# Communication in Distributed Systems I

## Course: Robotic Programming Environments

Michał Drwięga
michal.drwiega@pwr.edu.pl
www.mdrwiega.com/edu/rpe

Department of Cybernetics and Robotics
Wrocław University of Science and Technology

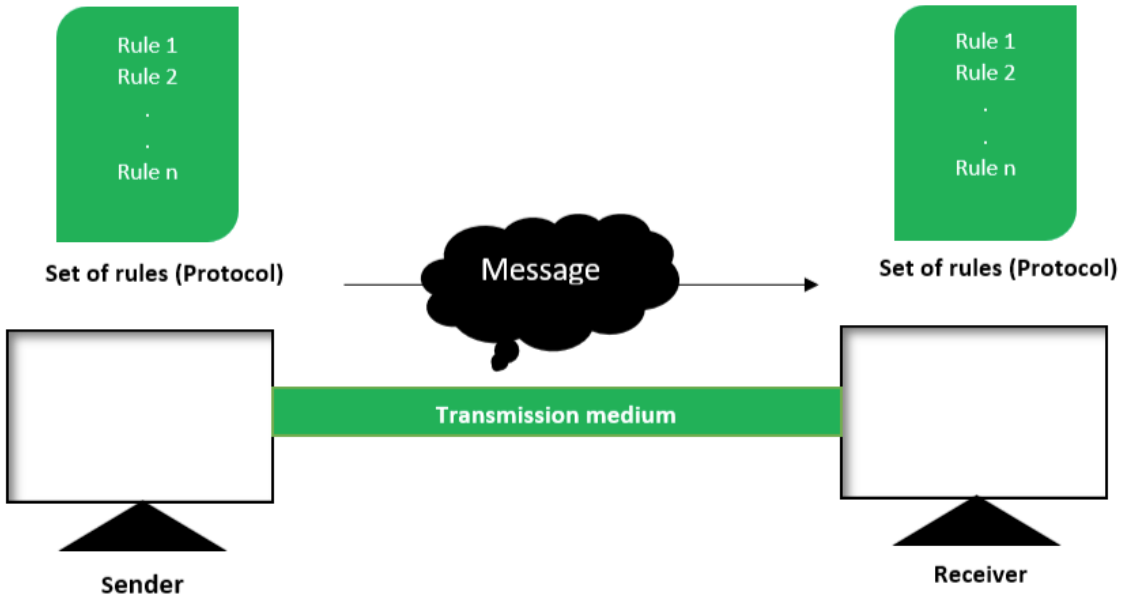
Wrocław University
of Science and Technology

# Content

- Communication basics: Mode, topology
- Embedded networking
- OSI model
- MQTT

# Basics of Data Communication

**Data communication** is the exchange of data between two devices via some kind of transmission medium (wire, wireless).




1

[1] https://www.geeksforgeeks.org/

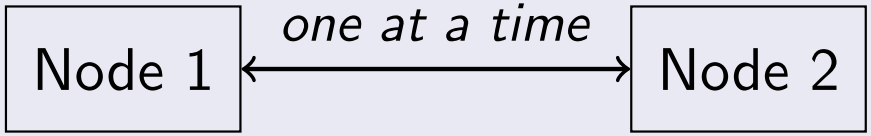
# Fundamentals of Data Communication

## Data Transmission Modes

**Simplex**

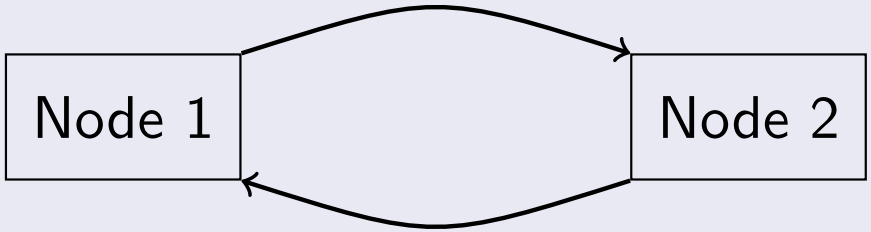

Communication flows in **one direction only** (e.g., keyboard to computer).

**Half-duplex**




Nodes can transmit and receive, but **not simultaneously** (e.g., walkie-talkie).

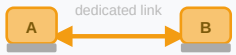
**Full-duplex**



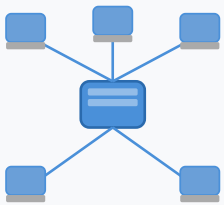

Nodes can transmit and receive **simultaneously in both directions** (e.g., telephone).

# Network Topology

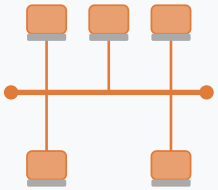
## Point to Point



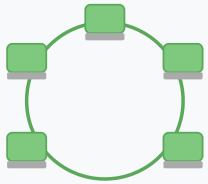

## Star



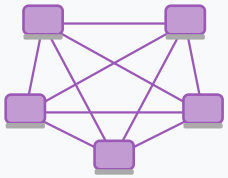
## Bus



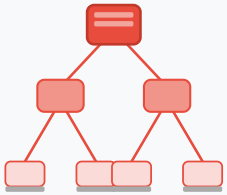
## Ring



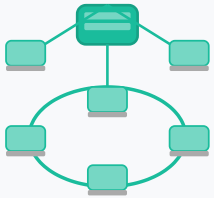
## Mesh



## Tree



## Hybrid



# Types of Embedded Networking

Wired interfaces

- **RS232**: full-duplex, max 1MBit/s, range 15 m
- **RS485**: half/full-duplex, max 10MBit/s, range 1200 m
- **I**$^2$**C**: half-duplex, max 3,6 Mbit/s, short range[2]
- **SPI**: full-duplex, $>$ 10 Mbit/s, short range
- **CAN**: half duplex, 1 Mbit/s
- **USB**: full-duplex, $>=$ 20Gbit/s, short range
- **Ethernet**: full-duplex, $>=$ 1 Gbit/s, long range

[2] https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

# RS-232 vs. RS-485

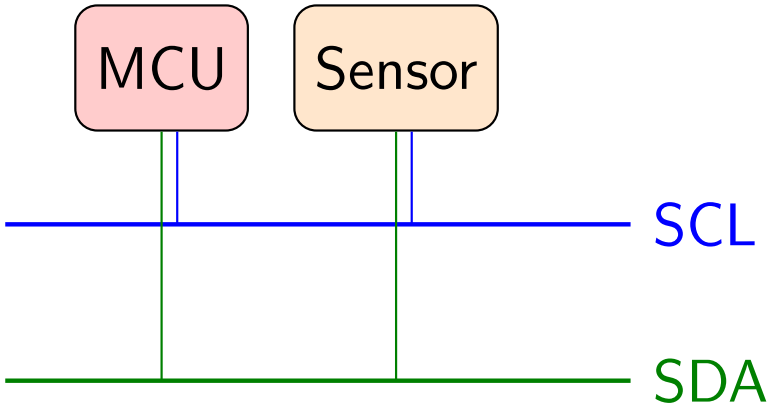
**RS-232**

- **Communication:** Point-to-point
- **Signaling:** Single-ended
- **Noise immunity:** Lower
- **Topology:** Point-to-point / star
- **Distance:** Short (typically up to 15 m)
- **Use cases:** PC peripherals, legacy serial ports

**RS-485**

- **Communication:** Multipoint (shared bus)
- **Signaling:** Differential
- **Noise immunity:** High
- **Topology:** Bus
- **Distance:** Long (up to 1200 m)
- **Use cases:** Industrial networks, building automation

# I²C (Inter-Integrated Circuit)




## Bus Characteristics

- **2-wire interface:**
  - SDA (Serial Data)
  - SCL (Serial Clock)
- **Multi-master, multi-slave** support
- **Open-drain** topology (pull-up resistors required)
- **7-bit or 10-bit** addressing

## Communication Features

- **Acknowledgment (ACK/NACK)** per byte
- **Clock stretching** for slow slaves
- **Start/Stop conditions** frame transactions
- **Multi-byte** burst transfers

## Speed Modes

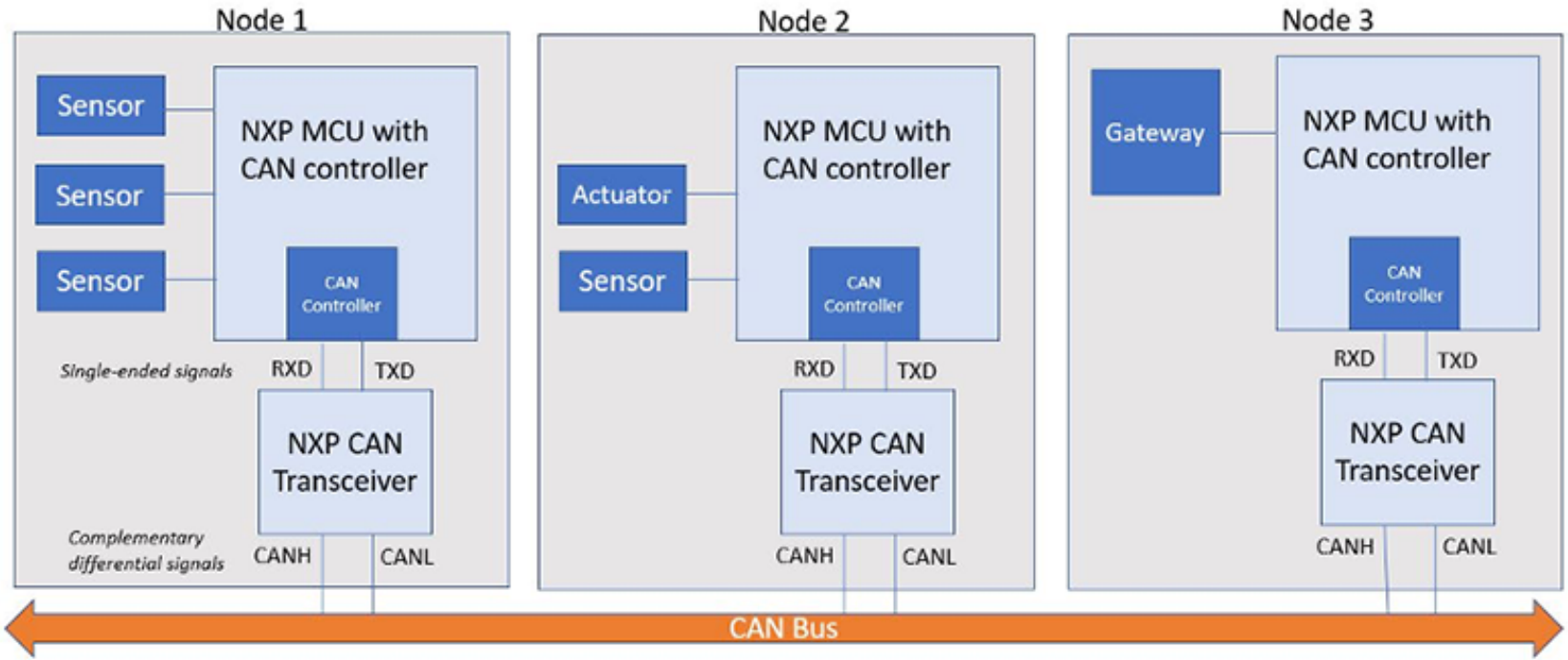
| Mode | Max Speed |
|---|---|
| Standard-mode | 100 kbit/s |
| Fast-mode | 400 kbit/s |
| Fast-mode Plus | 1 Mbit/s |
| High-speed mode | 3.4 Mbit/s |
| Ultra Fast-mode | 5 Mbit/s |

# SPI (Serial Peripheral Interface)

- **Interface**
  - Four-wire protocol: *MOSI*, *MISO*, *SCK*, *SS (CS)*
  - Synchronous communication using a shared clock (SCK)
- **Architecture**
  - Master–slave communication model
  - Typically point-to-point (one master, one slave)
  - No addressing — each slave requires a separate *SS/CS* line
- **Performance**
  - High-speed communication
  - Typical speeds: **1 Mbps – tens of Mbps**
  - **Full-duplex**: simultaneous transmission and reception
- **Applications**
  - SD cards
  - Displays
  - ADC/DAC converters
  - Sensors

# CAN (Control Area Network)

- Developed specifically for the automotive industry.
- Utilizes a two-wire, half-duplex communication system.
- Message prioritization: Each message is assigned a priority level.

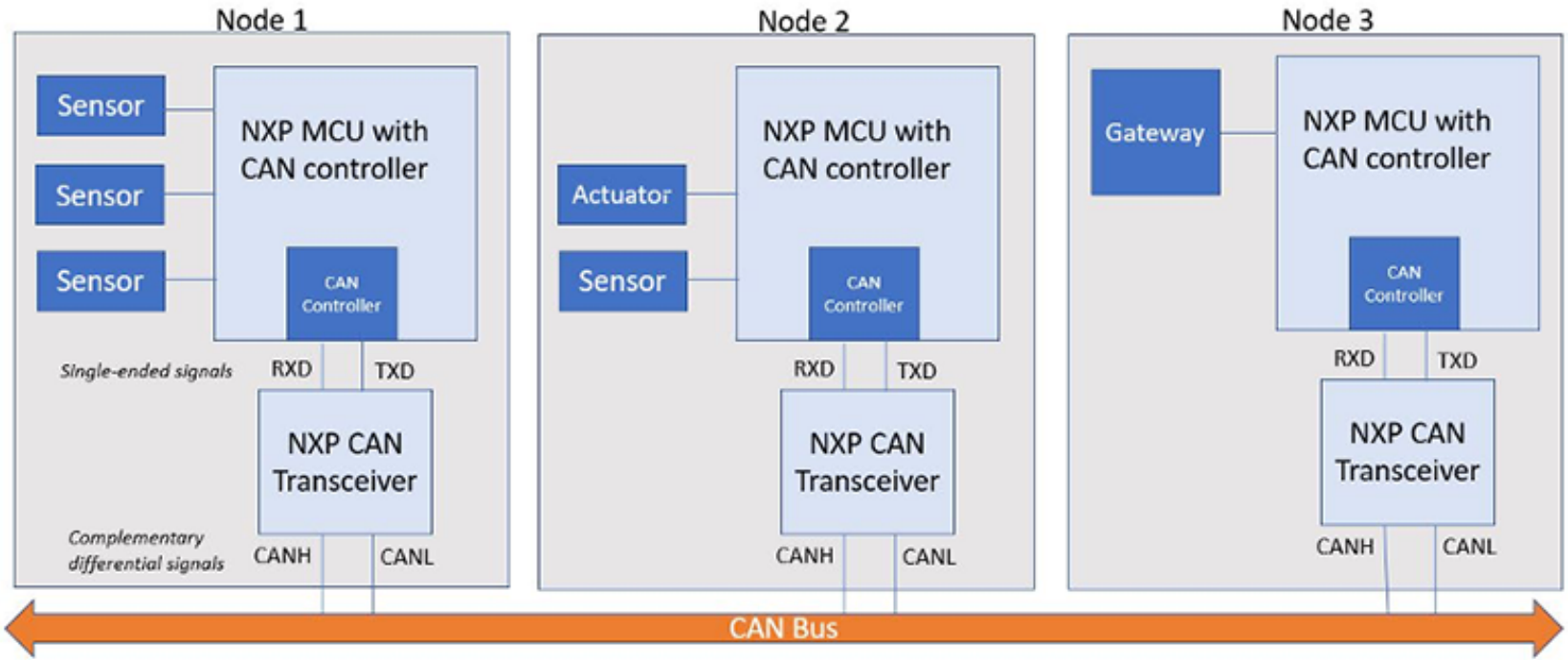



3

[3] https://community.nxp.com/

# CAN (Control Area Network)

- Throughput up to 1 Mb/s.
- Hard real-time - hardware arbitration.
- Multimaster, no routing, broadcast communication.
- Advanced error handling: CRC, temporary and permanent failures.

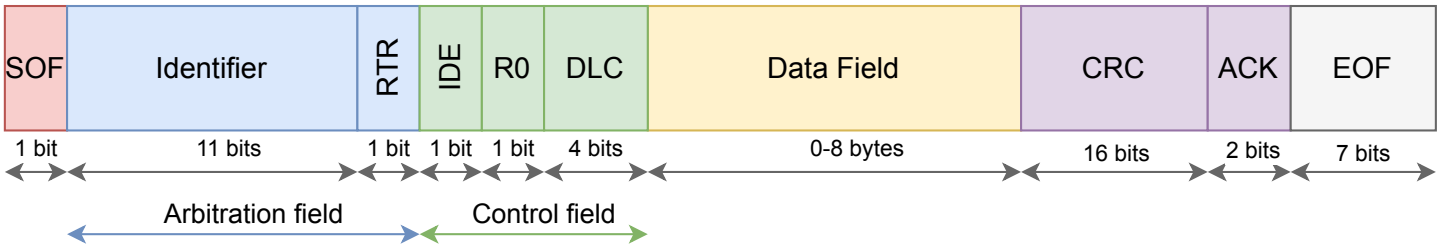



3

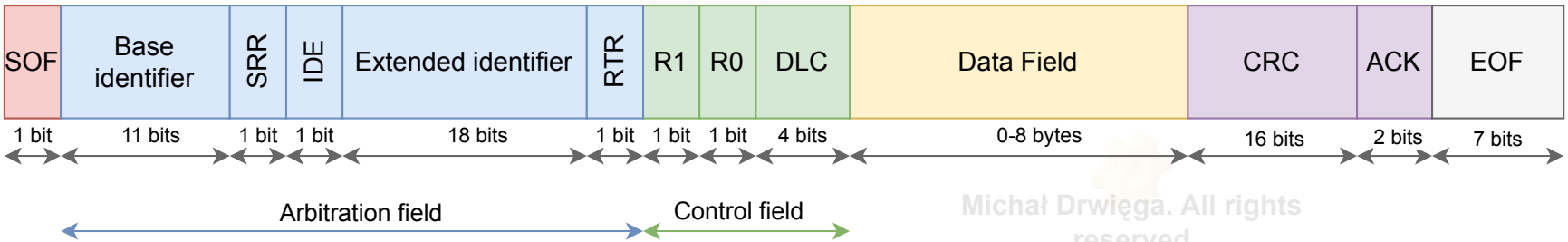
[3] https://community.nxp.com/

# CAN Frame

Standard frame (11-bit identifier)




Extended frame (29-bit identifier)




Michał Drwięga. All rights reserved

Fields:

- **SOF/EOF** – Start/end of frame
- **RTR** – Remote Transmission Request
- **IDE** – Identifier Extension bit
- **R0/R1** – Reserved bits (future use)
- **DLC** – Data Length Code
- **DATA** – Payload
- **CRC** – Error detection
- **ACK** – Acknowledgment

# CAN FD Overview

- **CAN FD (Flexible Data-rate)** is an extension of the classical CAN protocol.
- **Key Features:**
  - Supports **larger data payloads** up to 64 bytes (vs. 8 bytes in classic CAN)
  - Allows **higher bit rates** during the data phase (up to 8 Mbps)
  - Backward compatible with classical CAN for arbitration
- **Advantages:**
  - Increased throughput for modern automotive and industrial applications
  - Efficient use of bus time and reduced latency
- **Applications:** automotive networks, industrial automation, robotics

# USB

History

- **1996** – USB 1.0 (1.5 Mbps, 12 Mbps)
- **2000** – USB 2.0 (up to 480 Mbps)
- **2008** – USB 3.0 *SuperSpeed* (up to 5 Gbps)
- **2019** – USB 3.2 (up to 20 Gbps)

Architecture

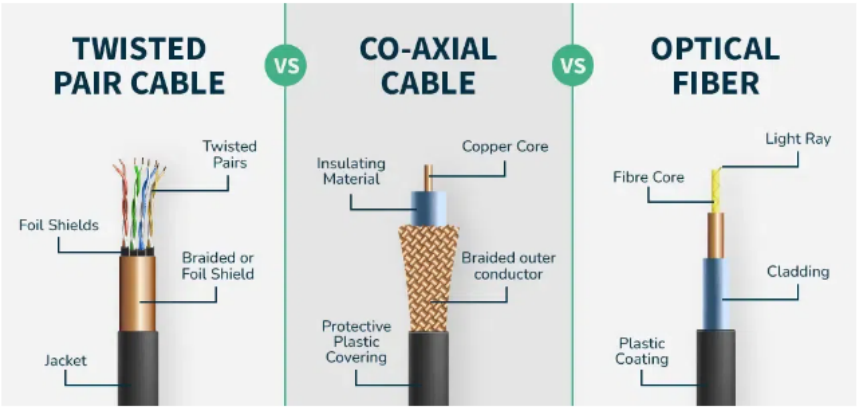
- **Host-device model:**
  - The host controls the bus and initiates all communication
  - Devices respond to host requests
  - Devices can act as *hubs* or *endpoints*
- **Tree structure:**
  - Multiple devices connected via hubs
  - Forms a hierarchical tree structure
  - Each device receives a unique address from the host

# USB (Universal Serial Bus)

- Layers:
  - **Physical layer:**
    - Defines the electrical signaling and connectors.
  - **Link layer:**
    - Handles packet framing, error detection, and flow control.
    - Uses polling to manage communication with devices.
  - **Protocol Layer:**
    - Defines device communication mechanisms
    - Supports multiple transfer types
- Transfer types:
  - *Control* – configuration, commands, and status
  - *Bulk* – large, non-time-critical data (e.g., storage)
  - *Isochronous* – continuous, time-sensitive data (e.g., audio/video)
  - *Interrupt* – small, low-latency events (e.g., keyboard, mouse)
- **Question:** Is USB a real-time communication protocol?

# Ethernet

- Developed by Xerox in 1973-1974.
- Standardized by IEEE 802 for Local Area Networks (LAN).
- Physical layer evolved: coaxial, twisted pair and fiber-optic




4

- Autonegotiation: Procedure by which two connected devices select common transmission parameters, such as speed and duplex mode.
- Hardware Address: 48 bits, unique to each device.
  Example: 00:1A:2B:3C:4D:5E
- Real-time ethernet: RTnet, EtherCAT, etc.

[4] https://https://www.geeksforgeeks.org

# Ethernet - basic data frame

10/100 IEEE 802.3™ Frame

| | |
|---|---|
| 7 octets | Preamble |
| 1 octet | Start Frame Delimiter (SFD) |
| 6 octets | Destination Address (DA) |
| 6 octets | Source Address (SA) |
| 2 octets | Length (≤ 1500)<br>Type (≥ 1536) |
| 46 octets<br>to<br>1500 octets | Client Data (Payload) |
| | Pad (if necessary) |
| 4 octets | Frame Check Sequence (FCS) |

[5]

[5] https://ww1.microchip.com/downloads/en/AppNotes/01120a.pdf

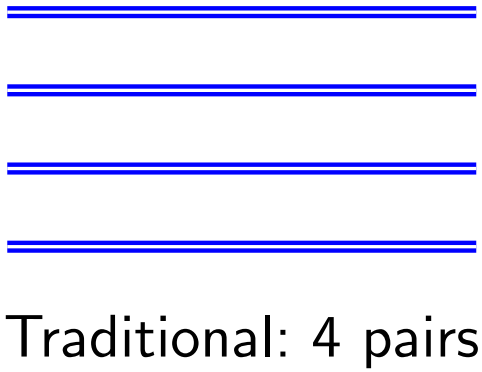
# Single Pair Ethernet (SPE)

## Definition and Basic Concept

### Definition

**Single Pair Ethernet (SPE)** is a technology that transmits data over a single twisted pair of wires, unlike traditional Ethernet which requires 2 or 4 pairs.

- Developed originally for automotive industry
- Now expanding to industrial and building automation
- Enables **borderless IP-based communication** from cloud to sensor



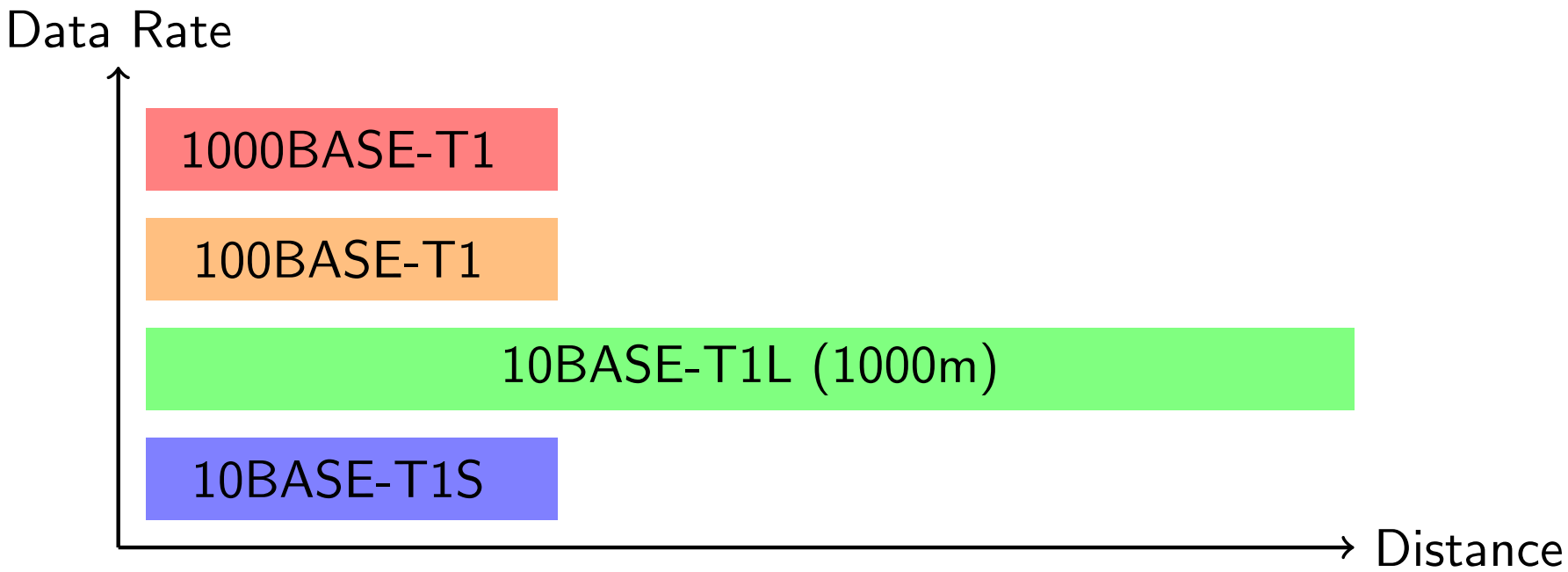
Traditional: 4 pairs

SPE: 1 pair

# SPE Standards

## IEEE 802.3 Portfolio

| Standard | Data Rate | Max Dist. | Application |
|---|---|---|---|
| 10BASE-T1S | 10 Mbit/s | 15 m | Sensors |
| 10BASE-T1L | 10 Mbit/s | **1000 m** | Automation |
| 100BASE-T1 | 100 Mbit/s | 15 m | Automotive |
| 1000BASE-T1 | 1 Gbit/s | 15 m | High-speed apps |

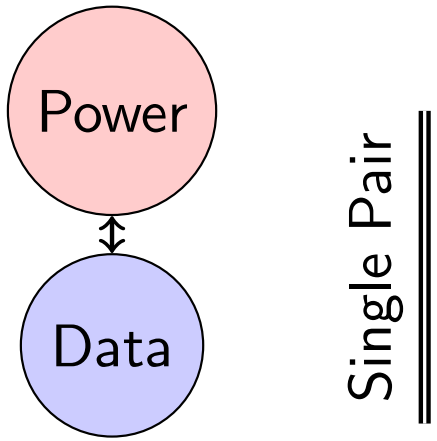


# SPE: Power over Data Line (PoDL)

## PoDL Concept

SPE enables simultaneous transmission of:

- **Data** over the twisted pair
- **Power** (24V / up to 60W)




## Design Consideration

If Power over Single Pair is desired, **different filter components** must be used compared to data-only applications.

- **Weight and cost reduction** with proper design
- Eliminates need for separate power cabling

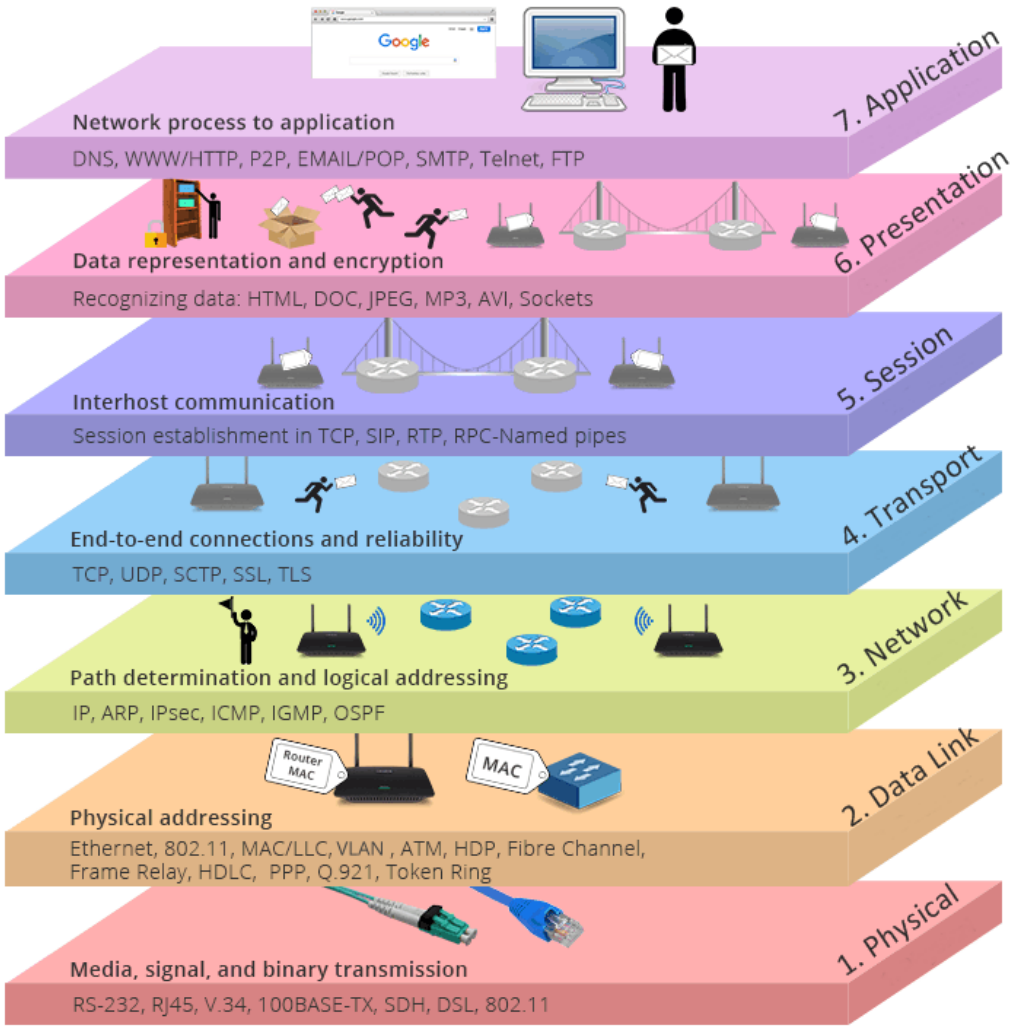
# SPE vs CAN

| **Aspect** | **CAN** | **SPE** |
|---|---|---|
| Communication model | Message-based | Ethernet / IP-based |
| Topology | Bus | Point-to-point or bus |
| Bandwidth | Up to 1 Mbps (CAN FD: 8 Mbps) | 10 Mbps – 1 Gbps |
| Latency behavior | Predictable, bounded | Low, depends on scheduling |
| Determinism | High (priority arbitration) | High with TSN / scheduling |
| Cable length | Up to 500 m (low bitrate) | Up to 1 km (10BASE-T1L) |
| Node count | Limited (bus load) | High (switched network) |
| Scalability | Limited node count | Highly scalable |
| Power delivery | Separate power line | PoDL (data + power) |
| Protocol stack | Simple (CAN frames) | Full Ethernet / IP stack |
| Typical use | Sensors, actuators | sensors, compute nodes |

# OSI (Open Systems Interconnection) Model

- OSI model characterizes and standardizes how different software and hardware components involved in a network communication should interact with one another.
- The earliest model of computer networks developed by International Organization of Standarization (ISO) in 1984
- OSI model does not include any specifications for network implementation
- It defines seven-layer architecture for communication system

# OSI Model




6

[6] https://community.fs.com/

# OSI Model

7 **Application** (Data) – High-level APIs, including resource sharing, remote file access, directory services

6 **Presentation** (Data)– Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption

5 **Session** (Data) – Managing communication sessions: synchronization, termination of connectivity

4 **Transport** (Segments) – Transmission of data segments between points on a network, including segmentation

3 **Network** (Packet/Datagram) – Structuring and managing a multi-node network, including addressing, routing and traffic control

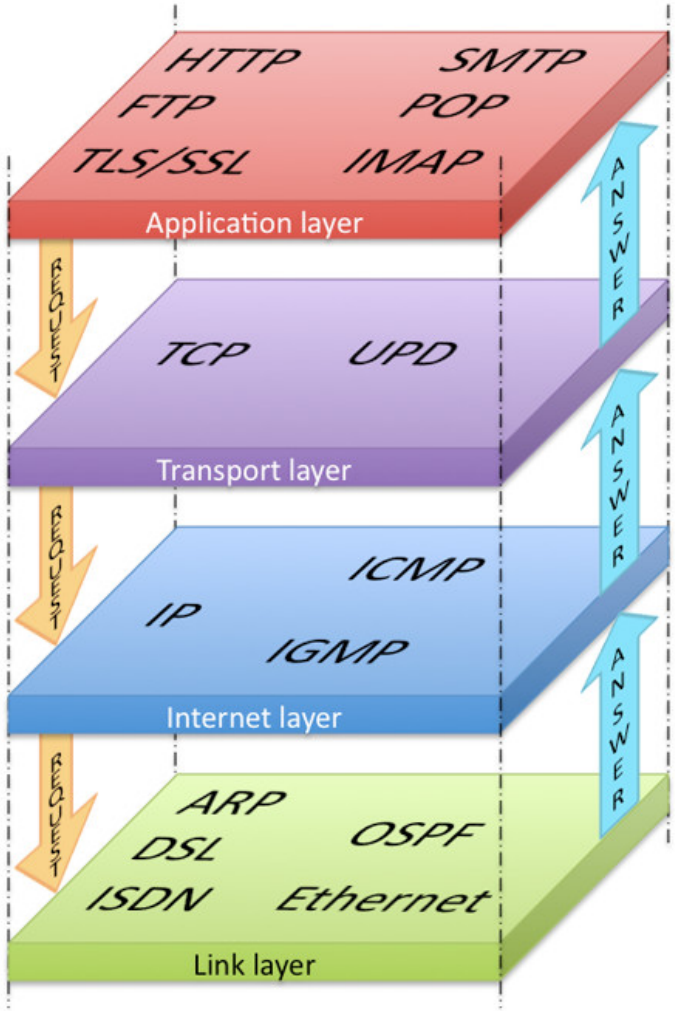
2 **Data link** (Bit/Frame) – Reliable transmission of data frames between two nodes connected by a physical layer

1 **Physical** (Bit) – Defines the electrical and physical requirements for networked devices with control of the transmission

# TCP/IP Protocol Stack

**TCP/IP Layers:**

- **Application** – HTTP, SMTP, RTSP
- **Transport** – TCP (reliable), UDP (low latency)
- **Internet** – IP (routing, addressing)
- **Link / Physical** – Ethernet, Wi-Fi, SPE

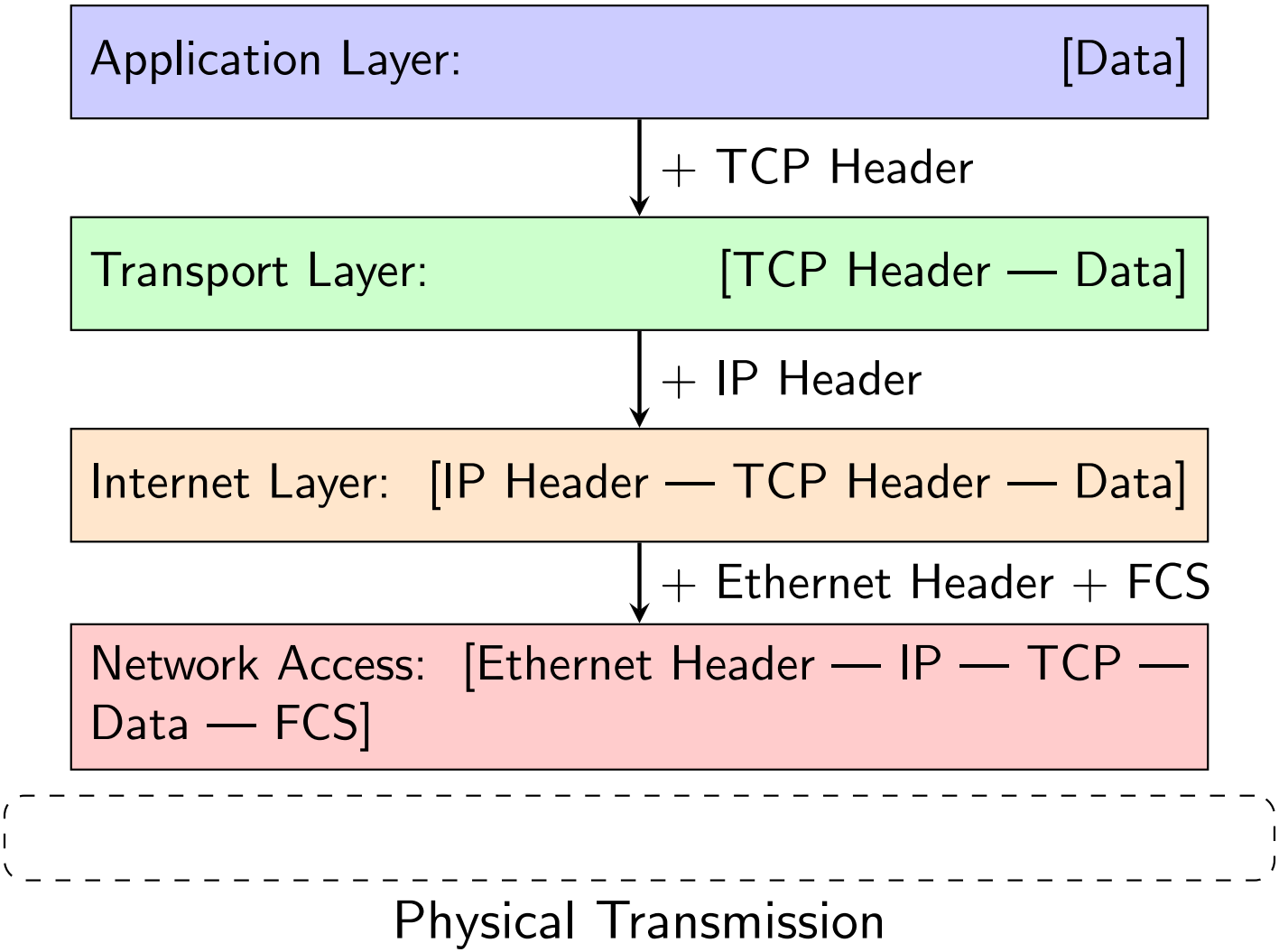



*a*

[a] https://linkedin.com/

# TCP/IP Stack

## Data Encapsulation Example

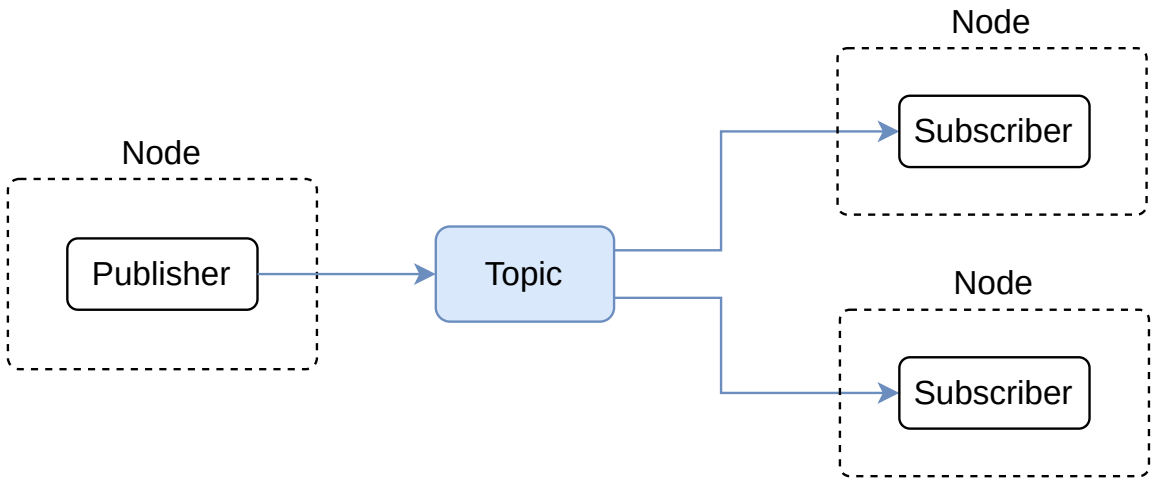


# Communication models in distributed systems

# Communication models: Publish-Subscribe Model

## Fundamental Principles

- **Event-Driven Architecture:** Communication based on events rather than direct requests
- **Components:**
  - **Publishers:** Produce and publish events without knowing consumers
  - **Subscribers:** Express interest in specific event types



# MQTT: Lightweight IoT Protocol

## What is MQTT?

- **MQTT (Message Queuing Telemetry Transport)** Lightweight, publish-subscribe network protocol for IoT and M2M communication.
- Designed for **low-bandwidth, high-latency, or unreliable networks**.
- Emphasizes **simplicity, reliability, and small message overhead**.
- It uses TCP/IP.

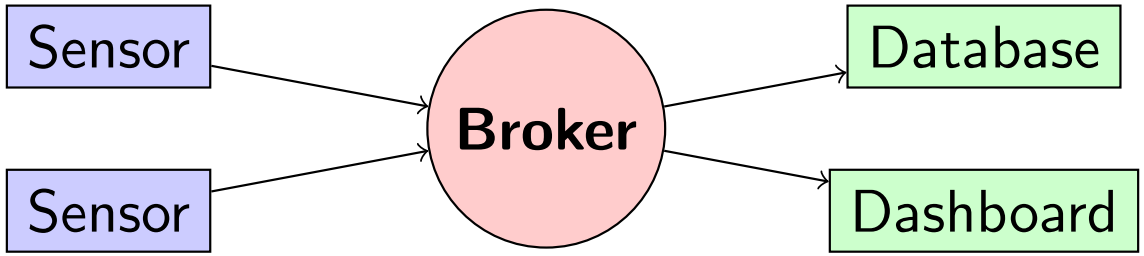
## Key Features

- Broker-based architecture
- Topic-based filtering
- Three QoS levels

## Common Uses

- Smart home sensors
- Industrial monitoring
- Vehicle telemetry
- Healthcare devices

# MQTT Architecture

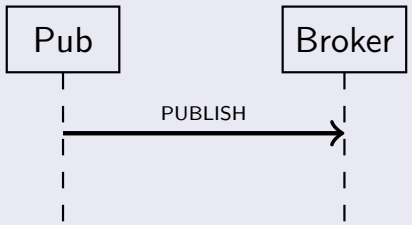



Broker mediates all communication

- **Publishers:** Sends messages to a topic.
- **Subscribers:** Receives messages from topics of interest.
- **Broker:** Central server that routes messages between publishers and subscribers.

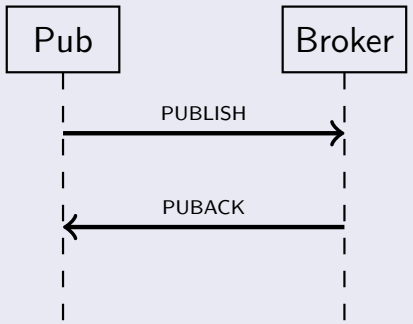
# Quality of Service (QoS)

## QoS 0: At most once



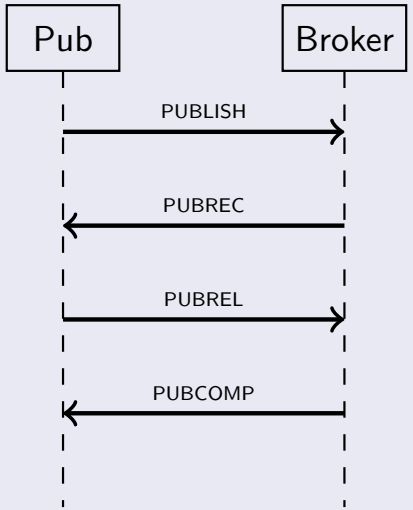

- Fire and forget
- No acknowledgment
- Fastest, least reliable

## QoS 1: At least once




- Guaranteed delivery
- Confirmation required
- May have duplicates

## QoS 2: Exactly once




- Four-step handshake
- Guaranteed no duplicates
- Highest overhead

# Advanced Features

## Retained Messages

- Broker stores last message on a topic
- New subscribers get it immediately
- **Example:** Last known temperature

## Last Will and Testament (LWT)

- Client sets "will message" on connect
- Broker publishes if client dies unexpectedly
- **Example:** Detect device failures

## Clean Session

- **True:** No state persistence (stateless)
- **False:** Subscriptions and messages persist (for offline clients)

# MQTT Security

**Security Layers**

- TLS encryption
- Username/password
- Client certificates
- Topic ACLs

**Ports**

- 1883: Unencrypted
- 8883: TLS
- 8080: WebSocket
- 8083: Secure WS

**Best Practice**

Always use TLS in production + unique credentials per device

# MQTT Topics

## Topic Hierarchy

`house/room1/temperature`

### Single-Level Wildcard (+)

- `house/+/temperature`
- Matches: `house/kitchen/temperature`
- Matches: `house/bedroom/temperature`

### Multi-Level Wildcard (#)

- `house/#`
- Matches: `house/kitchen/temp`
- Matches: `house/bedroom/humidity`

# MQTT Brokers and Clients

**Open Source Brokers**

- **Mosquitto:** Lightweight, reference broker
- **EMQX:** Scalable, cluster-capable
- **VerneMQ:** High-availability, distributed

**Cloud / Commercial Brokers**

- **HiveMQ:** Enterprise-grade broker
- **AWS IoT Core:** AWS integrated broker
- **Azure IoT Hub:** Microsoft cloud broker

**Popular MQTT Clients (Libraries)**

- **Eclipse Paho:** Official MQTT client library; supports multiple languages (Python, Java, C, JavaScript)
- **MQTT.js:** Node.js client for MQTT

# Mosquitto: Installation & Basic Setup

## Installation

Linux (Debian/Ubuntu): `sudo apt-get install mosquitto mosquitto-clients`

## Starting the Broker

- **Foreground mode:**
  `mosquitto -v`
- **Background mode:**
  `mosquitto -d`
- **Custom port:**
  `mosquitto -p 1884`
- **With config:**
  `mosquitto -c mosquitto.conf`

## Service Management

- `sudo systemctl start mosquitto`
- `sudo systemctl stop mosquitto`
- `sudo systemctl restart mosquitto`
- `sudo systemctl status mosquitto`

# Mosquitto: Publishing Messages

## mosquitto_pub - Publish messages to broker

Basic publish: `mosquitto_pub -h localhost -t test -m "Hello"`

With QoS level: `mosquitto_pub -h localhost -t test -m "m" -q 1`

Retained message: `mosquitto_pub -h localhost -t test -m "m" -r`

## Example:

```
mosquitto_pub -h localhost
       -t sys/temperature -m "23.5" -q 1 -r
```

(Publishes retained temperature with QoS 1)

# Mosquitto: Subscribing to Topics

## `mosquitto_sub` - Subscribe to topics

Basic subscription: `mosquitto_sub -h localhost -t test`

Verbose output: `mosquitto_sub -h localhost -t test -v` (shows topic names)

Multiple topics: `mosquitto_sub -h localhost -t topic1 -t topic2`

Single-level wildcard (+): `mosquitto_sub -h localhost -t 'home/+/temperature'`

Multi-level wildcard (#): `mosquitto_sub -h localhost -t 'home/#'`

## Example:

`mosquitto_sub -h localhost -v -t 'sensors/#'`

(Subscribes to all sensor data with verbose output)

# VDA 5050: Standardizing Robot Fleet Communication

## What is VDA 5050[7]?

- **Standard interface** for communication between autonomous vehicles (AGV/AMR) and a master control system.
- Developed by the German Association of the Automotive Industry (VDA) and VDMA.

## Technical Foundation

- **MQTT** as the communication protocol (lightweight, scalable).
- **JSON** as the data format.
- Pre-defined topics: order, state, visualization, connection.
- QoS 0 for most topics, QoS 1 for connection topic.

## Key Benefits

- **Plug & Play**: Reduced integration time for new vehicles.
- **Mixed fleets**: Single control system manages diverse robot types.

[7] https://www.vda.de/en/topics/automotive-industry/vda-5050

# MQTT Python publisher example

```python
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc, properties):
    if rc == 0:
        print("Connected successfully to the broker.")
    else:
        print(f"Connection failed with code {rc}")

if __name__ == '__main__':
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    client.on_connect = on_connect
    client.connect("localhost", 1883, 60)
    client.loop_start()

    while True:
        client.publish("temperature", 10.0)
        time.sleep(1)

    client.disconnect()
```

# MQTT Python subscriber example

```python
import paho.mqtt.client as mqtt

def on_message(client, userdata, msg):
    print(f'Received: {msg.topic} {msg.qos} {msg.payload}')

def on_connect(client, userdata, flags, rc, properties):
    print(f"Connected with result code {rc}")

if __name__ == '__main__':
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    client.on_connect = on_connect
    client.on_message = on_message

    client.connect("localhost", 1883, 60)
    client.subscribe("temperature")

    while client.loop() == 0:
        pass

    client.disconnect()
```

# Summary

- Embedded networking
- OSI model
- MQTT

# The end

Thank you for your attention!