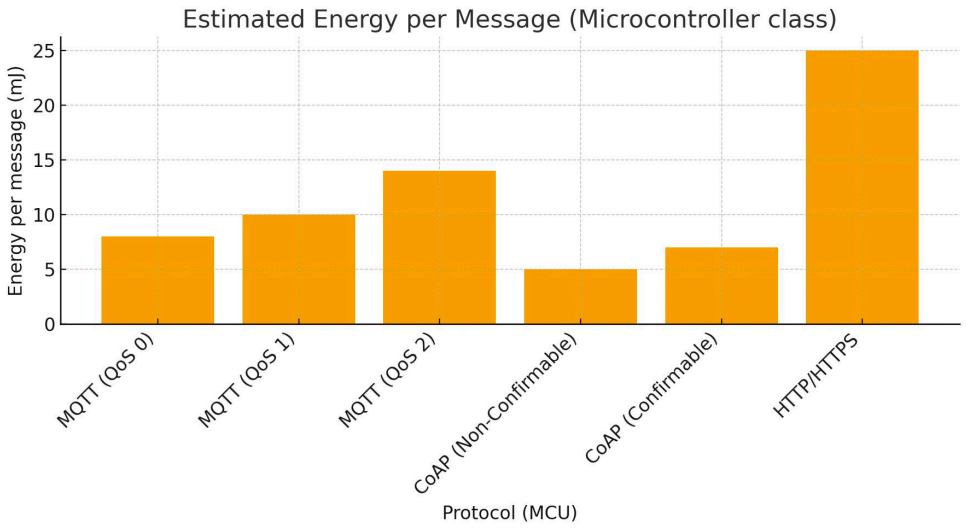


Figure 3. Estimated energy per message.

Recent empirical evaluations [57] indicate that lightweight publish/subscribe protocols, such as MQTT and CoAP, generally outperform traditional request/response paradigms like HTTP when deployed in constrained network environments characterized by limited bandwidth and processing capacity. MQTT achieves high scalability and efficient message dissemination through its broker-based architecture and multi-level Quality of Service (QoS) mechanisms, ensuring reliable data delivery under varying network conditions. Conversely, CoAP enables RESTful communication over UDP with reduced transmission overhead, offering lower latency and enhanced efficiency for resource-limited devices. Building upon CoAP, the Lightweight M2M (LwM2M) protocol integrates standardized device management and monitoring functionalities, rendering it a preferred solution for large-scale cellular IoT deployments where remote configuration and maintenance are essential.

The studies [58,59] evaluate the performance of MQTT and CoAP in a challenging satellite environment characterized by high latency. Its findings are crucial for extending the scope of your analysis beyond terrestrial networks. The performance trade-offs must be considered to highlight how the behavior of protocols changes in delay-tolerant networks (DTNs), a critical consideration for global IoT deployments.

This section moves beyond protocol description to a critical synthesis of the performance, security, and architectural trade-offs inherent in IoT protocol selection, drawing on empirical evidence and current research trends.

4.1. Performance and Energy Efficiency Trade-Offs

The constrained nature of many IoT devices makes performance and energy efficiency paramount. Empirical studies consistently reveal a clear hierarchy influenced primarily by the underlying transport protocol and message semantics.

The TCP vs. UDP Divide: The most significant differentiator is the choice between TCP (e.g., MQTT, HTTP, AMQP) and UDP (e.g., CoAP, MQTT-SN). TCP's connection-oriented nature provides reliability through acknowledgments and retransmissions, but this comes at a steep cost for battery-powered devices. The three-way handshake for connection establishment and the overhead of maintaining a session state consume substantial energy, especially in low-duty-cycle scenarios where devices sleep between transmissions [51].

A Clear Efficiency Hierarchy: Based on empirical evaluations [48,51], a general hierarchy of energy efficiency can be established for common telemetry patterns:

CoAP (Non-Confirmable): Often the most efficient, leveraging UDP's statelessness for fire-and-forget messaging. Its minimal 4-byte header and lack of connection overhead make it ideal for frequent, small updates in loss-tolerant applications.

MQTT-SN: A close contender, specifically designed to minimize overhead over non-TCP networks like Zigbee or LoRaWAN. By using short topic IDs and avoiding complex headers, it achieves efficiency similar to, and in some tests even surpassing, CoAP [51].

MQTT: While lightweight, its reliance on TCP places it a tier below. Its energy consumption is highly dependent on the chosen Quality of Service (QoS). QoS 0 (at most once) is relatively efficient, but QoS 1 (at least once) and especially QoS 2 (exactly once) introduce significant acknowledgment handshakes that increase latency and power draw [31]. However, MQTT's ability to maintain a single long-lived connection for multiple messages can be more efficient than repeated CoAP request/response cycles in some continuous streaming scenarios.

HTTP/HTTPS: Consistently the least efficient for frequent telemetry. Its verbose, text-based headers and the stateless, request/response model that often requires a new TLS handshake per request result in high bandwidth usage and energy consumption [48].

For maximizing battery life in constrained, periodic reporting scenarios, UDP-based protocols (CoAP, MQTT-SN) are superior. MQTT becomes the preferred choice when broker-based mediation, session persistence, and stronger delivery guarantees are required, accepting the inherent overhead of TCP.

4.2. Security Overheads and Lightweight Solutions

Security is non-negotiable in IoT, but traditional mechanisms like TLS (for TCP) and DTLS (for UDP) introduce significant computational and communication overhead that can overwhelm constrained devices.

The Burden of Traditional Security: Implementing TLS/DTLS involves costly public-key cryptography during the handshake phase, demanding considerable CPU cycles and memory. Studies measuring TLS 1.3 on MQTT have recorded execution time and energy consumption increases of 6% to 23% during the authentication and key exchange phases compared to unsecured communication [52]. This overhead directly conflicts with the goal of extended battery life and can be prohibitive for Class 1 constrained devices (as per RFC 7228).

Emerging Lightweight Alternatives: The industry has responded with innovative solutions designed for a constrained world:

OSCORE (Object Security for Constrained RESTful Environments): This approach moves security from the transport layer to the application layer. Instead of securing the entire channel, OSCORE encrypts and authenticates individual CoAP messages (objects) end-to-end. This allows messages to remain secure even when passing through untrusted intermediaries or gateways, without requiring a full TLS/DTLS session with each hop [53].

eeDTLS (Energy-Efficient Datagram Transport Layer Security): This proposal optimizes the DTLS handshake itself. Techniques include aggressive header compression (reducing overhead from 77 bytes to as low as 7 bytes), using certificate URLs instead of transmitting full certificates, and caching security sessions to avoid repeated full handshakes. Banerjee et al. [60] demonstrated that eeDTLS can reduce energy consumption by over 90% for the handshake phase.

While promising, these lightweight solutions face hurdles. OSCORE requires a pre-shared security context and shifts complexity to the application, demanding more sophisticated device management. eeDTLS is not yet a standardized IETF protocol, limiting its widespread implementation in commercial stacks. The primary challenge remains the tension between robust security, which is complex, and the extreme simplicity required by the most constrained devices. Widespread adoption will depend on their integration into major IoT platforms and their proven resilience against real-world attacks.

4.3. Interoperability and the Gateway Problem

IoT ecosystems are rarely homogeneous. The coexistence of diverse protocols, data models, and network technologies (IP vs. non-IP) creates a significant interoperability challenge, for which the gateway has become a ubiquitous, yet problematic, solution.

The Gateway: A Necessary Evil: Gateways are pragmatic tools for protocol translation, e.g., converting a LoRaWAN payload to an MQTT message for the cloud. However, they introduce critical issues:

Security Breakdown: Gateways often terminate security sessions (e.g., LoRaWAN's MAC-layer encryption), converting to a new session (e.g., MQTT over TLS) to the cloud. This breaks end-to-end security and integrity, turning the gateway into a trusted—and vulnerable—man-in-the-middle [28].

Complexity and Single Point of Failure: Each gateway adds configuration, management, and maintenance overhead. It also becomes a potential single point of failure for all devices behind it.

Standardized Frameworks for Cohesion: To mitigate gateway sprawl, industry standards aim to provide unified management and data models:

OMA LwM2M: This standard directly addresses the problem by defining a uniform RESTful interface and a standardized object model for device management and service enablement. Instead of proprietary gateways performing custom translations, LwM2M servers can manage any LwM2M client device consistently, regardless of the underlying network.

oneM2M: This global standard takes a broader view, defining a common service layer that can be embedded across devices, gateways, and cloud platforms. oneM2M provides abstractions that allow different protocols (e.g., MQTT, CoAP) to carry standardized data structures, thereby promoting interoperability at the application level rather than just the transport level [28].

The long-term vision is to minimize ad hoc protocol translation through gateways by maximizing the use of IP-based stacks (like 6LoWPAN/Thread) and standardized application-layer protocols like LwM2M, which natively support end-to-end security models like OSCORE.

4.4. The Cloud–Edge Continuum

Modern IoT architecture distributes intelligence across a continuum from the cloud to the edge, and protocol selection must adapt to this topology.

The Edge Domain: At the extreme edge, within constrained device networks (e.g., a Thread mesh or an 802.15.4 sensor network), CoAP is the native choice. Its RESTful model allows direct device-to-device communication, and its low overhead is ideal for resource-constrained microcontrollers. A lightweight MQTT broker can also be deployed at a local gateway or edge server to efficiently collect data from many devices within a facility, enabling local processing and control without cloud dependency. This reduces latency and bandwidth usage to the cloud.

The Cloud Domain: For integration with web-based cloud services, user dashboards, and enterprise systems, HTTP/HTTPS and WebSockets are the lingua franca. Their ubiquity in web infrastructure makes them the easiest path for ingesting data into cloud platforms like AWS IoT or Azure IoT Hub. WebSockets, in particular, are excellent for maintaining a persistent, full-duplex connection between a cloud application and a gateway for real-time control and streaming.

Bridging the Divide: The most common pattern involves a hybrid approach. Edge gateways act as protocol translators, subscribing to local MQTT topics or receiving CoAP messages from field devices. The gateway then aggregates, filters, and forwards this data to the cloud using a more web-friendly protocol like HTTPS or MQTT over WebSockets. This architecture leverages the strengths of each protocol: efficiency at the edge and seamless integration in the cloud, while also enabling critical offline operation at the edge when the cloud connection is lost [56].

Protocols such as MQTT and CoAP continue to dominate low-power and resource-constrained environments due to their architectural simplicity, lightweight message formats, and operational flexibility. LwM2M extends these capabilities by introducing comprehensive lifecycle management functions, making it particularly suitable for large-scale remote device provisioning and maintenance. In contrast, OPC UA and AMQP are better aligned with the requirements of Industrial IoT (IIoT) applications, where deterministic communication, high reliability, and semantically rich data exchange are critical. Furthermore, emerg-

ing hybrid IoT architectures increasingly employ combinations of these protocols through interoperable gateways and edge computing nodes, enabling optimized trade-offs between performance, scalability, and interoperability across heterogeneous network domains.

4.5. Recommendations for Practical Checklist Choosing Protocols

1. What is the power budget (battery life target)?
2. Expected message frequency and size (tiny single-byte telemetry vs. frequent streams).
3. Range and deployment environment (indoor basement vs. rural field).
4. Is end-to-end IP required? (if yes, prefer 6LoWPAN/Thread or cellular; if no, plan gateway translation).
5. Latency requirements (commands vs. telemetry).
6. Security and provisioning constraints (Do you need mutual auth/FOTA?).
7. Operational model: who manages connectivity (operator for NB-IoT vs. private LoRaWAN)?
8. Interoperability needs: do you need to talk to standard cloud services directly?

5. Practical Deployments of IoT Communication Protocols

This section summarizes real-world implementations of MQTT, CoAP, and HTTP/HTTPS and analyzes the engineering rationale behind their adoption in operational IoT systems. The comparison highlights how protocol selection aligns with device capabilities, network constraints, and application requirements.

5.1. MQTT in Operational IoT Systems

MQTT is widely deployed in large-scale systems that require asynchronous telemetry, low bandwidth consumption, and robust communication over unreliable or mobile networks. Major cloud platforms such as Amazon AWS IoT Core and Bosch IoT Suite rely on MQTT as the primary ingestion protocol for sensor and machine telemetry. Industrial environments (e.g., Siemens MindSphere) integrate MQTT to efficiently aggregate measurements from PLCs, SCADA components, and gateway devices.

In the automotive sector, MQTT-style publish/subscribe communication is employed in vehicle telematics platforms (e.g., Tesla's fleet telemetry), where intermittent 4G/LTE connectivity demands a protocol resilient to connection loss and variable latency. Telecom operators (e.g., Vodafone NB-IoT) also adopt MQTT because its small message size and flexible QoS levels reduce power consumption and support extended battery life in constrained NB-IoT devices. The most well-known applications are as follows:

5.1.1. Smart Home/Consumer IoT

- Philips Hue Bridge (v1/v2) uses MQTT internally for sensor and switch updates.
- Amazon AWS IoT Core officially supports MQTT as its primary protocol.
- Google Cloud IoT Core (until shutdown in 2023) used MQTT as its recommended protocol.

5.1.2. Industrial and SCADA

- Siemens MindSphere uses MQTT for factory telemetry ingestion.
- Bosch IoT Suite uses MQTT for industrial sensors and machine data.
- ABB Ability™ platform uses MQTT for remote asset monitoring.

5.1.3. Automotive/Mobility

- Tesla (fleet telematics) uses MQTT-like lightweight pub/sub for vehicle status and OTA update coordination.
- Volkswagen Industrial Cloud uses MQTT for machine–cloud synchronization.

5.1.4. Telecom

- Vodafone IoT Platform supports MQTT for NB-IoT and LTE-M devices.
- Ericsson IoT Accelerator uses MQTT for device-cloud communication.

5.2. CoAP in Constrained and Low-Power Networks

CoAP has become the dominant protocol for resource-constrained IoT nodes, particularly those operating on battery power, low data-rate networks, or mesh topologies. With a compact 4-byte base header and UDP transport, CoAP minimizes transmission overhead and enables low-latency communication with minimal energy cost.

This efficiency has led to its adoption in smart city infrastructures such as intelligent street lighting systems (e.g., Philips CityTouch) and municipal sensor networks (parking, waste, environmental monitoring). The OMA LwM2M device management standard—widely used in NB-IoT modules from Quectel, u-blox, and Huawei—builds directly on CoAP for remote configuration, firmware updates, and telemetry transfer.

In residential IoT ecosystems, CoAP underpins the communication stack of Thread-based devices and contributes to commissioning procedures in Matter (formerly CHIP). These mesh environments benefit from CoAP's multicast capabilities and minimal packet size, enabling reliable communication in devices with limited CPU and memory resources. The most well-known applications are as follows:

5.2.1. Smart City Systems

- Street lighting systems (e.g., Philips CityTouch) use CoAP for low-power luminaire communication.
- Smart parking sensors in European pilot cities use CoAP because of NB-IoT/LwM2M compatibility.

5.2.2. Lightweight M2M (LwM2M)

CoAP is the transport protocol for the OMA LwM2M device-management standard and is used in

- Quectel NB-IoT modules;
- u-blox NB-IoT devices;
- Huawei OceanConnect IoT Platform;
- ARM Pelion Device Management.

5.2.3. Home Automation

- OpenThread/Google Thread (Mesh networks used in Smart Home) use CoAP messages for commissioning and control.
- Matter (formerly CHIP) uses CoAP-based messaging for local-device commissioning.

5.2.4. Constrained Industrial Devices

- Smart meters in Germany's Smart Metering Infrastructure (SMI) (partially) use CoAP for readings under low-power conditions.

5.3. HTTP/HTTPS in High-Payload and Cloud-Centric Applications

HTTP/HTTPS remains the preferred choice for IoT applications requiring large payload transmission, web service integration, and compatibility with enterprise security mechanisms. Consumer IoT devices that interact heavily with cloud services—such as Ring, Nest, and Wyze cameras—use HTTPS to transmit event metadata, images, and video clips securely. The reliance on TLS and REST interfaces aligns with existing web infrastructure, facilitating easier deployment and interoperability.

Wearable device ecosystems (e.g., Fitbit, Garmin, Apple HealthKit) also utilize HTTPS to synchronize sensor data, apply OAuth2-based authentication, and integrate with cloud analytics services.

Within industrial environments, OPC UA over HTTPS is used by systems such as Siemens SIMATIC and Schneider Electric EcoStruxure to interface with corporate IT platforms while traversing enterprise firewalls. The ubiquity of HTTP(S) and its strong security model make it ideal for device management APIs, firmware delivery, and interactions with cloud-based digital twins.

5.3.1. Consumer IoT

- Smart TVs (Samsung, LG)—all use REST APIs over HTTPS for cloud integration.
- Ring cameras, Nest devices, Wyze, Arlo → HTTPS for event uploads, images, and video control channels.
- IFTTT integrations heavily rely on HTTPS REST.

5.3.2. Industrial IoT

OPC UA over HTTPS is used in the following:

- Siemens SIMATIC systems
- Schneider Electric EcoStruxure
- REST-based APIs used for device management in industrial gateways.

5.3.3. Wearables

- Fitbit, Garmin, Apple HealthKit upload telemetry via HTTPS REST endpoints.

5.3.4. Automotive

Mercedes Me, BMW ConnectedDrive, Volvo OnCall use HTTPS REST APIs for:

- remote lock/unlock
- climate control
- telemetry upload

5.3.5. Cloud Platforms

Every major cloud vendor provides HTTPS REST APIs:

- AWS IoT (Device Shadow)
- Azure IoT Hub (REST-based twin updates)
- Google Firebase IoT integrations

5.4. Summary: Matching Protocols to Deployment Constraints

- MQTT is selected for scalable telemetry, publish/subscribe workflows, and challenging cellular environments.
- CoAP is preferred for low-power, UDP-based, and mesh or NB-IoT deployments, where minimal overhead is essential.
- HTTP/HTTPS dominates systems with large data transfers, cloud integration, and stringent enterprise IT requirements.

These real-world deployments demonstrate that no single universal protocol exists; instead, each protocol serves a distinct operational niche. Consequently, future IoT architectures increasingly rely on adaptive, context-aware protocol selection to optimize performance across heterogeneous device classes and network conditions. Table 4 summarizes the real-world protocol applications.

Table 4. Summarizes of the real word protocols application.

Protocol	Real-World Implementation/Product	Industry	Why This Protocol Was Chosen (Technical Reasoning)
MQTT	AWS IoT Core	Cloud IoT	Designed for millions of devices needing scalable pub/sub messaging; low overhead; supports QoS.
MQTT	Bosch IoT Suite	Industrial IoT	Efficient telemetry from machines; reliable delivery via QoS 1/2; async event model.
MQTT	Siemens MindSphere	Industrial automation	Low network load, good for PLC/SCADA integration and aggregated sensor streams.
MQTT	Tesla fleet telematics (internal)	Automotive	Lightweight binary protocol works well over mobile networks with unstable connectivity.
MQTT	Vodafone NB-IoT Device Platform	Telecom	NB-IoT requires minimal overhead and long battery life; MQTT fits constraints.
CoAP	Philips CityTouch smart street lighting	Smart City	UDP-based low latency; extremely low overhead (4-byte header); ideal for constrained nodes.
CoAP	OMA LwM2M (Quectel, u-blox NB-IoT modules)	Telecom/NB-IoT	CoAP enables device management, firmware updates, and resource access on small MCUs.
CoAP	Thread/Google Nest ecosystem	Smart Home Mesh	Mesh networks require tiny packets and fast local delivery; CoAP works over IEEE 802.15.4.
CoAP	Matter/CHIP (commissioning)	Smart Home	Secure onboarding + low power; CoAP + DTLS/OSCORE ideal for constrained device commissioning.
HTTP/HTTPS	Ring, Nest, Wyze cameras	Consumer IoT	HTTPS supports large payloads (images, video events) and integrates with web infrastructure.
HTTPS	BMW ConnectedDrive, Volvo OnCall	Automotive	REST APIs fit cloud services; HTTPS ensures end-to-end encryption and easy backend integration.
HTTPS	Fitbit, Garmin, Apple HealthKit	Wearables	Cloud synchronization uses RESTful APIs; HTTPS simplifies authentication (OAuth2).
HTTPS	OPC UA over HTTP(S) (Siemens, Schneider)	Industrial	HTTP(S) allows enterprise firewall traversal and integration with IT systems.

5.5. Comparison: Why Each Protocol Was Chosen in Practice

Below is an engineering-oriented justification for MQTT, CoAP, and HTTP/HTTPS in real deployments.

5.5.1. MQTT Key Reasons

- Extremely low bandwidth usage
- Publish/subscribe model fits asynchronous telemetry
- Supports QoS levels (0,1,2)
- Works well on unreliable networks (cellular, NB-IoT, satellite)
- Very small client code footprint

Type of Systems Chosen by MQTT

- Telemetry-heavy
- Many-to-one aggregation
- Devices with intermittent connectivity

- Industrial gateways

Example Justification

Tesla and AWS IoT have millions of sensors and continuous telemetry and MQTT's pub/sub avoids HTTP overhead and supports reliable delivery even over unstable networks.

5.5.2. CoAP Key Reasons

- Ultra-low overhead (4-byte header vs. 200–800 bytes for HTTP/TLS)
- Runs on UDP (faster, lower latency, smaller packets)
- Optimized for constrained devices (Class-0/Class-1 MCUs)
- Native support for multicast (unique among IoT protocols)
- Basis of LwM2M (standard for NB-IoT module management)
- Works over Thread and Matter ecosystems

Type of Systems Chosen by CoAP

- Battery-powered sensors
- Smart city nodes (lighting, parking, waste bins)
- Smart home mesh networks
- NB-IoT device management
- Commissioning and local control

Example Justification

Philips CityTouch uses CoAP because streetlamps have tiny microcontrollers and need minimal overhead; CoAP's 4-byte header drastically reduces energy usage.

5.5.3. HTTP/HTTPS Key Reasons

- Universally supported (browsers, cloud services, firewalls)
- Ideal for large payloads (images, JSON, firmware)
- Easy integration with REST APIs and OAuth2 authentication
- Required for many enterprise/industrial IT systems
- End-to-end encryption with TLS

Type of Systems Chosen by HTTP/HTTPS

- Video doorbells, cameras
- Wearables syncing to cloud
- Car OEM cloud platforms
- Industrial OPC UA over HTTPS
- Any application requiring large or structured data (JSON, video, images)

Example Justification

Ring/Nest cameras send media files to the cloud and use HTTPS because it handles large payloads reliably and integrates with web services.

6. Conclusions

Transport and application protocols form the backbone of IoT communication, determining how devices exchange data, maintain reliability, and interact with cloud and enterprise systems. Each protocol offers a distinct combination of efficiency, scalability, security, and suitability for specific deployment scenarios. IoT application protocols face several challenges. These challenges are related to the drawbacks of application protocols. Those challenges can be summarized in the following points: not suitable for real-time and

industrial application, not suitable for constrained devices, and lack of interoperability, security mechanisms, and Quality of Service (QoS) [6].

A further energy saving can be achieved by UDP/IP, and DTLS header compression for packets exchanged between the IoT device and the gateway, e.g., as the eeDTLS protocol proposed by Banerjee et al. [60]. eeDTLS reduces protocol overheads by 91%, from 77 bytes to 7 bytes. eeDTLS also allows a client to send certificate URLs in place of certificates according to RFC 6066. In consequence, eeDTLS can reduce energy consumption by DTLS handshake computations [24].

Lightweight protocols such as MQTT, MQTT-SN, CoAP, and LwM2M are ideal for constrained devices, low-power networks, and scenarios requiring efficient telemetry or device management.

Enterprise-grade protocols like AMQP and OPC UA provide robust messaging, transactional reliability, and semantic interoperability, making them well-suited for industrial and large-scale deployments.

Web-focused protocols such as WebSockets and HTTP/HTTPS enable real-time integration with web dashboards, mobile applications, and cloud platforms, supporting devices with higher resources.

Extensible protocols like XMPP allow for flexible, presence-aware messaging, useful in interactive IoT environments where custom messaging or federation is required.

Selecting the appropriate protocol requires careful consideration of network constraints, device capabilities, security requirements, and application needs. In many deployments, hybrid approaches combining multiple protocols are common—for example, using CoAP or MQTT on constrained devices while integrating with WebSockets or HTTP for cloud dashboards.

By understanding the strengths, limitations, and use cases of each protocol, system architects can design IoT solutions that are scalable, resilient, secure, and efficient, ensuring long-term operational success across diverse environments—from home automation to industrial IoT.

Protocols like MQTT and CoAP excel in device-to-cloud data flow, while LwM2M addresses the critical vertical of device management, and OPC UA solves the problem of semantic interoperability in industrial settings.

Future research in the field of IoT communication protocols is expected to concentrate on enhancing interoperability across heterogeneous protocol stacks, enabling seamless data exchange among various devices and platforms. A major direction involves the integration of IoT systems with emerging 6G networks, which promise ultra-reliable low-latency communication (URLLC) and deterministic networking capabilities essential for mission-critical applications. Another promising avenue is the development of AI-driven adaptive protocol selection mechanisms, capable of dynamically optimizing communication efficiency based on network conditions and application requirements. Furthermore, digital twin architectures, particularly those combining OPC UA over MQTT, are increasingly recognized as key enablers for real-time industrial monitoring and predictive maintenance. In parallel, the advent of post-quantum cryptography (PQC) is anticipated to fundamentally transform IoT security paradigms, providing robust protection against quantum-era cyber threats and ensuring long-term data integrity and confidentiality in connected environments.

Security is a major concern in future-proof IoT deployments. PQC algorithms promise resilience against quantum attacks but introduce significant overhead in constrained environments. TLS/DTLS already imposes computational and communication costs that can limit throughput and increase latency. PQC's larger key sizes and higher computational demands exacerbate these issues, creating a critical need for lightweight security approaches, such as OSCORE, that maintain confidentiality and integrity without

overwhelming resource-limited devices. Research should focus on optimizing PQC implementations for constrained networks while balancing latency, energy consumption, and security.

Emerging applications in industrial automation, smart grids, and autonomous systems demand deterministic communication, where latency and reliability must be guaranteed. Existing protocols like MQTT and CoAP are not designed to provide such deterministic guarantees. They face challenges in real-time delivery under network congestion and are unable to satisfy strict temporal constraints. The integration of 6G networks and URLLC (Ultra-Reliable Low-Latency Communication) can provide the necessary network infrastructure for low-latency and highly reliable transmission; however, without protocol-level support, deterministic behavior cannot be ensured. This motivates research into protocols and middleware that inherently support predictable latency, reliability, and interoperability for real-time IoT applications.

The heterogeneity of IoT devices, network conditions, and application requirements implies that no single protocol can satisfy all scenarios. AI-driven adaptive mechanisms offer a promising solution, dynamically selecting or configuring communication protocols based on device capability, network state, and application demands. Machine learning models can optimize trade-offs among latency, reliability, throughput, and energy efficiency, enabling seamless adaptation to dynamic environments. By integrating AI with advanced network technologies such as 6G and URLLC, adaptive mechanisms can intelligently exploit available network guarantees, ensuring that mission-critical IoT applications meet their stringent requirements.

This approach acknowledges the absence of a universal solution and leverages intelligent adaptation to maximize system performance across diverse IoT deployments.

Author Contributions: Conceptualization, I.P. and L.M.U.; methodology, I.P.; investigation, I.P. and V.V.; writing—original draft preparation, I.P., A.D., and E.N.; writing—review and editing, I.P. and V.V.; supervision, I.P. and L.M.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National University of Science and Technology POLITEHNICA Bucharest.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study is available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Fatma, H.; Sofiane, O. A review of application protocol enhancements for IoT. In Proceedings of the Fifteenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2021), Barcelona, Spain, 7 October 2021; pp. 7–13.
2. Syed, A.; Sierra-Sosa, D.; Kumar, A.; Elmaghraby, A. Iot in smart cities: A survey of technologies, practices and challenges. *Smart Cities* **2021**, *4*, 429–475. [CrossRef]
3. Khaleefah, R.M.; Abed, A.A.; Al-Shareeda, M.A. Empirical Evaluation of MQTT, CoAP and HTTP for Smart City IoT Applications. *Int. J. Mechatron. Robot. Artif. Intell.* **2025**, *1*, 74–81. [CrossRef]
4. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
5. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
6. Hmissi, F.; Ouni, S. A Survey on Application Layer Protocols for IoT Networks. *Int. J. Adv. Telecommun.* **2022**, *15*, 11–22.

7. Bormann, C.; Ersue, M.; Keranen, A. *RFC 7228: Terminology for Constrained-Node Networks*; IETF: Fremont, CA, USA, 2014.
8. OASIS. MQTT Version 3.1.1. 2014. Available online: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (accessed on 9 September 2025).
9. OASIS. MQTT Version 5.0. 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed on 9 September 2025).
10. HiveMQ. MQTT-SN: What Is It and Why Does It Matter? Available online: https://www.hivemq.com/blog (accessed on 9 September 2025).
11. IETF. RFC 7252: The Constrained Application Protocol (CoAP). 2014. Available online: https://datatracker.ietf.org/doc/html/rfc7252 (accessed on 9 September 2025).
12. CoAP.space. CoAP Overview. Available online: https://coap.space/ (accessed on 9 September 2025).
13. Open Mobile Alliance. Lightweight Machine to Machine (LwM2M) Specifications. Available online: https://www.openmobilealliance.org/release/LightweightM2M/ (accessed on 9 September 2025).
14. AVSystem. Lightweight M2M (LwM2M) Overview. Available online: https://avsystem.com/blog/iot/lightweight-m2m-lwm2m-overview (accessed on 9 September 2025).
15. OASIS. AMQP 1.0 Core Specification. 2011. Available online: https://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html (accessed on 9 September 2025).
16. RabbitMQ. AMQP 0-9-1 Protocol Specification. Available online: https://www.rabbitmq.com/amqp-0-9-1-protocol.html (accessed on 10 September 2025).
17. IETF. RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core. 2011. Available online: https://datatracker.ietf.org/doc/html/rfc6120 (accessed on 10 September 2025).
18. Thangavel, D.; Ma, X.; Valera, A.; Tan, H.X.; Tan, C.K.Y. Performance Evaluation of MQTT and CoAP via a Common Middleware. In Proceedings of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 21–24 April 2014; pp. 1–6.
19. Luzuriaga, J.E.; Perez, M.; Boronat, P.; Cano, J.C.; Calafate, C.; Manzoni, P. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2015; pp. 931–936.
20. XMPP.org. XMPP Extensions. Available online: https://xmpp.org/extensions/ (accessed on 10 September 2025).
21. WHATWG. WebSockets Standard. Available online: https://websockets.spec.whatwg.org/ (accessed on 10 September 2025).
22. Al-Fuqaha, A.; Khreishah, A.; Guibene, W.; Othman, J.B. An Overview of Existing Approaches and Challenges in Securing IoT. In Proceedings of the 2016 IEEE International Conference on Smart Grid Communications (SmartGridComm), Sydney, Australia, 6–9 November 2016; pp. 1–6.
23. Al-Fuqaha, A.; Khreishah, A. Securing IoT Communications: Challenges and Approaches. In Proceedings of the 2016 IEEE International Conference on Smart Grid Communications (SmartGridComm), Sydney, Australia, 6–9 November 2016; pp. 1–6.
24. Goworko, M.; Wytrebowicz, J. A Secure Communication System for Constrained IoT Devices—Experiences and Recommendations. *Sensors* **2021**, *21*, 6906. [CrossRef] [PubMed]
25. IETF. RFC 6455: The WebSocket Protocol. 2011. Available online: https://datatracker.ietf.org/doc/html/rfc6455 (accessed on 10 September 2025).
26. IETF. RFC 2616: Hypertext Transfer Protocol—HTTP/1.1. 1999. Available online: https://datatracker.ietf.org/doc/html/rfc2616 (accessed on 10 September 2025).
27. Cirani, S.; Picone, M.; Gonizzi, P.; Veltri, L.; Ferrari, G. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sens. J.* **2014**, *15*, 1224–1234. [CrossRef]
28. Aloi, G.; Caliciuri, G.; Fortino, G.; Gravina, R.; Pace, P.; Russo, W.; Savaglio, C. A Mobile Multi-Technology Gateway to Enable IoT Interoperability. In Proceedings of the 2017 IEEE First International Conference on Internet of Things and Systems (ICIOTS), Tiraspol, Moldova, 5–7 July 2017; pp. 1–6.
29. AWS. The Difference Between HTTPS and HTTP. Available online: https://aws.amazon.com/compare/the-difference-between-https-and-http/ (accessed on 10 September 2025).
30. OPC Foundation. OPC UA Specifications. Available online: https://reference.opcfoundation.org/ (accessed on 11 September 2025).
31. EMQ. OPC UA Protocol Overview. Available online: https://www.emqx.com/en/blog/opc-ua-protocol (accessed on 11 September 2025).
32. Chacon, G.S.; Venegas, C.; Baca, M.; Rodriguez, I.; Marrone, L. Open middleware proposal for iot focused on industry 4.0. In Proceedings of the 2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), Atlántico, Colombia, 3 November 2018; pp. 1–6.
33. Nuratch, S. Applying the MQTT protocol on embedded system for smart sensors/actuators and iot applications. In Proceedings of the 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Chiang Rai, Thailand, 18–21 July 2018. no. 1.

34. Esfahani, A.; Mantas, G.; Matischek, R.; Saghezchi, F.B.; Rodriguez, J.; Bicaku, A.; Maksuti, S.; Tauber, M.G.; Schmittner, C.; Bastos, J. A lightweight authentication mechanism for m2m communications in industrial iot environment. *IEEE Internet Things J.* **2017**, *6*, 288–296. [CrossRef]
35. Sheng, Z.; Yang, S.; Yu, Y.; Vasilakos, A.V.; Mccann, J.A.; Leung, K.K. A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wirel. Commun.* **2013**, *20*, 91–98. [CrossRef]
36. Karagiannis, V.; Chatzimisios, P.; Vazquez-Gallego, F.; Alonso-Zarate, J. A Survey on application layer protocols for the internet of things. *J. Comput. Netw. Commun.* **2015**, *2015*, 295734.
37. D'Ortona, C.; Tarchi, D.; Raffaelli, C. Open-source MQTT-based End-to-End IoT system for smart city. *Scenar. J. Futur.* **2022**, *14*, 57. [CrossRef]
38. Panagou, I.C.; Katsoulis, S.; Nannos, E.; Zantalis, F.; Koulouras, G. A comprehensive evaluation of IoT cloud platforms: A feature-driven review with a decision-making tool. *Sensors* **2025**, *25*, 5124. [CrossRef]
39. Roccatello, E.; Pagano, A.; Levorato, N.; Rumor, M. State of the art in internet of things standards and protocols for precision agriculture with an approach to semantic interoperability. *J. Netw.* **2025**, *5*, 14. [CrossRef]
40. Sallabi, F.M.; Khater, H.M.; Tariq, A.; Hayajneh, M.; Shuaib, K.; Barka, E.S. Smart healthcare network management: A comprehensive review. *Mathematics* **2025**, *13*, 988. [CrossRef]
41. Hunkeler, U.; Truong, H. MQTT-S—A Publish/Subscribe Protocol for Wireless Sensor Networks. In Proceedings of the 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08), Bangalore, India, 6–10 January 2008.
42. Cirani, S.; Davoli, L.; Ferrari, G.; Léone, R.; Medagliani, P.; Picone, M.; Veltri, L. A Scalable and self-configuring architecture for service discovery in the Internet of Things. *IEEE IoT J.* **2014**, *1*, 508–521. [CrossRef]
43. Light, R.; Thompson, M. *The MQTT Protocol: A Technical Overview*; IBM Device: Armonk, NY, USA, 2016.
44. Shelby, Z.; Hartke, K.; Bormann, C. *The Constrained Application Protocol (CoAP). RFC 7252*; IETF: Fremont, CA, USA, 2014.
45. Hartke, K. *Observing Resources in CoAP. RFC 7641*; IETF: Fremont, CA, USA, 2015.
46. Fielding, R.; Reschke, J. *Hypertext Transfer Protocol (HTTP/1.1). RFC 7231*; IETF: Fremont, CA, USA, 2014.
47. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446*; IETF: Fremont, CA, USA, 2018.
48. Ochoa, H.J.J.; Peña, R.; Mezquita, Y.L.; Gonzalez, E.; Camacho-Leon, S. Comparative analysis of power consumption between MQTT and HTTP protocols in an IoT platform designed and implemented for remote real-time monitoring of long-term cold chain transport operations. *Sensors* **2023**, *23*, 4896. [CrossRef]
49. Dauda, A.; Flauzac, O.; Nolot, F. A Survey on IoT Application Architectures. *Sensors* **2024**, *24*, 5320. [CrossRef] [PubMed]
50. Ayoub, I.; Balakrichenan, S.; Khawam, K.; Ampeau, B. DNS for IoT: A Survey. *Sensors* **2023**, *23*, 4473. [CrossRef] [PubMed]
51. Jerzy, K.; Martyna, W.K.; Ilona, J.G.; Piotr, K.; Aleksandra, P.; Robert, W.; Teresa, S.-W. Energy Footprint and Reliability of IoT Communication Protocols for Remote Sensor Networks. *Sensors* **2025**, *25*, 6042. [CrossRef]
52. Gavriilidis, N.O.; Halkidis, S.T.; Petridou, S. Empirical Evaluation of TLS-Enhanced MQTT on IoT Devices for V2X Use Cases. *Appl. Sci.* **2025**, *15*, 8398. [CrossRef]
53. Łeska, S.; Furtak, J. Procedures for Building a Secure Environment in IoT Networks Using the LoRa Interface. *Sensors* **2025**, *25*, 3881. [CrossRef]
54. Alotaibi, A.; Aldawghan, H.; Aljughaiman, A. A Review of the Authentication Techniques for Internet of Things Devices in Smart Cities: Opportunities, Challenges, and Future Directions. *Sensors* **2025**, *25*, 1649. [CrossRef]
55. Al-Turjman, F.; Abujubbeh, M. IoT-enabled smart grid via SM: An overview. *Future Gener. Comput. Syst.* **2019**, *96*, 579–590. [CrossRef]
56. Ficili, I.; Giacobbe, M.; Tricomi, G.; Puliafito, A. From Sensors to Data Intelligence: Leveraging IoT, Cloud, and Edge Computing with AI. *Sensors* **2025**, *25*, 1763. [CrossRef] [PubMed]
57. Naik, N. Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE International Symposium on Systems Engineering (ISSE), Vienna, Austria, 11–13 October 2017; IEEE: New York, NY, USA, 2017; pp. 1–7.
58. Dizdarević, J.; Carpio, F.; Jukan, A.; Masip-Bruin, X. A survey of communication protocols for internet of things and their challenges. *Sensors* **2019**, *19*, 948.

59. Al-Masri, E.; Kalyanam, K.R.; Batts, J.; Kim, J.; Singh, S.; Vo, T.; Yan, C. Investigating Messaging Protocols for the Internet of Things (IoT). *IEEE Access* **2020**, *8*, 94880–94911. [CrossRef]
60. Banerjee, U.; Juvekar, C.; Fuller, S.H.; Chandrakasan, A.P. EeDTLS: Energy-efficient datagram transport layer security for the internet of things. In Proceedings of the 2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, 4–8 December 2017; pp. 1–6.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.