People
Process
The four pillars of IoE:
Data
Things

Internet of Everything DJS22CEC702

By Prof. Nilesh Ghavate

Module 5: IoT Communication Protocols

- IoT Protocols : MQTT, XMPP, DDS, AMQP, COAP, REST, IPv6, 6LoWPAN
- IoT Routing Protocols
- Data-centric and Flat-Architecture Protocols
- Flooding, Gossiping
- Sensor Protocols for Information via Negotiation (SPIN), SPIN PP (Push-Pull), SPIN EC (Energy Conserve), SPIN BC (Broadcast),SPIN RL(Reliable)
- LEACH Protocol.

IoT Communication Protocols

IoT Protocols- MQTT, XMPP, DDS, AMQP, COAP, REST, IPv6, 6LoWPAN.

IoT Routing Protocols, Data-centric and Flat-Architecture Protocols,

Flooding, Gossiping,

Sensor Protocols for Information via Negotiation (SPIN), SPIN PP, SPIN EC (Energy Conserve), SPIN BC (Broadcast),SPIN RL(Reliable),

LEACH Protocol

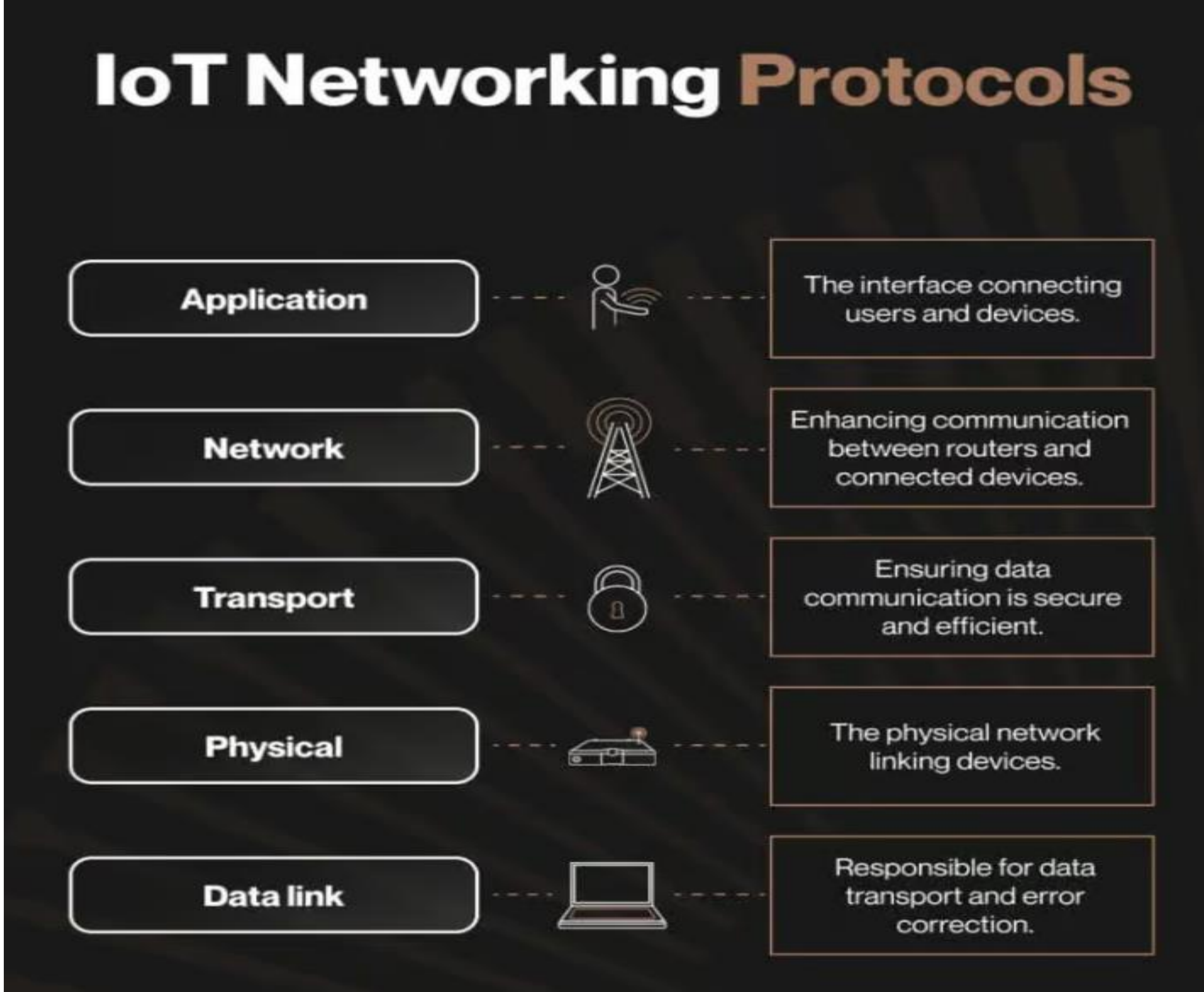
IOT Protocols

- IoT (Internet of Things) protocols are communication standards that enable devices, sensors, and servers to exchange data efficiently.
- They define how data is formatted, transmitted, and received between connected IoT components.
- IoT involves a wide range of devices — from tiny sensors to cloud servers — so different protocols are used at different layers (device, network, application).
- These protocols ensure interoperability, reliability, security, and scalability in IoT systems.
- They are chosen based on factors such as power consumption, bandwidth, latency, range, and device capability.

IOT Protocols

- IoT protocols are typically divided into three main categories:
- Application Layer Protocols – e.g., MQTT, CoAP, HTTP, AMQP, DDS,XMPP
- Network/Transport Layer Protocols – e.g., TCP/IP, UDP, 6LoWPAN (**IPv6 over Low Power Wireless Personal Area Networks)**, RPL (**Routing Protocol for Low-Power and Lossy Networks**)
- Physical/Link Layer (Wireless Communication) – e.g., Wi-Fi, Zigbee, BLE (**Bluetooth Low Energy)** , LoRaWAN (Long Range Wide Area Network), NB-IoT (**Narrowband Internet of Things)**

IOT Networking Protocols



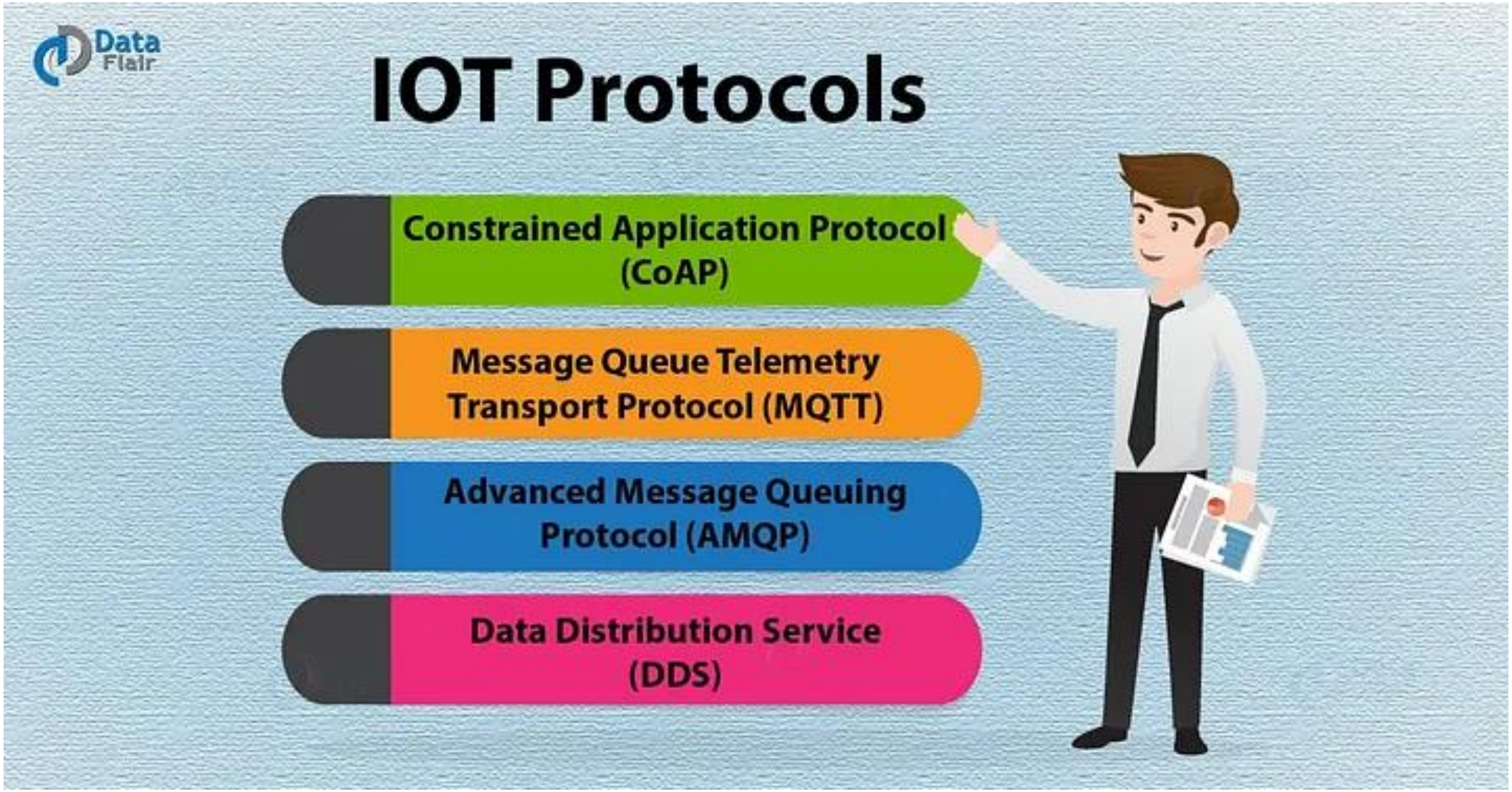
The interface connecting users and devices.

Enhancing communication between routers and connected devices.

Ensuring data communication is secure and efficient.

The physical network linking devices.

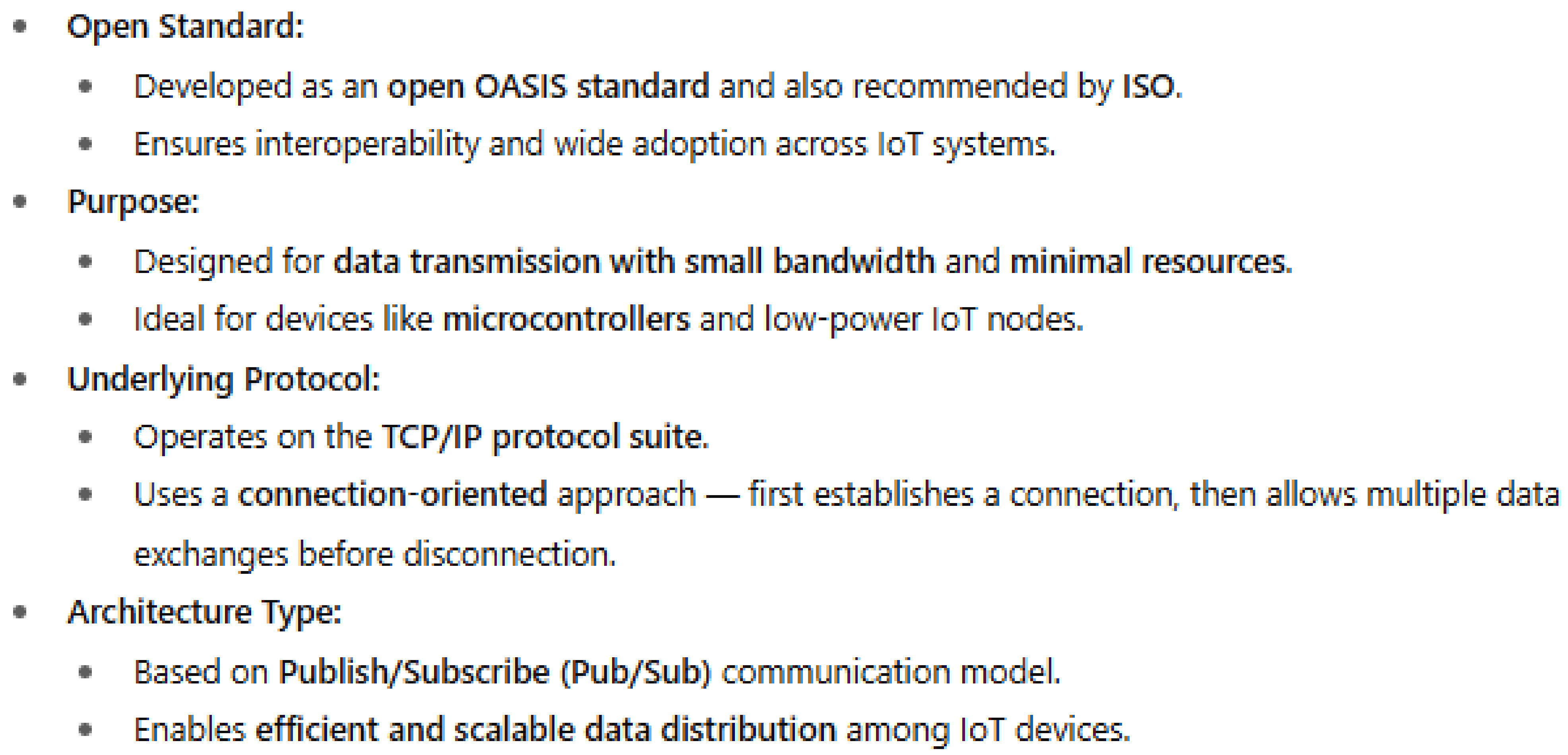
Responsible for data transport and error correction.



IOT Protocols

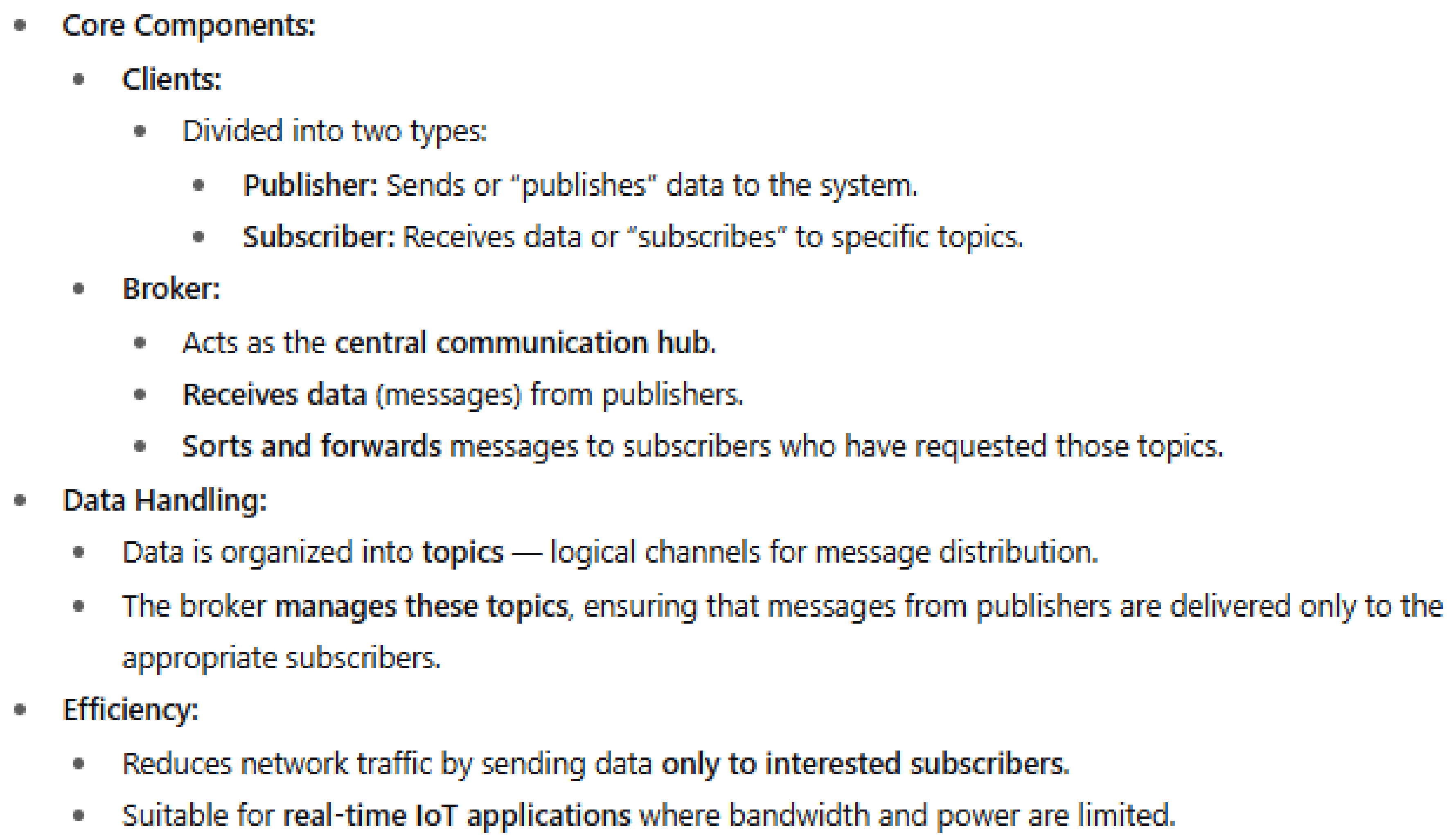
- Constrained Application Protocol (CoAP)
- Message Queue Telemetry Transport Protocol (MQTT)
- Advanced Message Queuing Protocol (AMQP)
- Data Distribution Service (DDS)

MQTT (Message Queuing Telemetry Transport)



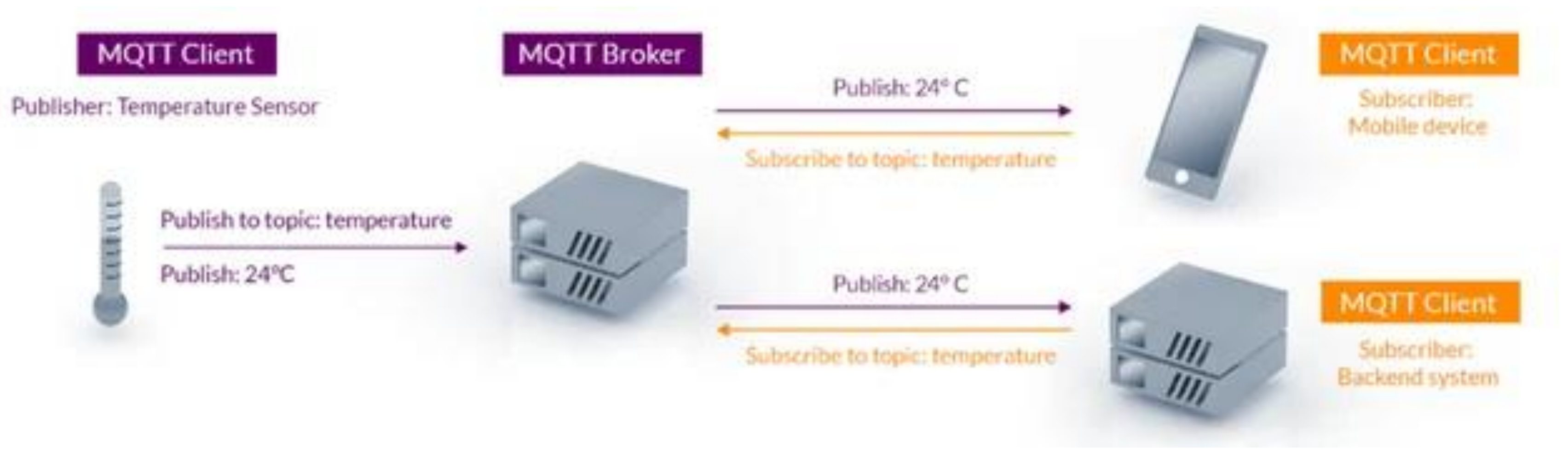
- **Open Standard:**
  - Developed as an **open OASIS standard** and also recommended by **ISO**.
  - Ensures interoperability and wide adoption across IoT systems.
- **Purpose:**
  - Designed for **data transmission with small bandwidth** and **minimal resources.**
  - Ideal for devices like **microcontrollers** and low-power IoT nodes.
- **Underlying Protocol:**
  - Operates on the **TCP/IP protocol suite.**
  - Uses a **connection-oriented** approach — first establishes a connection, then allows multiple data exchanges before disconnection.
- **Architecture Type:**
  - Based on **Publish/Subscribe (Pub/Sub)** communication model.
  - Enables **efficient and scalable data distribution** among IoT devices.

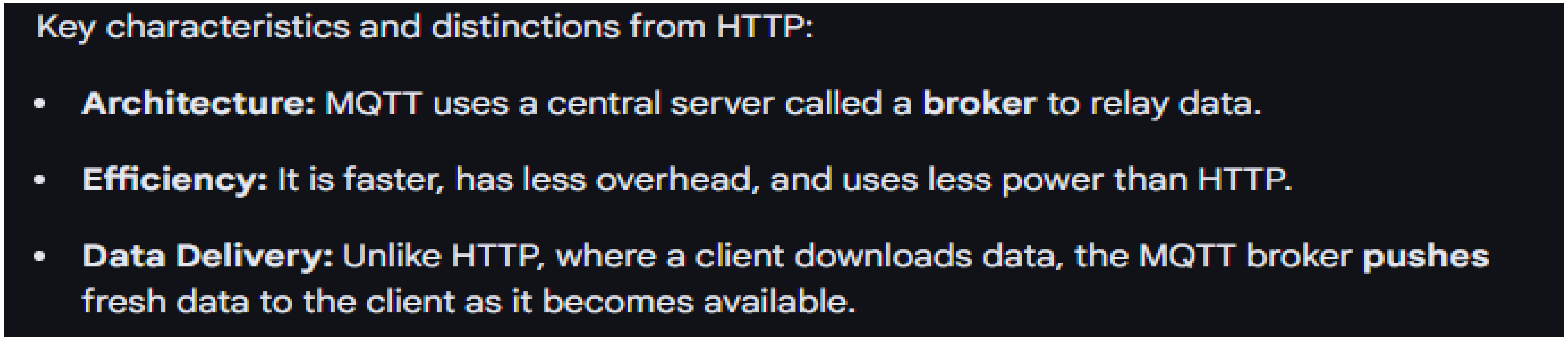
MQTT (Message Queuing Telemetry Transport)



- **Core Components:**
  - **Clients:**
    - Divided into two types:
      - **Publisher:** Sends or "publishes" data to the system.
      - **Subscriber:** Receives data or "subscribes" to specific topics.
  - **Broker:**
    - Acts as the **central communication hub.**
    - **Receives data** (messages) from publishers.
    - **Sorts and forwards** messages to subscribers who have requested those topics.
- **Data Handling:**
  - Data is organized into **topics** — logical channels for message distribution.
  - The broker **manages these topics**, ensuring that messages from publishers are delivered only to the appropriate subscribers.
- **Efficiency:**
  - Reduces network traffic by sending data **only to interested subscribers.**
  - Suitable for **real-time IoT applications** where bandwidth and power are limited.

MQTT (Message Queuing Telemetry Transport)


MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Backend system


Key characteristics and distinctions from HTTP:

- **Architecture:** MQTT uses a central server called a **broker** to relay data.
- **Efficiency:** It is faster, has less overhead, and uses less power than HTTP.
- **Data Delivery:** Unlike HTTP, where a client downloads data, the MQTT broker **pushes** fresh data to the client as it becomes available.

MQTT (Message Queuing Telemetry Transport)

To expand on the characteristics and distinctions of MQTT from HTTP, here are some additional points:

Easy Scalability: New devices can be easily added to an MQTT system without requiring changes to the existing infrastructure. Devices do not need to be compatible with each other, as they only need to communicate with the broker.

Broker-Based Communication: MQTT connects IoT devices through a central broker. There is no direct connection between clients; instead, the broker transmits messages between them.

One-to-Many and Many-to-Many Communication: Unlike some other protocols, MQTT's message relay is not limited to a one-to-one model. It uses topics for identification, allowing multiple devices to subscribe to the same topic (one-to-many) or several devices to post to the same topic with one or more subscribers (many-to-many). The broker handles all these communication paradigms.

Bi-directional Communication: MQTT is a bi-directional protocol that allows clients to both produce and consume data by publishing messages and subscribing to topics. This is a significant benefit for IoT devices, enabling them to send sensor data and receive configuration information and control commands. HTTP, on the other hand, is based on a request-response model, where the server only responds to requests initiated by a client

MQTT (Message Queuing Telemetry Transport)

Security: Both protocols can be secured with TLS/SSL encryption. MQTT also supports authentication methods like usernames, passwords, and client certificate

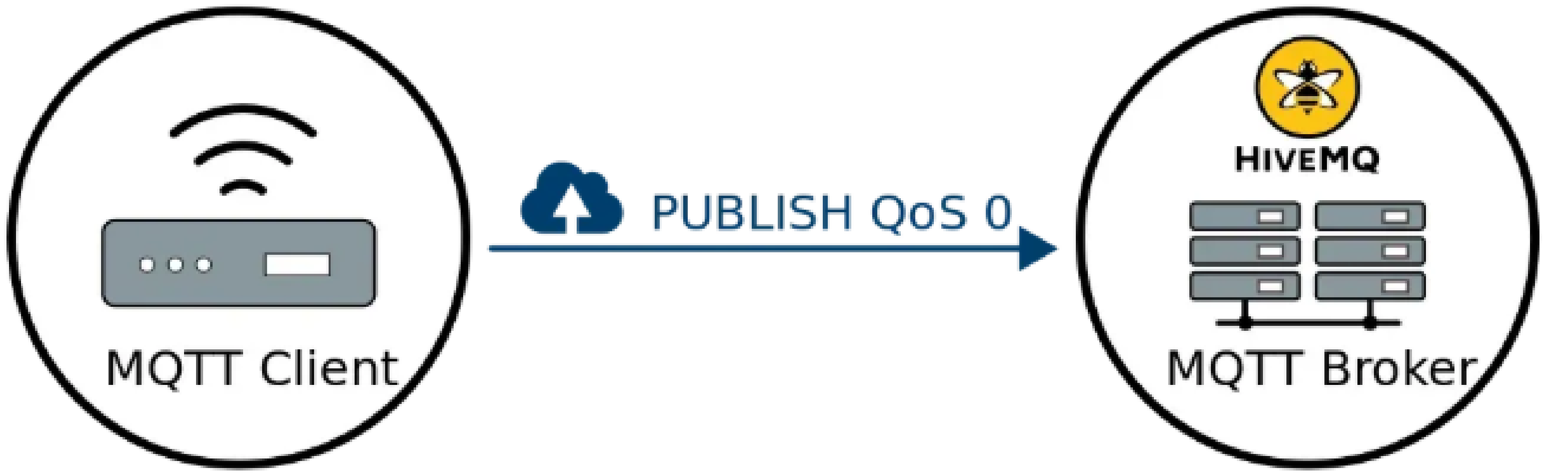
Connection and State: MQTT maintains a persistent connection with a broker, which allows it to handle unreliable networks and reduce reconnection time. HTTP is a stateless protocol, with a new connection typically established for each request and response cycle, which can increase overhead.

Quality of Service (QoS): MQTT has three defined Quality of Service levels to ensure reliable message delivery: "at most once" (QoS 0), "at least once" (QoS 1), and "exactly once" (QoS 2). HTTP does not have this built-in feature, requiring applications to implement their own reliability mechanisms.

QoS	Model	Reliability	Description
0	Fire and Forget	Unreliable	Message is delivered without duplication; no acknowledgment is required.
1	At Least Once	Reliable	Message is delivered at least once with possible duplications; acknowledgment is required (PUBACK).
2	Exactly Once	Reliable without duplications	Message is delivered exactly once by using a four-step handshake (PUBLISH, PUBREC, PUBREL, PUBCOMP) without duplication.

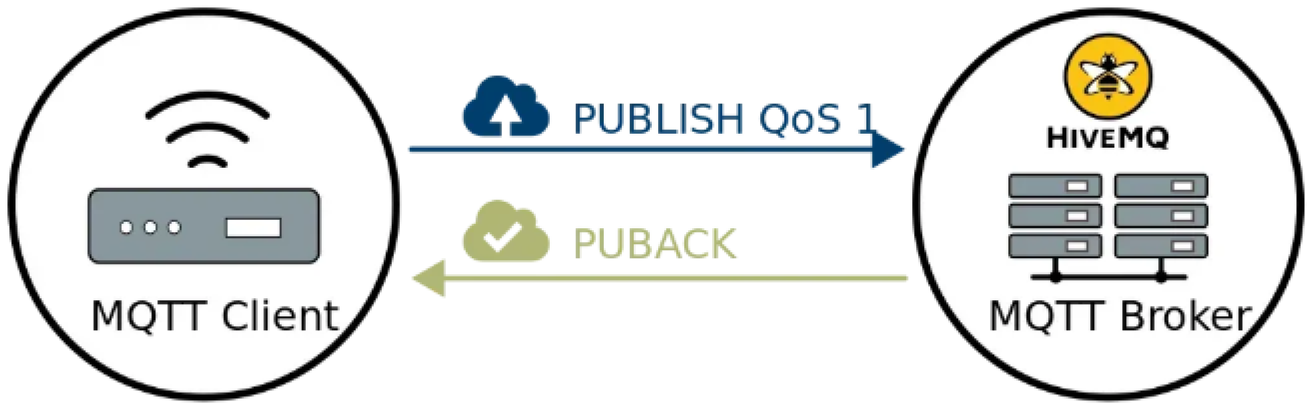
How does QoS 0 work in MQTT?

At the lowest level, QoS 0 in MQTT offers a best-effort delivery mechanism where the sender does not expect an acknowledgment or guarantee of message delivery. This means that the recipient does not acknowledge receiving the message, and the sender does not store or re-transmit it. QoS 0, commonly called “fire and forget,” functions akin to the underlying TCP protocol, where the message is sent without further follow-up or confirmation.



How does QoS 1 work in MQTT?

In QoS 1 of MQTT, the focus is on ensuring message delivery at least once to the receiver. When a message is published with QoS 1, the sender keeps a copy of the message until it receives a **PUBACK** packet from the receiver, confirming the successful receipt. If the sender doesn’t receive the **PUBACK** packet within a reasonable time frame, it re-transmits the message to ensure its delivery.

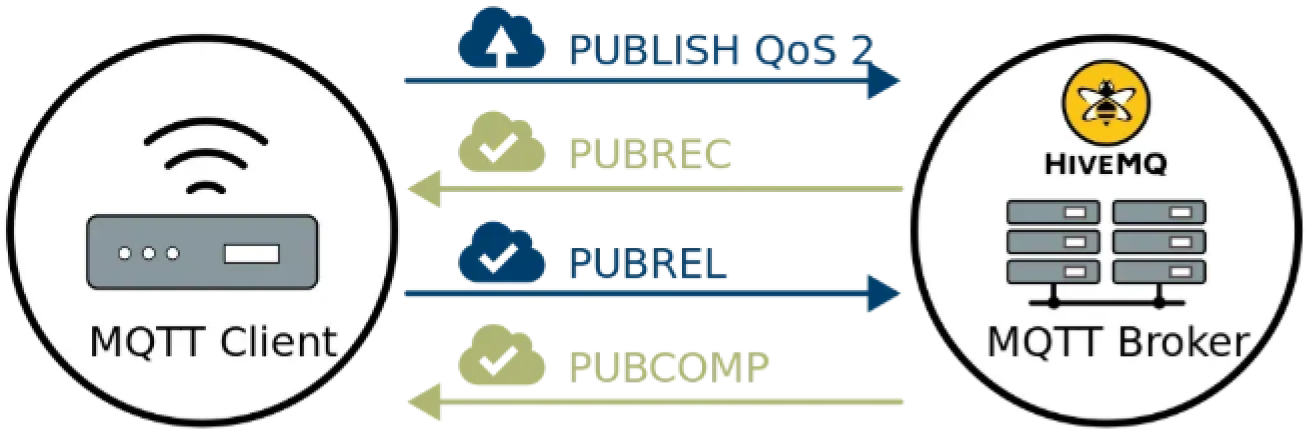


Quality of Service level 1: delivery at least once

Upon receiving the message, the receiver can process it immediately. For example, if the receiver is an MQTT broker, it distributes the message to all subscribing clients and responds with a PUBACK packet to acknowledge the receipt of the message.

How does QoS 2 work in MQTT?

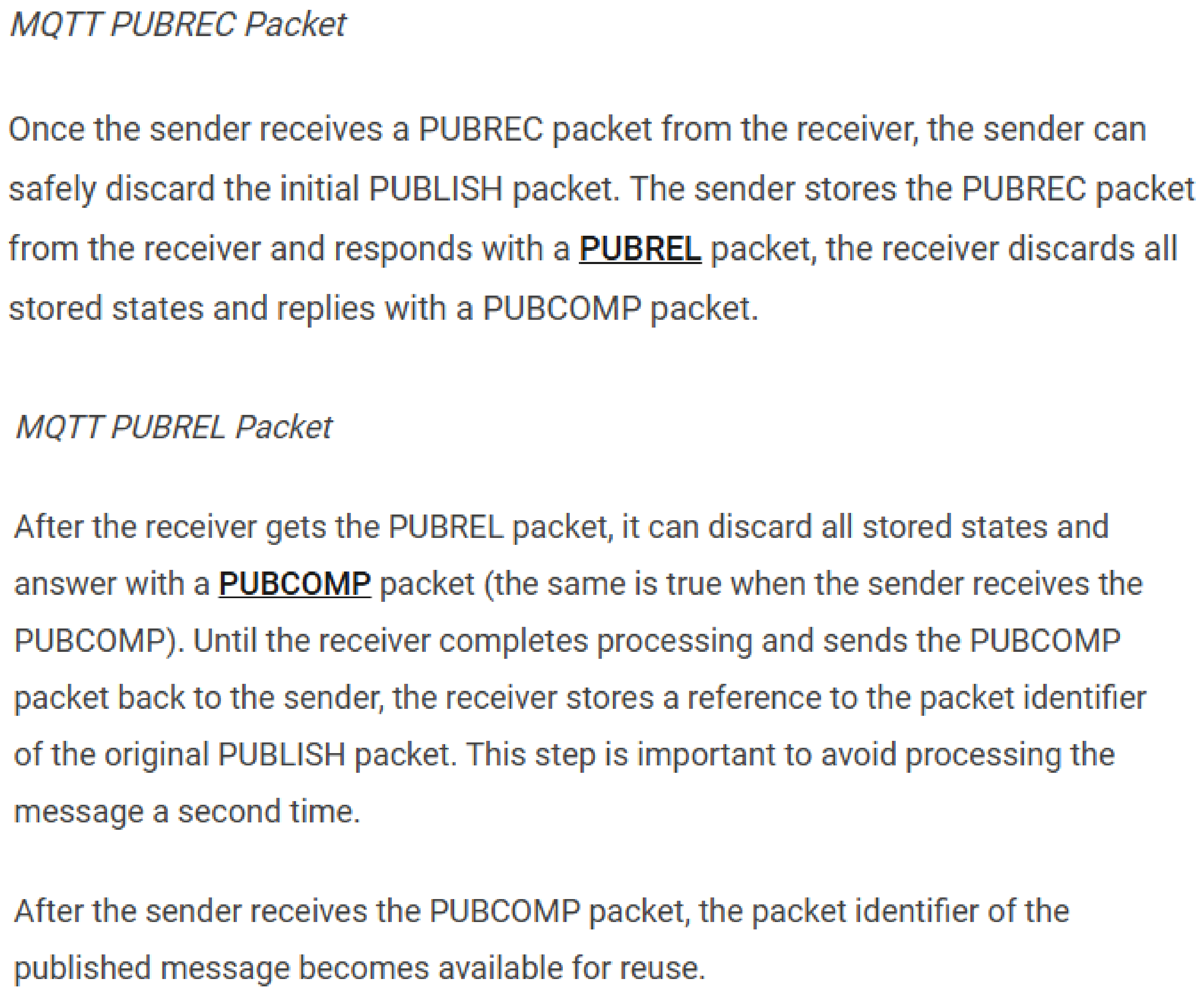
QoS 2 offers the highest level of service in MQTT, ensuring that each message is delivered exactly once to the intended recipients. To achieve this, QoS 2 involves a four-part handshake between the sender and receiver.



MQTT Quality of Service level 2: delivery exactly once

When a receiver gets a QoS 2 PUBLISH packet from a sender, it processes the publish message accordingly and replies to the sender with a **PUBREC** packet that acknowledges the PUBLISH packet. If the sender does not get a PUBREC packet from the receiver, it sends the PUBLISH packet again with a duplicate (DUP) flag until it receives an acknowledgment.

How does QoS 2 work in MQTT?



MQTT PUBREC Packet

Once the sender receives a PUBREC packet from the receiver, the sender can safely discard the initial PUBLISH packet. The sender stores the PUBREC packet from the receiver and responds with a **PUBREL** packet, the receiver discards all stored states and replies with a PUBCOMP packet.

MQTT PUBREL Packet

After the receiver gets the PUBREL packet, it can discard all stored states and answer with a **PUBCOMP** packet (the same is true when the sender receives the PUBCOMP). Until the receiver completes processing and sends the PUBCOMP packet back to the sender, the receiver stores a reference to the packet identifier of the original PUBLISH packet. This step is important to avoid processing the message a second time.

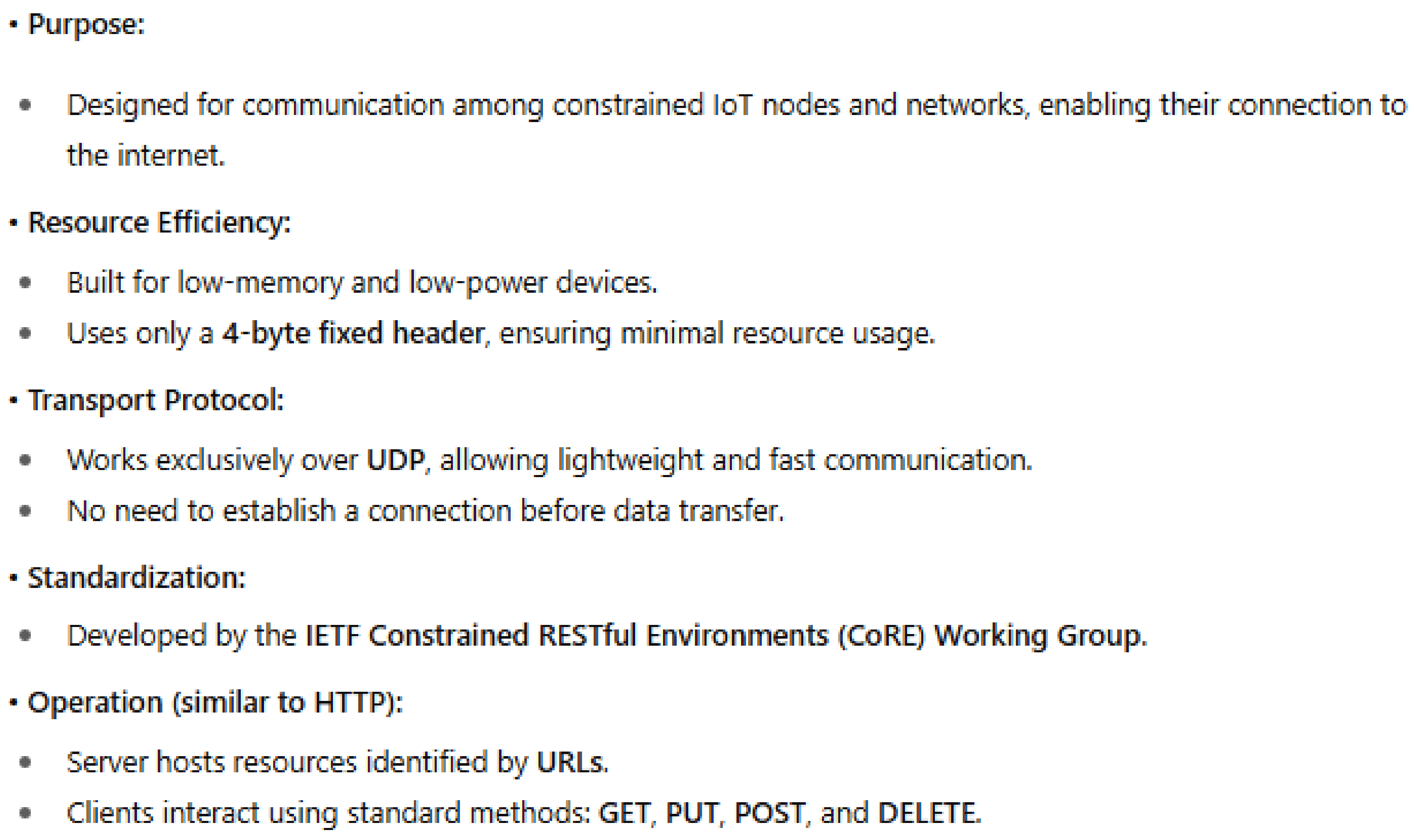
After the sender receives the PUBCOMP packet, the packet identifier of the published message becomes available for reuse.

How to host MQTT Broker?

- Two ways to host an MQTT broker: on a local server or on a cloud-based server.
- **Local Hosting:** This involves installing an open-source MQTT broker directly on server hardware. The document lists three examples of such brokers:
  - **Mosquitto**: A popular, lightweight open-source broker written in C. It is the default broker for edge networks.
  - **Mosca**: A simple broker ideal for small home networks. It is a Node.js-based broker that can be installed as a Node-RED node.
  - **emqttd**: A highly scalable, open-source broker written in Erlang.
- **Cloud-based Hosting:** Major firms like Google, Amazon, Microsoft, and IBM offer cloud-based MQTT servers and brokers. The text gives the example of Google Cloud IoT Core, which supports the MQTT protocol with a managed broker on port mqtt.googleapis.com:8883. The normal TCP port 8883 is designated for secure MQTT connections.

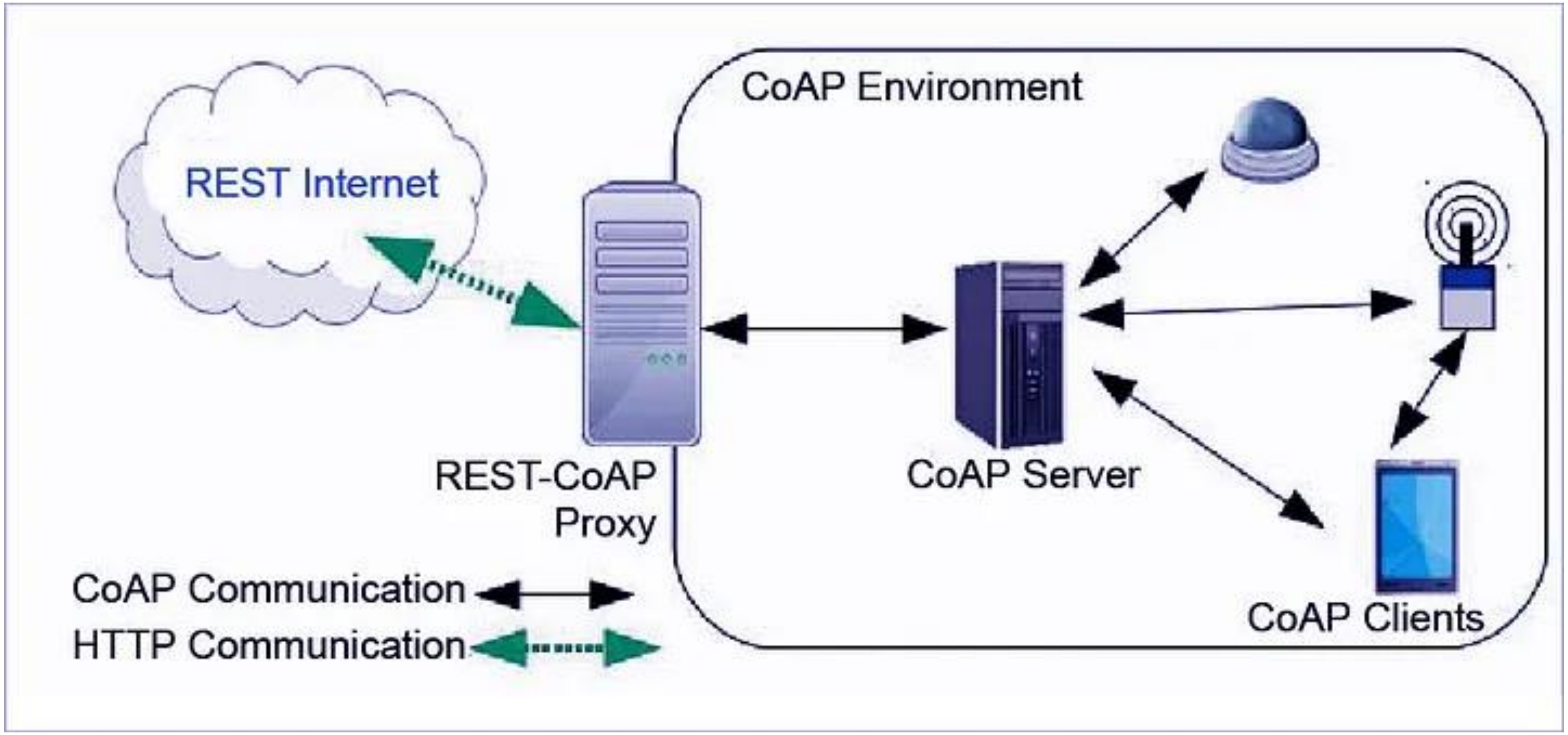
CoAP (Constrained Application Protocol)

CoAP is an internet utility protocol for constrained gadgets. It is designed to enable simple, constrained devices to join IoT through constrained networks having low bandwidth availability.

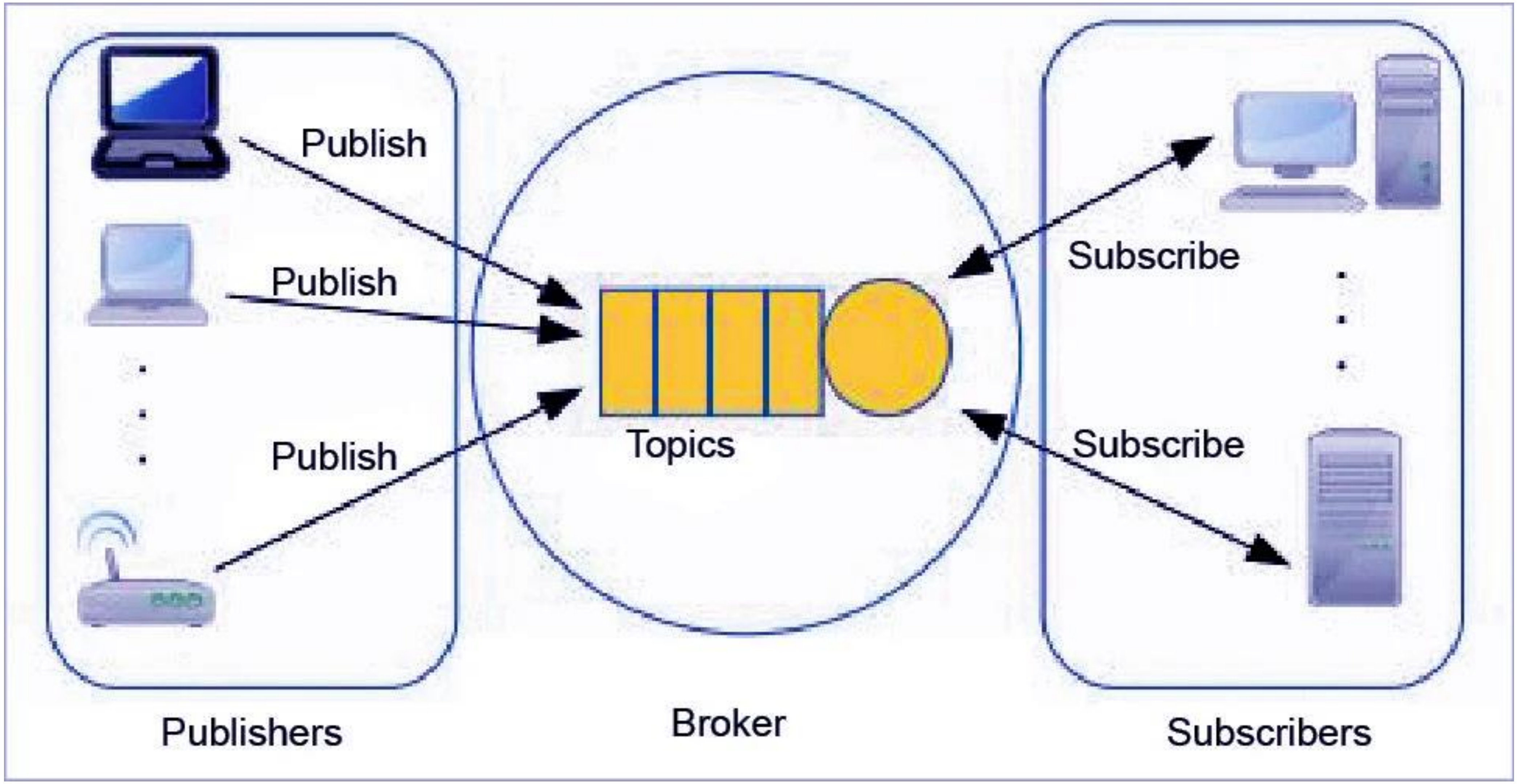


- **Purpose:**
  - Designed for communication among constrained IoT nodes and networks, enabling their connection to the internet.
- **Resource Efficiency:**
  - Built for low-memory and low-power devices.
  - Uses only a **4-byte fixed header**, ensuring minimal resource usage.
- **Transport Protocol:**
  - Works exclusively over **UDP**, allowing lightweight and fast communication.
  - No need to establish a connection before data transfer.
- **Standardization:**
  - Developed by the **IETF Constrained RESTful Environments (CoRE) Working Group.**
- **Operation (similar to HTTP):**
  - Server hosts resources identified by **URLs.**
  - Clients interact using standard methods: **GET**, **PUT**, **POST**, and **DELETE.**

CoAP (Constrained Application Protocol)



CoAP (Constrained Application Protocol)



CoAP (Constrained Application Protocol)

CoAP's methods (GET, PUT, POST, and DELETE) and response codes are similar to HTTP, which simplifies the process of mapping CoAP traffic to RESTful API logic.

Unlike MQTT, CoAP uses a resource model tied to a Universal Resource Identifier (URI), which is comparable to MQTT topics.

For example, a sensor publishing to a CoAP server would use a URI like

coap://devices/sensors/temperature,

while an MQTT client would use a topic like

/devices/sensors/temperature.

CoAP (Constrained Application Protocol)

- **Data Transmission:** A CoAP request sends a packet to a device, and the device's response is transmitted back in a response packet. A data packet can also be pushed to a device using a POST request to its URL.

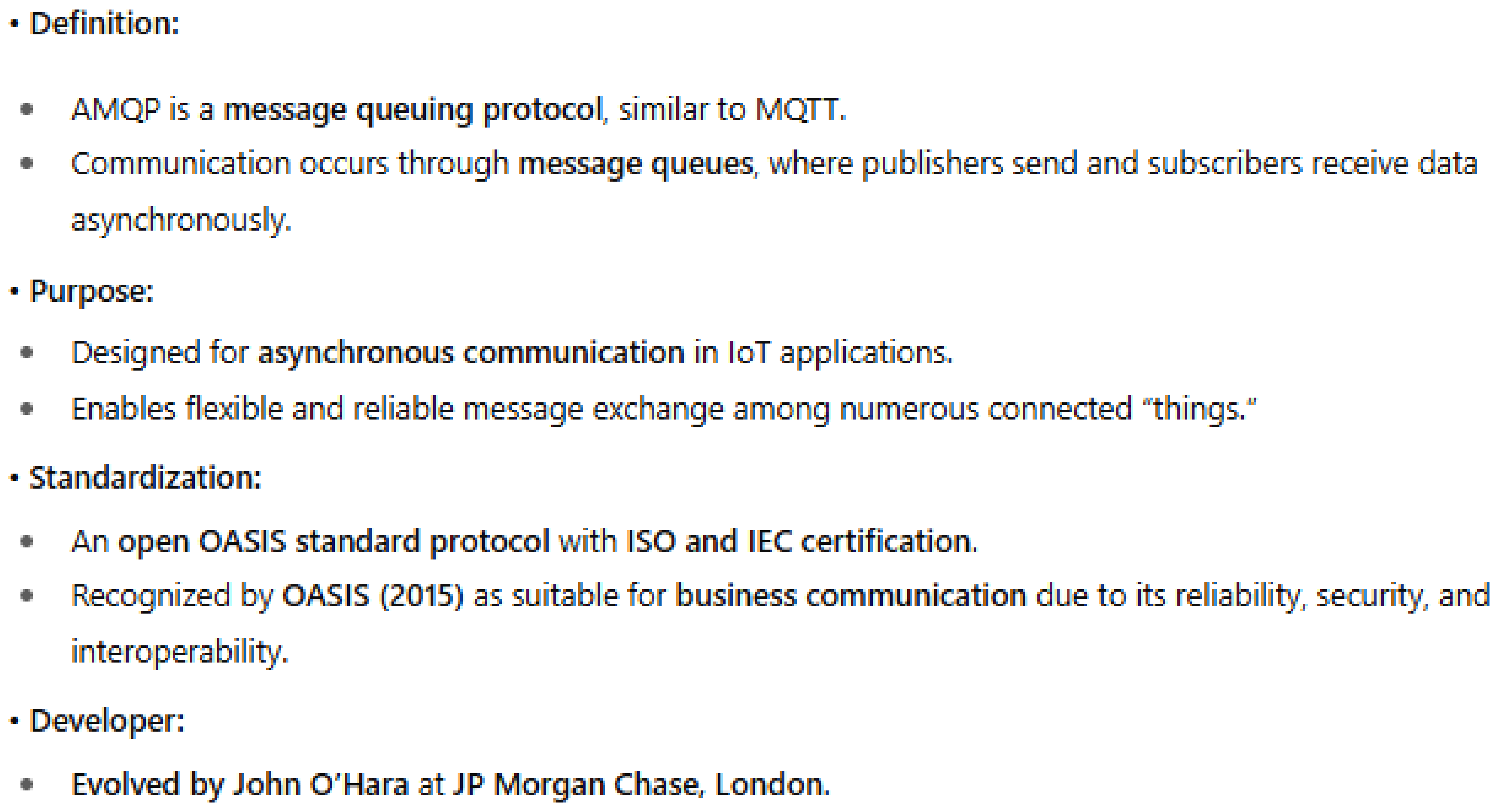
- **Similarities to HTTP:** The CoAP protocol shares many similarities with the HTTP protocol.

- **Key Differences from HTTP:** CoAP is designed primarily for IoT and Machine-to-Machine (M2M) communication. It has a low overhead, is easy to understand, and can send and receive messages asynchronously. It also has proxy and caching capabilities.

CoAP (Constrained Application Protocol)

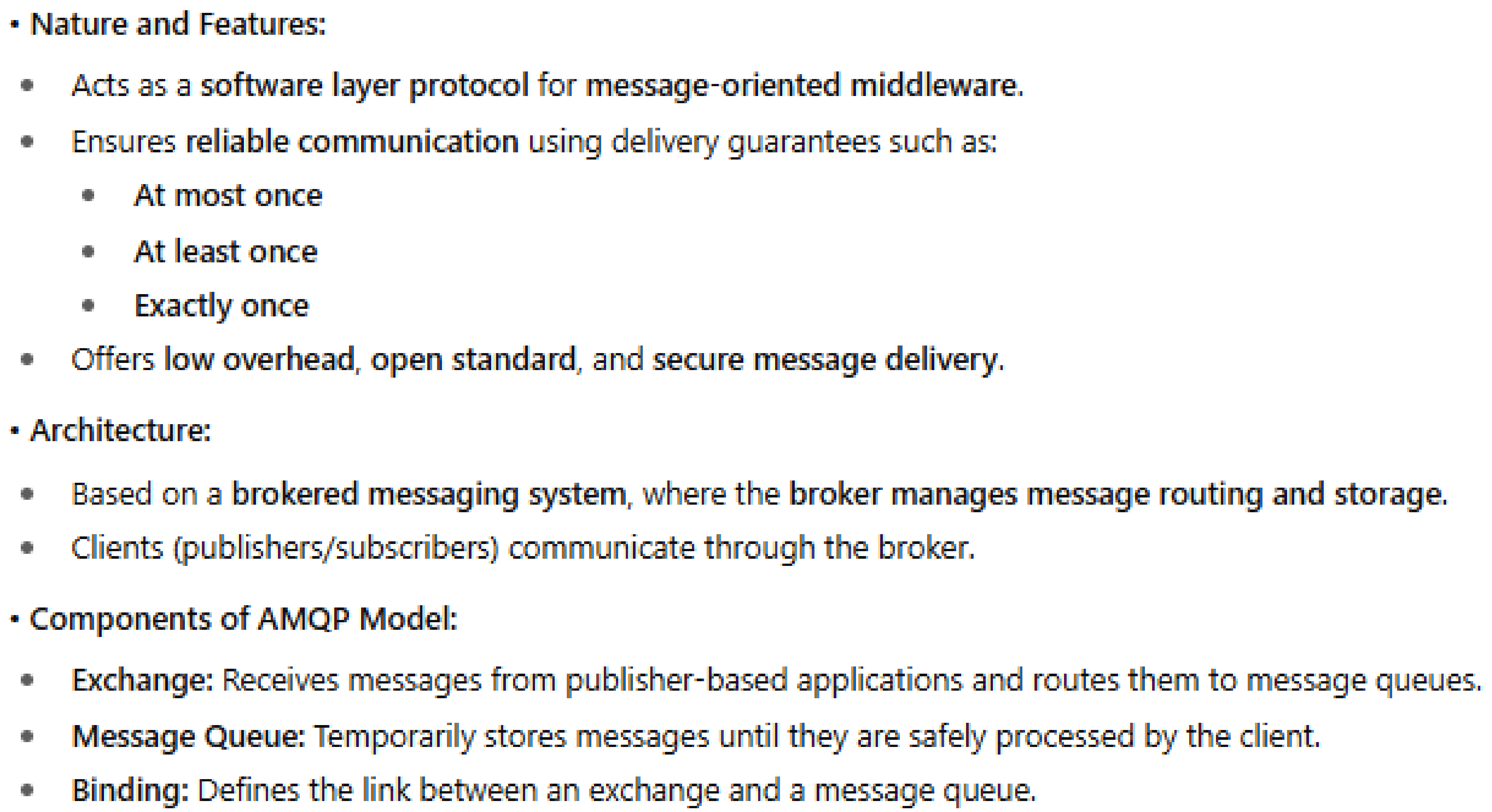
- **Reliability Mechanism:** CoAP creates its reliability using confirmable and non-confirmable messages, in contrast to MQTT's defined Quality of Service (QoS) levels.
  - **Confirmable Messages (CON):** These are dependable messages. A client will continue to retransmit a CON message using a timeout and back-off mechanism until it receives an acknowledgment (ACK) message from the server with the same message ID. If the server cannot process the CON message, it sends a reset (RST) message instead of an ACK. ACK messages can contain a payload and do not require further acknowledgments. [MQTT QoS 0]
  - **Non-confirmable Messages (NON):** These are unreliable messages used for non-critical transfers. The server does not acknowledge their arrival, but they are still given message IDs to help detect duplicates. [MQTT QoS 1]
- **Comparison to MQTT:** While CoAP does not have a defined QoS level model like MQTT, its confirmable and non-confirmable message types can be linked to MQTT's QoS semantics.

AMQP (Advanced Message Queuing Protocol)



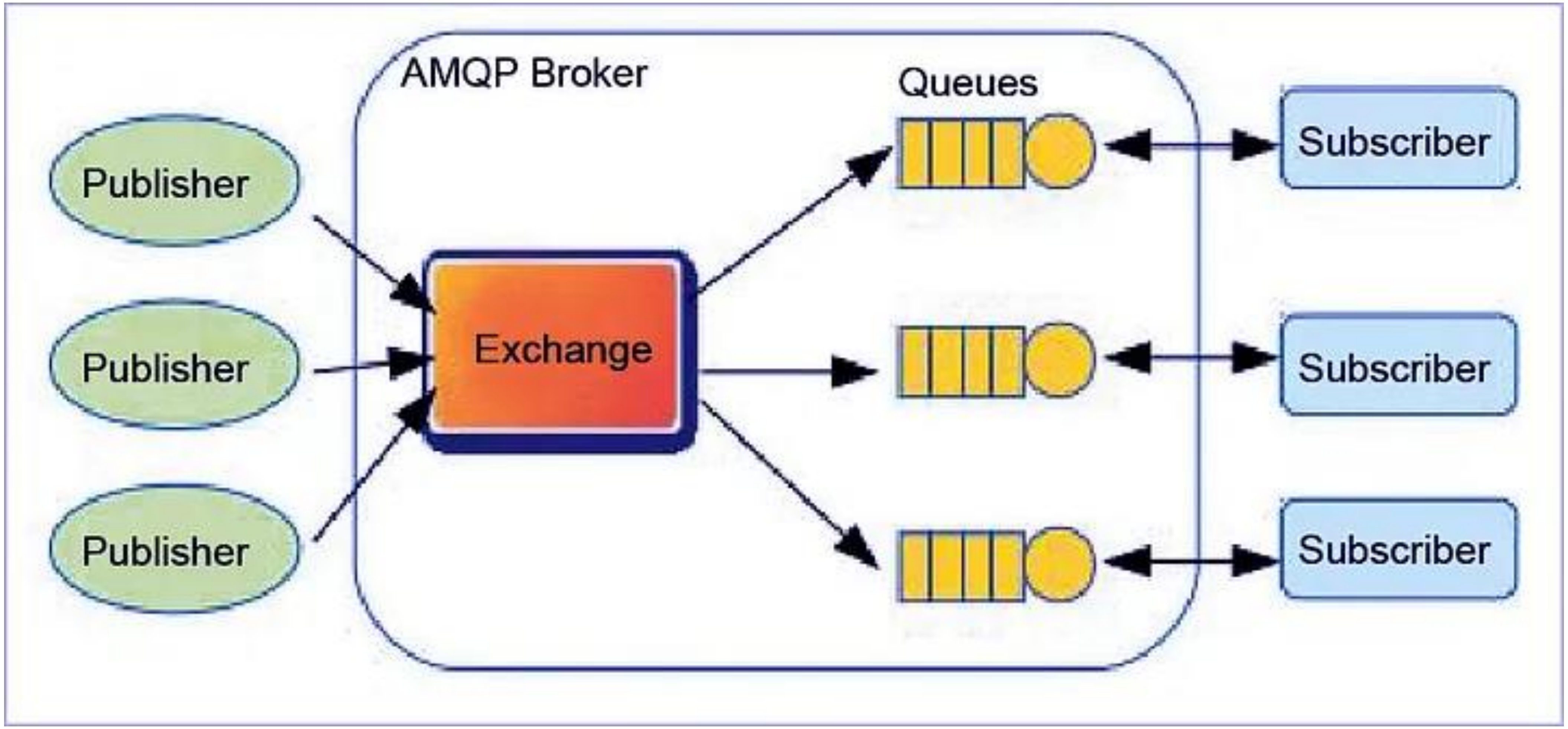
- **Definition:**
  - AMQP is a **message queuing protocol**, similar to MQTT.
  - Communication occurs through **message queues**, where publishers send and subscribers receive data asynchronously.
- **Purpose:**
  - Designed for **asynchronous communication** in IoT applications.
  - Enables flexible and reliable message exchange among numerous connected "things."
- **Standardization:**
  - An **open OASIS standard protocol** with **ISO and IEC certification.**
  - Recognized by **OASIS (2015)** as suitable for **business communication** due to its reliability, security, and interoperability.
- **Developer:**
  - **Evolved by John O'Hara at JP Morgan Chase, London.**

AMQP (Advanced Message Queuing Protocol)

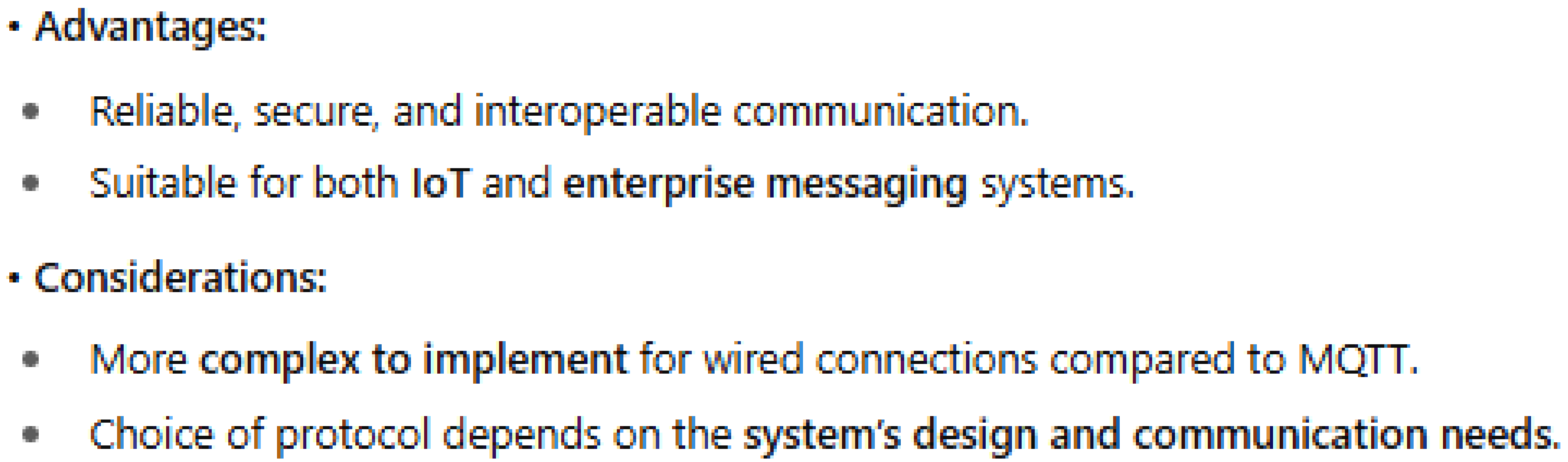


- **Nature and Features:**
  - Acts as a **software layer protocol** for **message-oriented middleware.**
  - Ensures **reliable communication** using delivery guarantees such as:
    - **At most once**
    - **At least once**
    - **Exactly once**
  - Offers **low overhead**, **open standard**, and **secure message delivery.**
- **Architecture:**
  - Based on a **brokered messaging system**, where the **broker manages message routing and storage.**
  - Clients (publishers/subscribers) communicate through the broker.
- **Components of AMQP Model:**
  - **Exchange:** Receives messages from publisher-based applications and routes them to message queues.
  - **Message Queue:** Temporarily stores messages until they are safely processed by the client.
  - **Binding:** Defines the link between an exchange and a message queue.

AMQP (Advanced Message Queuing Protocol)



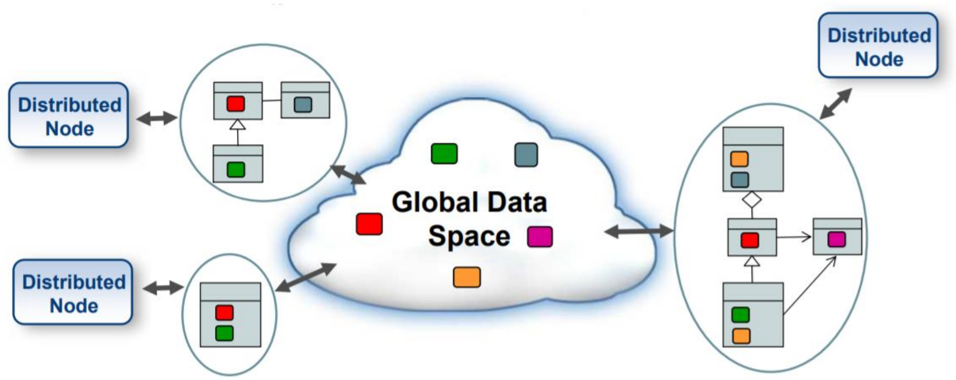
AMQP (Advanced Message Queuing Protocol)



- **Advantages:**
  - Reliable, secure, and interoperable communication.
  - Suitable for both IoT and enterprise messaging systems.
- **Considerations:**
  - More complex to implement for wired connections compared to MQTT.
  - Choice of protocol depends on the system's design and communication needs.

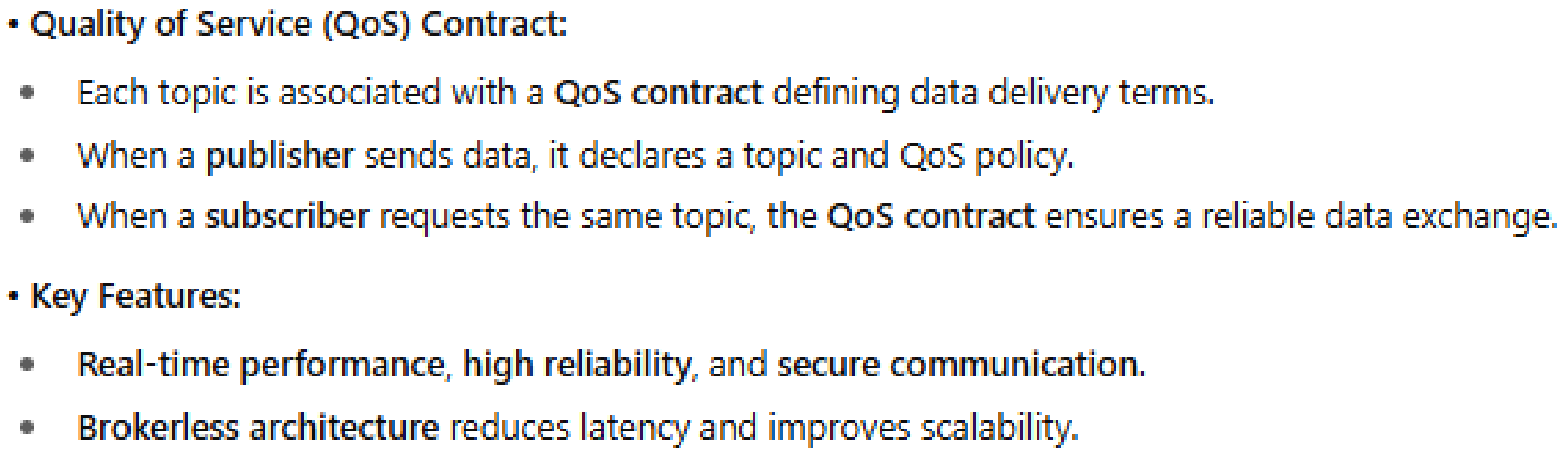
Data Distribution Service (DDS)

It enables a scalable, real-time, reliable, excessive-overall performance and interoperable statistics change via the submit-subscribe technique. DDS makes use of multicasting to convey high-quality QoS to applications.



- **Developer:**
  - Developed by the **Object Management Group (OMG).**
  - **First open and interoperable middleware protocol.**
- **Purpose:**
  - Provides **secure, real-time data distribution** for IoT systems.
  - Commonly used in **industrial IoT and Industry 4.0** applications.
- **Architecture:**
  - Uses a **Publisher/Subscriber model**, similar to MQTT.
  - Unlike MQTT, **DDS does not use a broker** — data flows directly among nodes.
- **Global Data Space (GDS):**
  - Acts as a **virtual shared memory** for data exchange.
  - Not a physical memory but a **concept combining local data stores** across all connected nodes.

Data Distribution Service (DDS)



- **Quality of Service (QoS) Contract:**
  - Each topic is associated with a **QoS contract** defining data delivery terms.
  - When a **publisher** sends data, it declares a topic and QoS policy.
  - When a **subscriber** requests the same topic, the **QoS contract** ensures a reliable data exchange.
- **Key Features:**
  - **Real-time performance**, **high reliability**, and **secure communication**.
  - **Brokerless architecture** reduces latency and improves scalability.

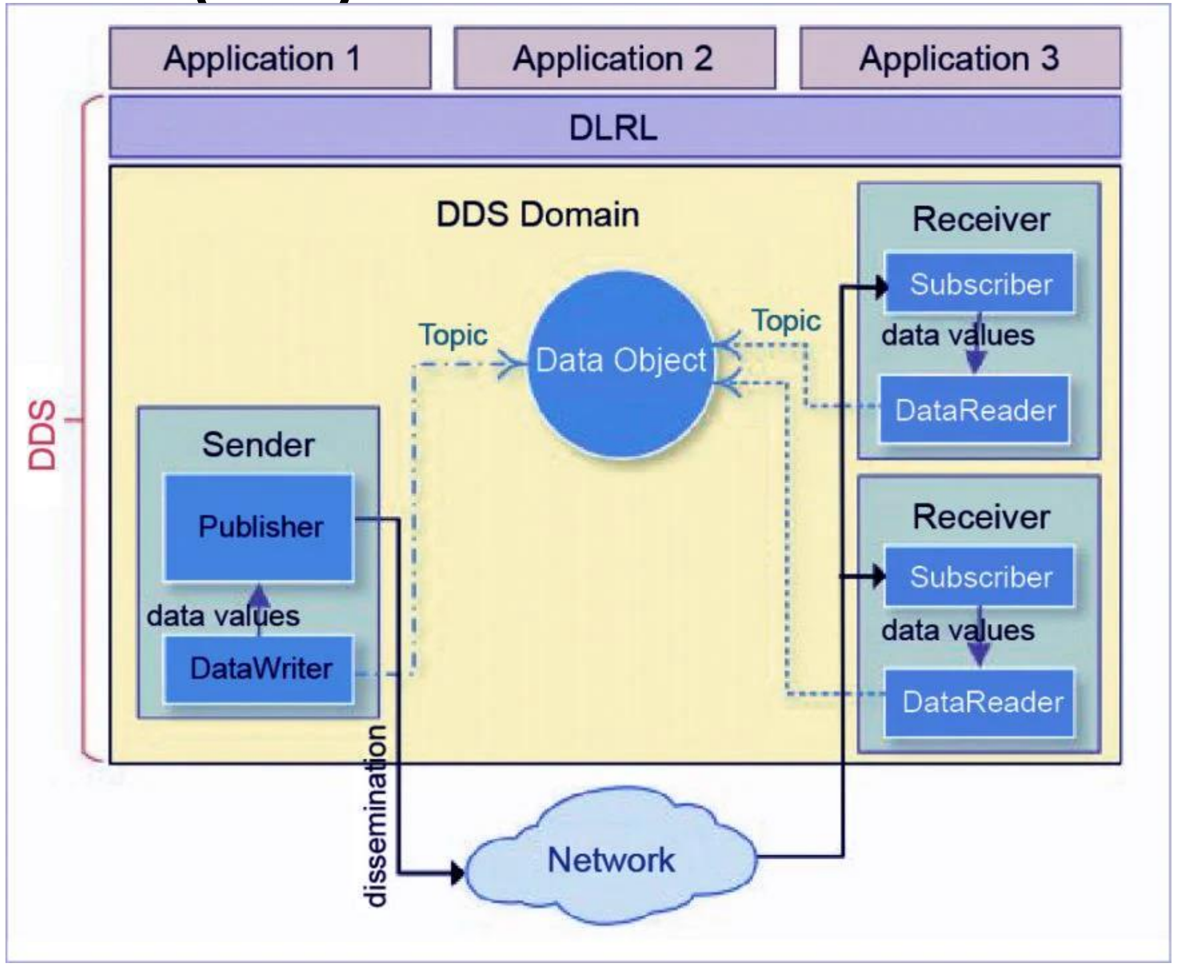
Data Distribution Service (DDS)

DDS is deployed in platforms ranging from low-footprint devices to the cloud and supports green bandwidth usage in addition to the agile orchestration of system additives.

The DDS – IoT protocols have two fundamental layers:
Data-Centric Publish-Subscribe (DCPS) and
Data Local Reconstruction Layer (dlrl).

DCPS layer plays the task of handing over the facts to subscribers (Core data exchange (topics, QoS, real-time)) , and

the DLRL layer Object-oriented abstraction of DCPS data ..it presents an interface to DCPS functionalities, permitting the sharing of distributed data amongst IoT enabled objects.



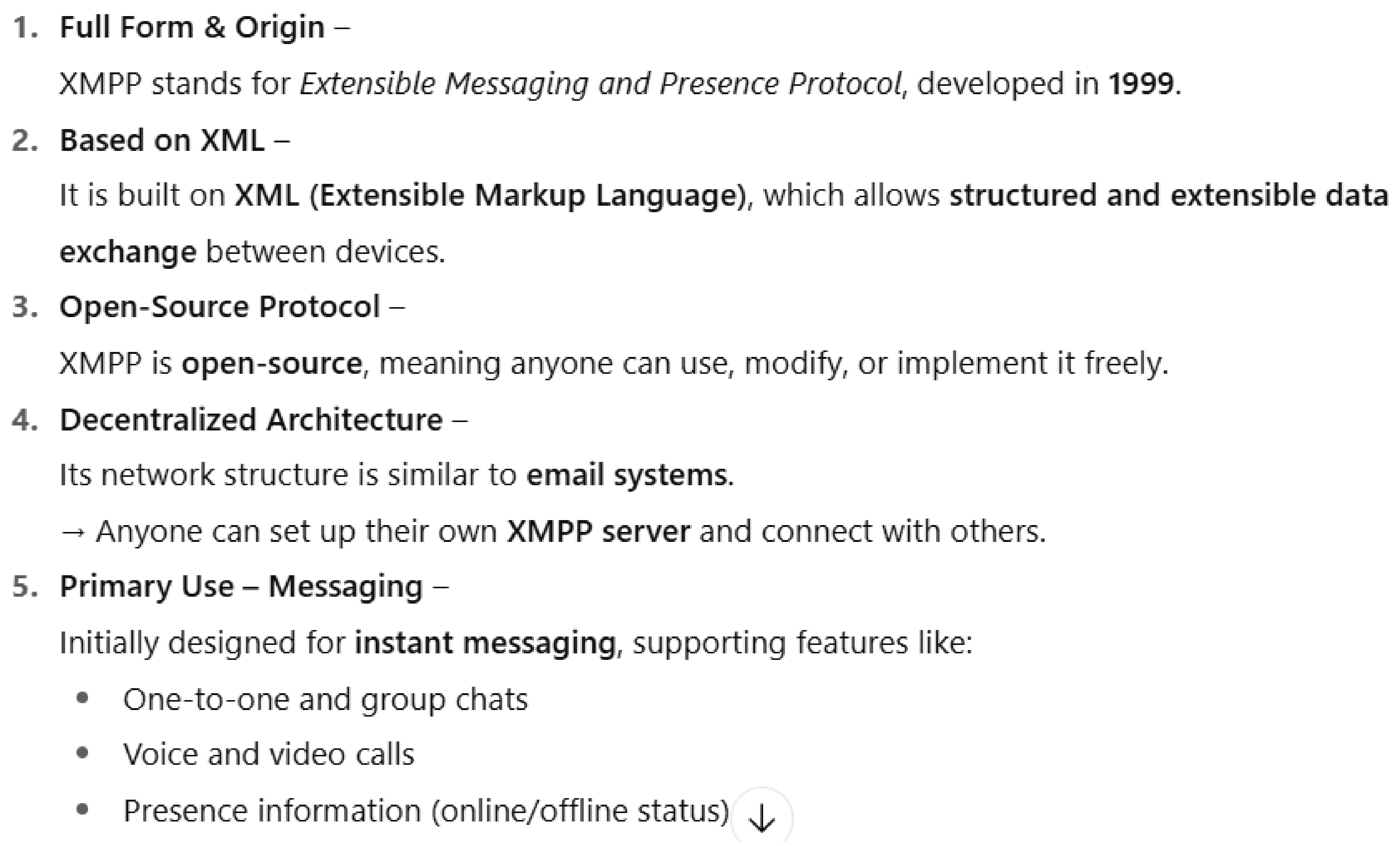
Application Layer Protocols - **define how IoT devices communicate data to applications or servers.**

Protocol	Description	Strengths	Limitations	Use Cases
MQTT (Message Queuing Telemetry Transport)	Lightweight publish-subscribe protocol	Low bandwidth, low power, supports unreliable networks	Requires broker, not suitable for very large payloads	Home automation, sensor networks, industrial IoT
CoAP (Constrained Application Protocol)	RESTful protocol for constrained devices, uses UDP	Lightweight, supports multicast, low overhead	Less reliable than TCP, security needs DTLS	Smart lighting, smart metering, building automation
HTTP/HTTPS	Standard web protocol	Widely used, easy integration	Heavyweight for constrained devices, high bandwidth	Web-enabled IoT devices, cloud communication
AMQP (Advanced Message Queuing Protocol)	Reliable messaging protocol	High reliability, ensures message delivery	Heavyweight, more complex than MQTT	Financial services, industrial IoT requiring guaranteed delivery
DDS (Data Distribution Service)	Real-time publish-subscribe protocol	Real-time, scalable, QoS support	Complex, higher resource usage	Autonomous vehicles, robotics, mission-critical IoT

IoT Protocols Comparison Table

Feature	MQTT	CoAP	HTTP/HTTPS	AMQP	DDS
Communication Model	Publish–Subscribe	Request–Response	Request–Response	Message Queue	Publish–Subscribe
Transport Layer	TCP	UDP	TCP	TCP	TCP / UDP
Bandwidth Usage	Very Low	Very Low	High	Moderate–High	Moderate
Latency	Low	Very Low	High	Moderate	Very Low (Real-time)
Reliability	Medium (QoS 0–2 levels)	Low (best effort)	High	Very High	Very High
Power Consumption	Low	Very Low	High	Moderate	Moderate–High
Security	TLS/SSL	DTLS	TLS/SSL	TLS/SSL	Built-in Security QoS
Scalability	High	High	Low	Moderate	Very High
Message Size	Small	Small	Large	Large	Variable
Best For	Remote sensors, home automation	Constrained IoT devices	Web-connected IoT systems	Industrial IoT, financial apps	Autonomous systems, robotics
Example Use Case	Smart home devices, wearables	Smart lighting, meters	Cloud dashboards	Banking, logistics IoT	Self-driving vehicles, drones

XMPP (Extensible Messaging and Presence Protocol)



1. **Full Form & Origin** –

   XMPP stands for *Extensible Messaging and Presence Protocol*, developed in **1999**.

2. **Based on XML** –

   It is built on **XML (Extensible Markup Language)**, which allows **structured and extensible data exchange** between devices.

3. **Open-Source Protocol** –

   XMPP is **open-source**, meaning anyone can use, modify, or implement it freely.

4. **Decentralized Architecture** –

   Its network structure is similar to **email systems**.

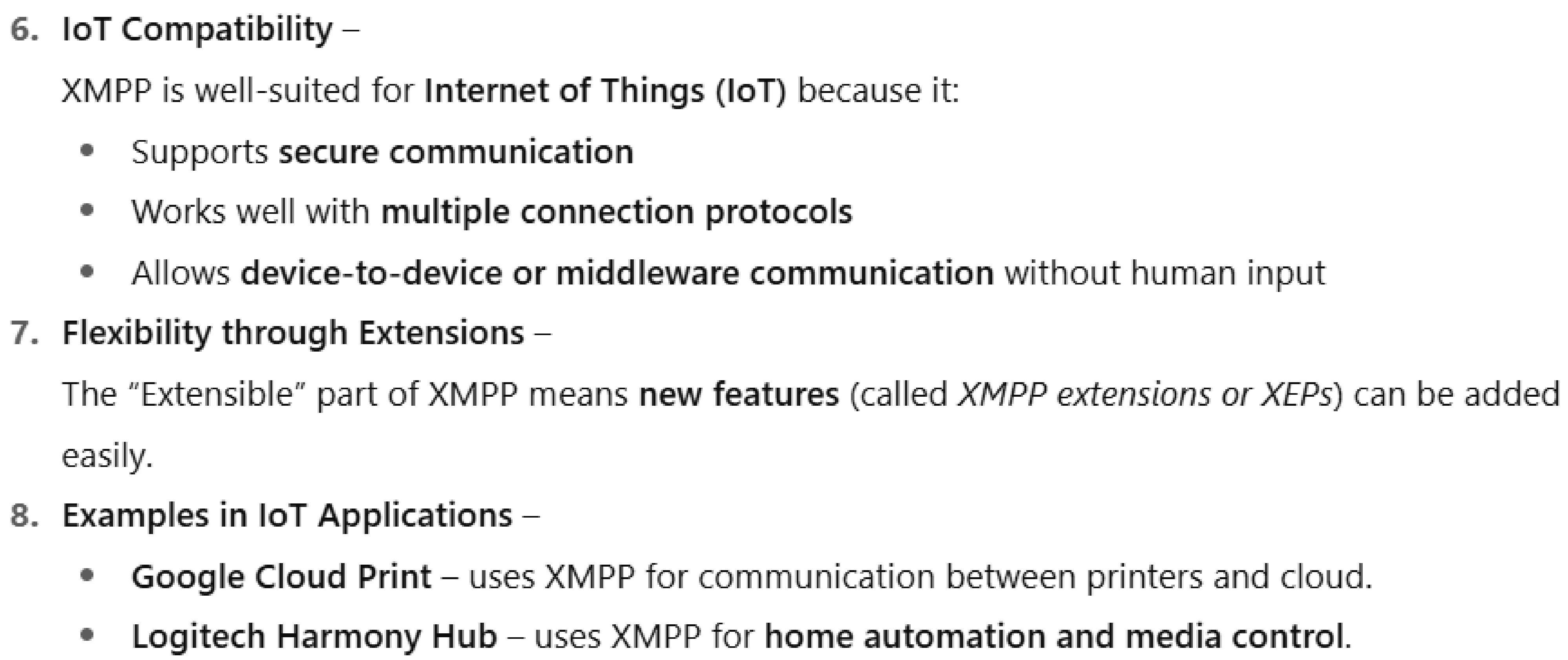
   → Anyone can set up their own **XMPP server** and connect with others.

5. **Primary Use – Messaging** –

   Initially designed for **instant messaging**, supporting features like:

   - One-to-one and group chats
   - Voice and video calls
   - Presence information (online/offline status)

XMPP (Extensible Messaging and Presence Protocol)



6. **IoT Compatibility** –

   XMPP is well-suited for **Internet of Things (IoT)** because it:

   - Supports **secure communication**
   - Works well with **multiple connection protocols**
   - Allows **device-to-device or middleware communication** without human input

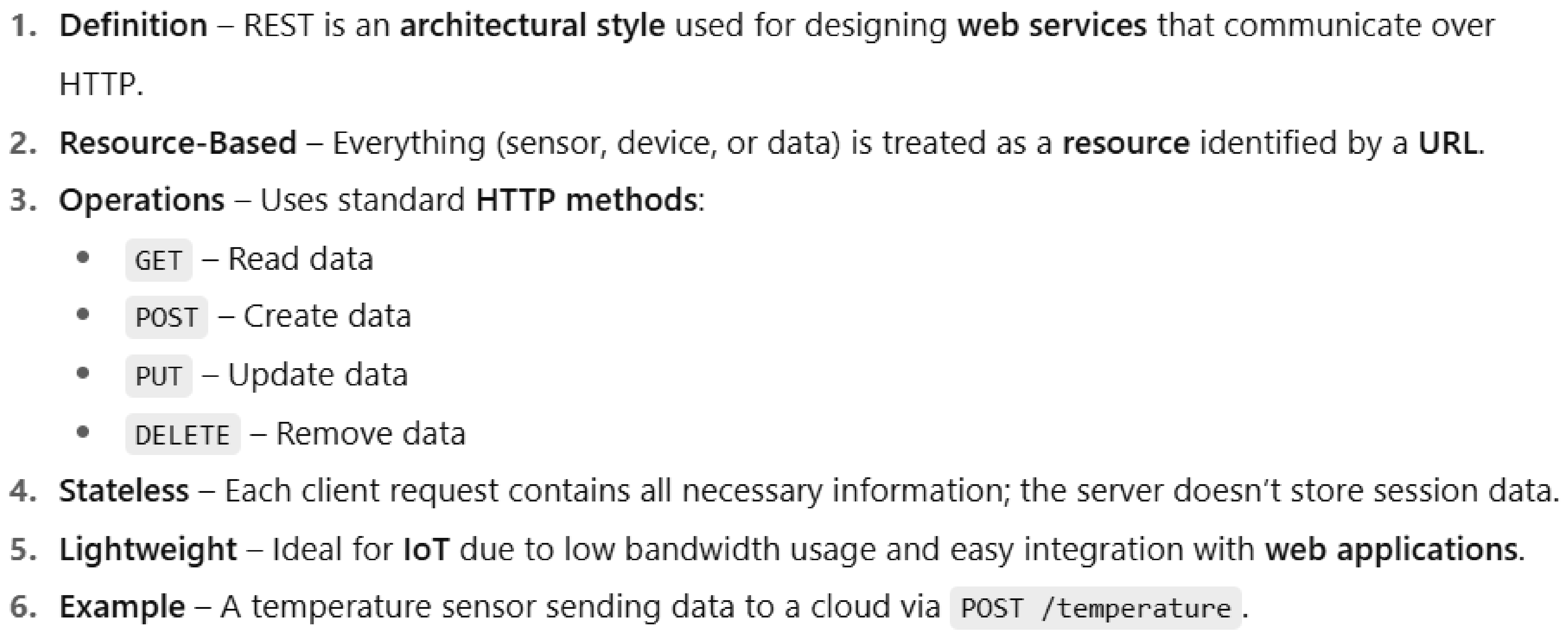
7. **Flexibility through Extensions** –

   The “Extensible” part of XMPP means **new features** (called *XMPP extensions or XEPs*) can be added easily.

8. **Examples in IoT Applications** –

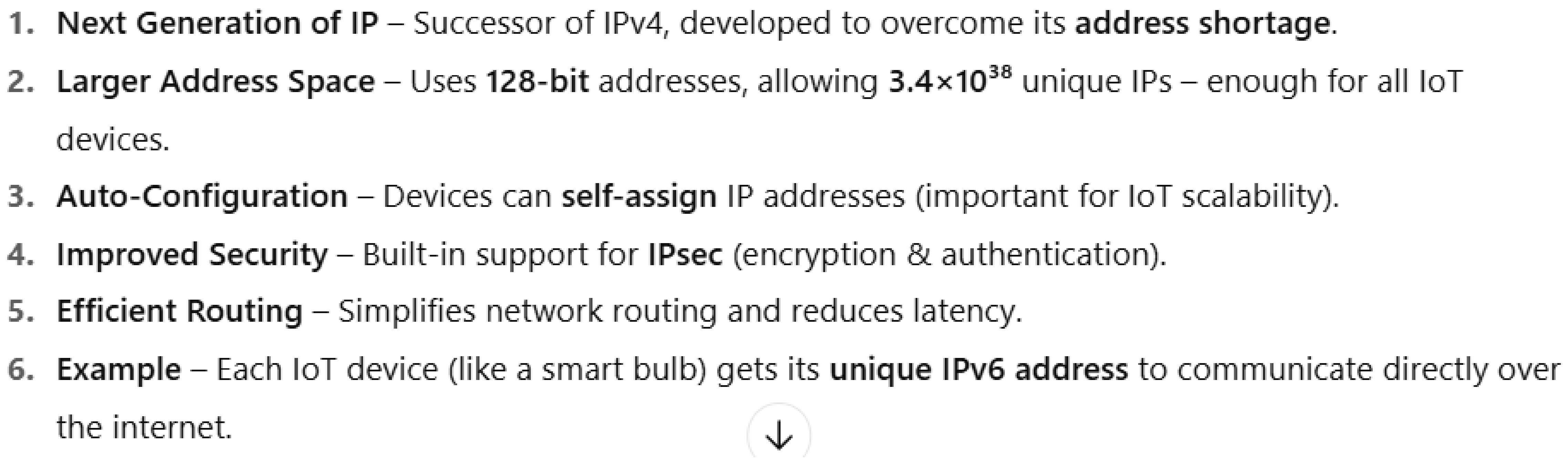
   - **Google Cloud Print** – uses XMPP for communication between printers and cloud.
   - **Logitech Harmony Hub** – uses XMPP for **home automation and media control**.

REST (Representational State Transfer)



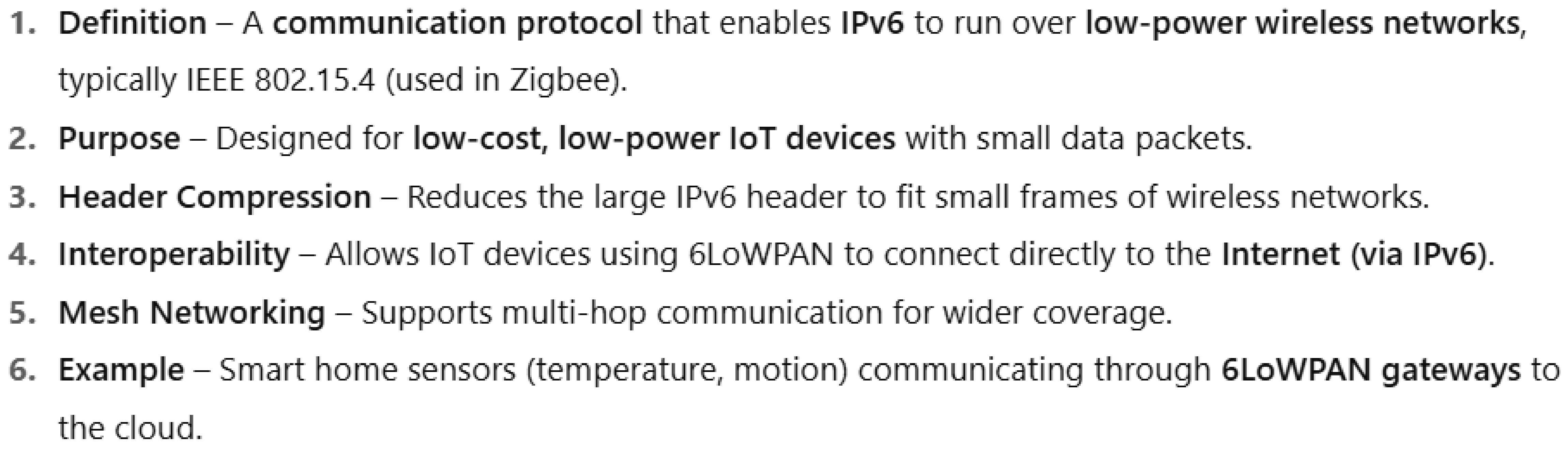
1. **Definition** – REST is an **architectural style** used for designing **web services** that communicate over HTTP.
2. **Resource-Based** – Everything (sensor, device, or data) is treated as a **resource** identified by a **URL**.
3. **Operations** – Uses standard **HTTP methods**:
   - `GET` – Read data
   - `POST` – Create data
   - `PUT` – Update data
   - `DELETE` – Remove data
4. **Stateless** – Each client request contains all necessary information; the server doesn't store session data.
5. **Lightweight** – Ideal for **IoT** due to low bandwidth usage and easy integration with **web applications**.
6. **Example** – A temperature sensor sending data to a cloud via `POST /temperature`.

IPv6 (Internet Protocol version 6)



1. **Next Generation of IP** – Successor of IPv4, developed to overcome its **address shortage**.
2. **Larger Address Space** – Uses **128-bit** addresses, allowing **3.4×10^{38}** unique IPs – enough for all IoT devices.
3. **Auto-Configuration** – Devices can **self-assign** IP addresses (important for IoT scalability).
4. **Improved Security** – Built-in support for **IPsec** (encryption & authentication).
5. **Efficient Routing** – Simplifies network routing and reduces latency.
6. **Example** – Each IoT device (like a smart bulb) gets its **unique IPv6 address** to communicate directly over the internet.

6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks)



1. **Definition** – A **communication protocol** that enables **IPv6** to run over **low-power wireless networks**, typically IEEE 802.15.4 (used in Zigbee).
2. **Purpose** – Designed for **low-cost, low-power IoT devices** with small data packets.
3. **Header Compression** – Reduces the large IPv6 header to fit small frames of wireless networks.
4. **Interoperability** – Allows IoT devices using 6LoWPAN to connect directly to the **Internet (via IPv6)**.
5. **Mesh Networking** – Supports multi-hop communication for wider coverage.
6. **Example** – Smart home sensors (temperature, motion) communicating through **6LoWPAN gateways** to the cloud.

Network & Transport Layer Protocols - **Handle how data moves between devices.**

Protocol	Description	Strengths	Limitations	Use Cases
TCP/IP	Reliable connection-oriented protocol	Reliable delivery, widely supported	Higher latency, more overhead	All IoT internet-connected devices
UDP	Connectionless protocol	Low latency, lightweight	No guaranteed delivery	CoAP, streaming data, gaming IoT
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks	Efficient over low-power wireless networks	Limited payload size	Wireless sensor networks, smart homes
RPL (Routing Protocol for Low-Power and Lossy Networks)	Routing protocol for low-power networks	Optimized for IoT mesh networks	Complexity in large networks	Smart cities, industrial monitoring

Wireless Communication Protocols - **physical & link layers, commonly used in IoT devices.**

Protocol	Frequency / Range	Strengths	Limitations	Use Cases
Bluetooth / BLE	2.4 GHz / short-range	Low power, easy pairing	Short range, limited data rate	Wearables, smart home devices
Zigbee	2.4 GHz / mesh network	Low power, mesh networking	Limited range per hop	Home automation, lighting control
LoRa / LoRaWAN	Sub-GHz / long-range	Long range, very low power	Low data rate, high latency	Agriculture, smart cities, asset tracking
Wi-Fi	2.4 / 5 GHz / medium range	High bandwidth	High power consumption	Smart appliances, cameras
NB-IoT / LTE-M	Cellular	Wide coverage, reliable	Data cost, higher power than LPWAN	Smart metering, city IoT, industrial IoT

IOT Protocols

- MQTT → Best for low-power, low-bandwidth IoT (sensors, wearables).
- CoAP → Best for constrained devices using UDP (smart meters, lighting).
- HTTP → Best for web integration and APIs, not energy efficient.
- AMQP → Best for enterprise-grade reliable messaging.
- DDS → Best for real-time, mission-critical IoT (robots, vehicles).

IoT Routing Protocols – Overview

Routing protocols in IoT are responsible for **efficient data transmission** between sensors, gateways, and cloud servers — ensuring reliability, low energy use, and scalability.

IoT networks are often **wireless sensor networks (WSNs)** — devices (nodes) sense data and forward it to a base station or gateway.

Because IoT nodes are **resource-constrained** (limited power, memory, and range), the routing mechanism must be **energy-efficient** and **adaptive**.

IoT Routing Protocols – Overview

IoT routing protocols are mainly categorized as:
Data-Centric Protocols
Flat-Architecture Protocols

Data-Centric Routing Protocols

In **data-centric routing**, communication is based on **data attributes (what)** rather than **node identity (who)**.

The idea is to **avoid redundant data transmission** and **reduce energy consumption**.

The **sink (base station)** sends a query specifying data attributes.
Sensor nodes that have the matching data respond.
Nodes may **aggregate or fuse** data to minimize transmission cost.

Data-Centric Routing Protocols

Focuses on content instead of node address.

Supports data aggregation (removes duplicates).

Reduces network traffic and power consumption.

Best suited for dense sensor networks.

Example:

1) **SPIN (Sensor Protocols for Information via Negotiation) -** Uses **meta-data negotiation** before sending actual data.
2) **Directed Diffusion -** The sink broadcasts an interest (query) specifying desired data attributes. Nodes that detect matching data create a gradient (path) back to the sink.

Flat-Architecture Routing Protocols

In **flat architecture**, all sensor nodes are **equal in role and responsibility** — there are **no hierarchical levels** (like cluster heads or gateways).

Each node cooperates in sensing, processing, and forwarding data.

Characteristics:

- **Peer-to-peer communication** among nodes.
- **No fixed hierarchy** (every node can act as source or relay).
- Suitable for **small to medium-sized** networks.
- Focus on **energy balance** among nodes to prolong network life.
- Often combined with **data-centric** approaches.

Flat-Architecture Routing Protocols

Advantages:

- **Simple design** and easy to deploy.
- **Load distribution** across nodes.
- High **fault tolerance** (since all nodes are equal).

Disadvantages:

- Not ideal for **large-scale** IoT networks.
- Increased **routing overhead** due to multiple peer connections.
- Harder to manage as network size grows.

Example 1: Smart Agriculture System

Scenario:
In a large smart farm, hundreds of soil and temperature sensors are deployed across fields.

How Data-Centric Routing Works:

- The **central controller (sink)** sends a query:
  "Send me temperature readings above 35°C and soil moisture below 20%."
- Only those **sensor nodes** with matching conditions respond.
- Nearby nodes **aggregate data** (e.g., average temperature of the zone) before sending to the controller.
- This avoids sending duplicate data from every sensor.

Benefits:

- Saves energy (only relevant data is transmitted).
- Reduces network traffic.
- Enables real-time irrigation decisions.

Example 2: Smart Home IoT Network

Scenario:
A smart home includes multiple IoT devices — smart lights, motion sensors, thermostats, and door locks — all connected in a **mesh network** (e.g., Zigbee or Thread).

How Flat Architecture Works:

- All devices are **equal**; there's no central router controlling everything.
- Data (like a motion alert) hops from one device to another until it reaches the home hub or gateway.
- If one path fails, another route is automatically used (self-healing).

Flooding - routing techniques in IoT

Flooding is the **simplest data dissemination technique** in IoT and sensor networks.

In flooding, **every node that receives a data packet broadcasts it to all its neighboring nodes** — except the one it received it from.

This process continues until the packet reaches the destination or its **Time-To-Live (TTL)** expires.

How It Works:

- A source node generates a data packet (for example, sensor reading).
- It broadcasts the packet to all nearby nodes.
- Each receiving node checks if it has already seen that packet:
- If not, it rebroadcasts the packet.
- If yes, it discards the duplicate.
- The process repeats until the packet reaches the sink (base station).

Flooding - routing techniques in IoT

Example:

Smart Forest Monitoring System
- Node A detects a fire and broadcasts an alert message.
- Its neighbors (B, C, D) receive it and re-broadcast to their neighbors.
- Eventually, the alert reaches the **base station (sink)** through multiple paths.

Result: The message is delivered quickly — but may be sent multiple times through redundant paths.

Emergency Alert Systems in IoT (e.g., wildfire or gas leak alerts) may use flooding for **fast broadcast** when reliability matters more than efficiency.

Zigbee discovery phase uses limited flooding to find new devices in a network.

Flooding - routing techniques in IoT

Advantages:

- **Simple to implement** — no need for routing tables or topology information.
- **Reliable delivery** — high chance that the packet will reach the destination (multiple paths).
- **Useful in dynamic networks** — works even if nodes move or fail.

Disadvantages:

- **Redundant messages** — multiple copies of the same packet flood the network.
- **High energy consumption** — due to repeated transmission by all nodes.
- **Network congestion (implosion)** — nodes get overwhelmed by too many messages.
- **Overlap problem** — nearby nodes may sense and send duplicate data.

Gossiping - routing techniques in IoT

Gossiping is a **modified version of flooding** that reduces redundant messages.

Instead of sending data to **all neighbors**, each node sends it to **only one randomly chosen neighbor**.

That neighbor again forwards the data to **another random neighbor**, and so on — like how *rumors spread in a crowd*.

How It Works:

- A source node sends data to **one random neighbor**.
- That neighbor picks another random neighbor (not the sender) and forwards it.
- The process continues until all nodes eventually receive the message.

Gossiping - routing techniques in IoT

Example: Smart Home Sensor Network

- A motion sensor detects movement and sends a message to **one** nearby light node.
- That node forwards it to another neighbor (say, the smart door lock).
- Gradually, the message reaches all devices — but without overloading the network.

Result: Fewer redundant messages compared to flooding; slower spread but more energy-efficient.

· **Smart Grid communication** — where energy meters or transformers exchange updates gradually using **gossip protocols** to balance load information.
· **Peer-to-Peer IoT systems** (like distributed blockchain or smart city nodes) use gossiping for slow but reliable data diffusion.

Gossiping - routing techniques in IoT

Advantages:

- Reduces redundancy — fewer duplicate packets.
- Less energy consumption — fewer transmissions.
- Prevents congestion — avoids broadcast storms.
- Simpler implementation — no need for routing tables.

Disadvantages:

- Longer delay — because only one neighbor is contacted each time.
- Possible message loss — if a node fails or a packet drops, some nodes may never receive the message.
- Not ideal for time-sensitive applications.

Flooding vs Gossiping

Flooding → Simple and fast but **inefficient**; best for **emergency or discovery** phases.
Gossiping → Efficient and low-energy but **slower**; best for **periodic data sharing or decentralized systems**.

SPIN- Sensor Protocols for Information via Negotiation

- **SPIN** stands for **Sensor Protocols for Information via Negotiation**.

- SPIN is a **data-centric routing protocol** designed for **Wireless Sensor Networks (WSN)** and **IoT** to efficiently disseminate data among sensor nodes.

- Instead of blindly flooding the network with data, SPIN uses **meta-data negotiation** before transmitting actual data.
  This avoids **redundant data transmission** and **saves energy**.

SPIN- Sensor Protocols for Information via Negotiation

- **SPIN** stands for **Sensor Protocols for Information via Negotiation**.

- SPIN is a **data-centric routing protocol** designed for **Wireless Sensor Networks (WSN)** and **IoT** to efficiently disseminate data among sensor nodes.

- Instead of blindly flooding the network with data, SPIN uses **meta-data negotiation** before transmitting actual data.
  This avoids **redundant data transmission** and **saves energy**.

Working Principle – “Negotiation Before Transmission”

SPIN operates in **three stages**:

ADV (Advertisement):

- A node that has new data advertises it using a small meta-data message.
- Meta-data describes the type of data (e.g., "Temperature Data, 32°C at location X").

REQ (Request):

- Neighboring nodes interested in that data send a **REQ** message to request it.

DATA (Data Transmission):

- The node sends the actual **data** to those who requested it.

This process ensures that only interested nodes receive the data, reducing **unnecessary transmission**.

Working Principle – “Negotiation Before Transmission”

Advantages:

- Reduces redundant data transmission.
- Energy-efficient compared to flooding.
- Adaptive — works well with dynamic topologies.
- Simpler to implement and scalable.

Disadvantages:

- Not suitable for time-critical or real-time data delivery.
- Data may not reach all nodes if no neighbor shows interest.
- Works well only in networks with cooperative nodes.

SPIN Variants

SPIN has several improved versions designed for different network conditions and requirements.

- SPIN-PP (Point-to-Point)
- SPIN-EC (Energy-Conserve)
- SPIN-BC (Broadcast)
- SPIN-RL (Reliable)

SPIN-PP (Point-to-Point)

SPIN-PP is designed for point-to-point communication — where nodes communicate directly (not via broadcast).

Features:

- Each node sends ADV → REQ → DATA only to one specific neighbor at a time.
- Useful in low-density networks where nodes are few and far apart.
- Reduces collision and interference.

Use Case Example:
Sparse IoT deployments such as pipeline monitoring or rural environmental sensing.

SPIN-EC (Energy-Conserve)

SPIN-EC focuses on energy conservation and network lifetime extension.

Key Feature:

- Nodes **monitor their residual energy** before participating in data negotiation.
- If a node's energy falls **below a threshold**, it stops participating in data transmission.

How It Works:

- Each node checks:
  If (Energy > Threshold) → Send ADV/REQ/DATA
  Else → Remain Idle (to save power)

Benefit:

- Maximizes **network lifetime** by preventing low-energy nodes from dying early.

Use Case Example:

- Battery-powered IoT sensor networks (e.g., forest fire detection).

SPIN-BC (Broadcast)

SPIN-BC is a broadcast version of SPIN, used when all nodes can directly hear each other (one-hop network).

Features:

- Uses **broadcast** instead of unicast.
- No need for meta-data negotiation for every neighbor separately.
- Simplifies communication in small or dense IoT networks.

Drawback:

- May cause more **redundant transmissions** compared to SPIN-PP.

Use Case Example:

- Smart classroom or small-area IoT systems where all sensors are within range of each other.

SPIN-RL (Reliable)

SPIN-RL focuses on reliable data delivery in lossy or unstable networks.

Features:

- Ensures **acknowledgments (ACKs)** for DATA packets.
- **Retransmits** data if ACK is not received.
- Maintains a record of transmitted packets to handle failures.

Benefit:

- Guarantees data reaches all interested nodes even in poor signal conditions.

Use Case Example:

- **Industrial IoT** or **disaster monitoring systems** where reliability is critical.

SPIN-RL (Reliable)

SPIN-RL focuses on reliable data delivery in lossy or unstable networks.

Features:

- Ensures **acknowledgments (ACKs)** for DATA packets.
- **Retransmits** data if ACK is not received.
- Maintains a record of transmitted packets to handle failures.

Benefit:

- Guarantees data reaches all interested nodes even in poor signal conditions.

Use Case Example:

- **Industrial IoT** or **disaster monitoring systems** where reliability is critical.

LEACH Protocol (Low-Energy Adaptive Clustering Hierarchy)

LEACH is a **hierarchical (cluster-based) routing protocol** designed to:

- Minimize energy consumption,
- Prolong the network lifetime,
- Reduce data transmission load in Wireless Sensor Networks (WSN).

It was one of the **first protocols** to use the concept of **clustering** for energy efficiency.

Working Principle

LEACH operates in **rounds**, and each round has **two phases**:

LEACH - 1. Setup Phase

- **Cluster Formation:** The network is divided into clusters.
- **Cluster Head (CH) Selection:**
  - Each node generates a random number between 0 and 1.
  - If the number is less than a threshold, it becomes a **Cluster Head (CH)** for that round.
  - The threshold ensures that every node becomes a CH once in every 1/p rounds (where *p* = desired percentage of CHs).
- **Advertisement:** Selected CHs broadcast their status.
- **Join Request:** Normal nodes join the nearest CH based on signal strength.
- **TDMA Schedule:** The CH assigns time slots to its member nodes to send data.

LEACH - 2. Steady-State Phase

· Member nodes send their data to CH during their assigned **TDMA slot**.
· CH performs **data aggregation/compression**.
· CH transmits the aggregated data to the **base station (sink)**.
· After one round, the network goes back to the setup phase, and new CHs are elected to balance energy use.

Example

Imagine a **smart agriculture field** with 100 sensor nodes measuring soil moisture and temperature.

- 10 nodes are selected as **Cluster Heads** (CHs).
- Each CH gathers data from its nearby 9 nodes.
- CHs aggregate and send data to the **base station** (e.g., farm control center).
- In the next round, **different nodes** become CHs to balance energy load.

LEACH

- **Advantages**

- Reduces energy consumption through clustering and data aggregation.
- Distributes energy load evenly (rotating CHs).
- Reduces direct long-distance communication with base station.
- Scalable for large IoT networks.

- **Disadvantages**

- Random CH selection may not be optimal — some CHs may die early.
- Not suitable for **mobile nodes** (assumes static deployment).
- Performs poorly when nodes are unevenly distributed.
- Single-hop communication from CH to base station can waste energy in large areas.

Reference

- https://www.nexpcb.com/blog/different-data-protocols-which-one-to-choose
- https://onomondo.com/blog/iot-networking-protocols-overview-and-advantages/
- https://medium.com/@rinu.gour123/4-major-iot-protocols-mqtt-coap-amqp-dds-46016897c3e9
- https://data-flair.training/blogs/iot-protocols/
- https://data-flair.training/blogs/iot-technology/