Communication Protocols in Distributed Systems

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

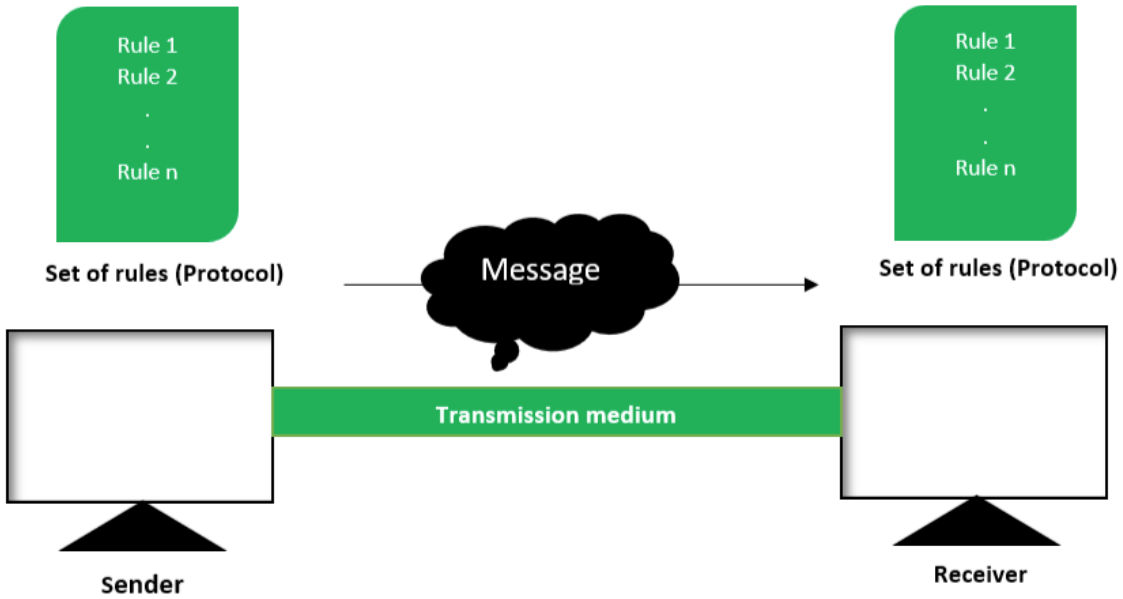
Wrocław University
of Science and Technology

Content

- Basics of data communication
- OSI model
- Embedded networking
- Communication paradigms
- Data representation and serialization

Basics of Data Communication

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



1

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

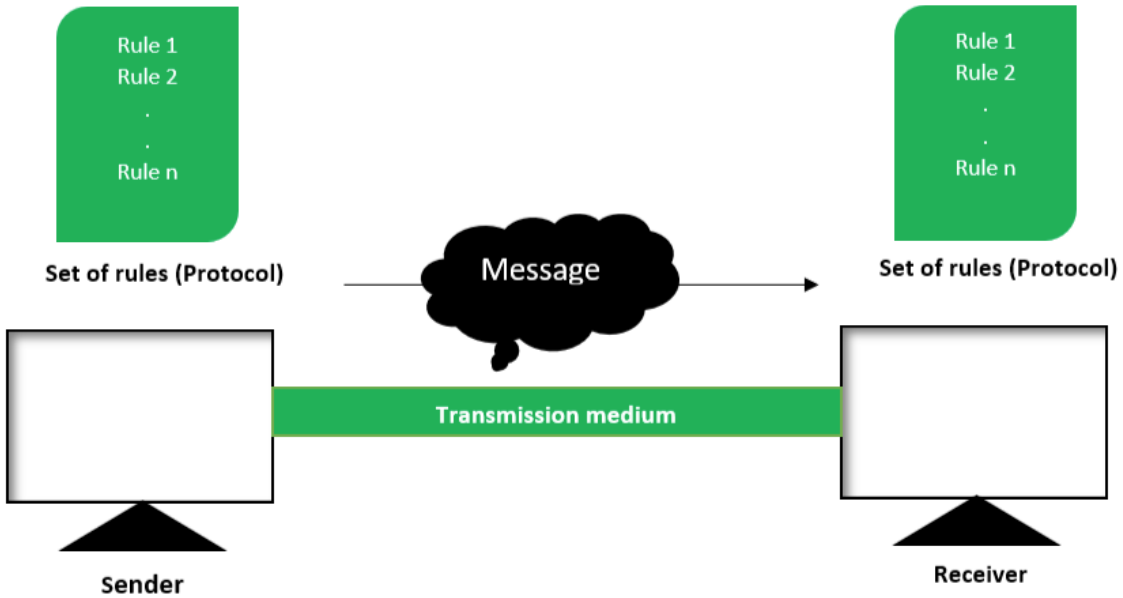
Basics of Data Communication

System components

Message - information to be communicated

Sender - the device that sends the message

Receiver - the device that receives the message



1

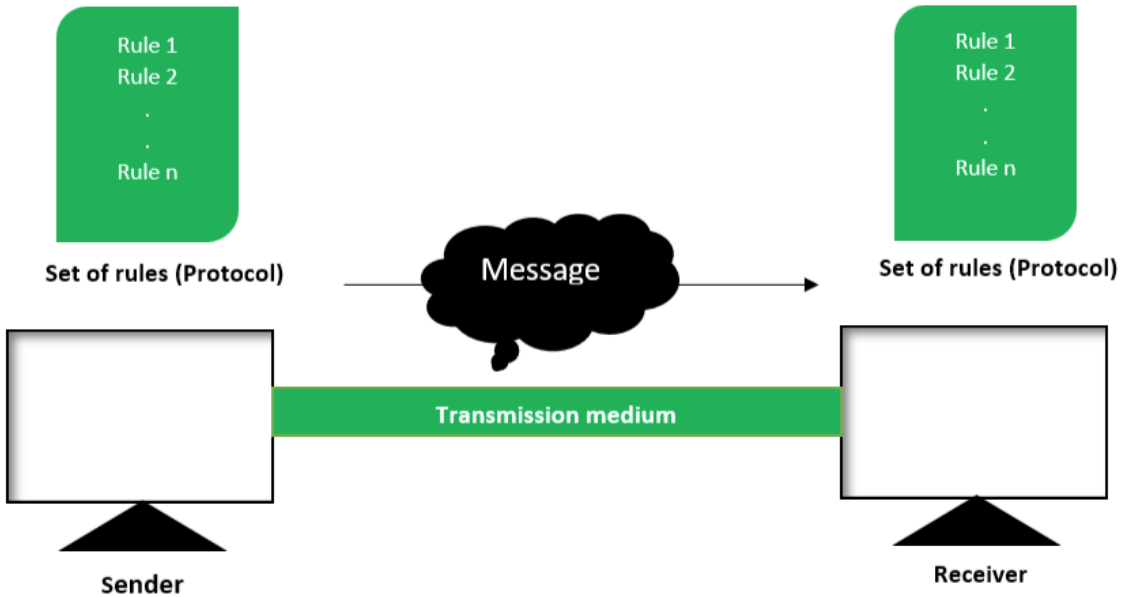
[1] https://www.geeksforgeeks.org/

Basics of Data Communication

System components

Transmission medium - the physical path by which a message travels

Protocol - a set of different rules that represents the agreement between devices



1

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

Basics of Data Communication

Effectiveness of data communication

Delivery - data should be delivered to the correct destination.

Accuracy - data should be delivered without any errors.

Timeless - data should be delivered as they are produced - without significant delay

Basics of Data Communication

Modes of communication

Simplex - Communication is one-directional.

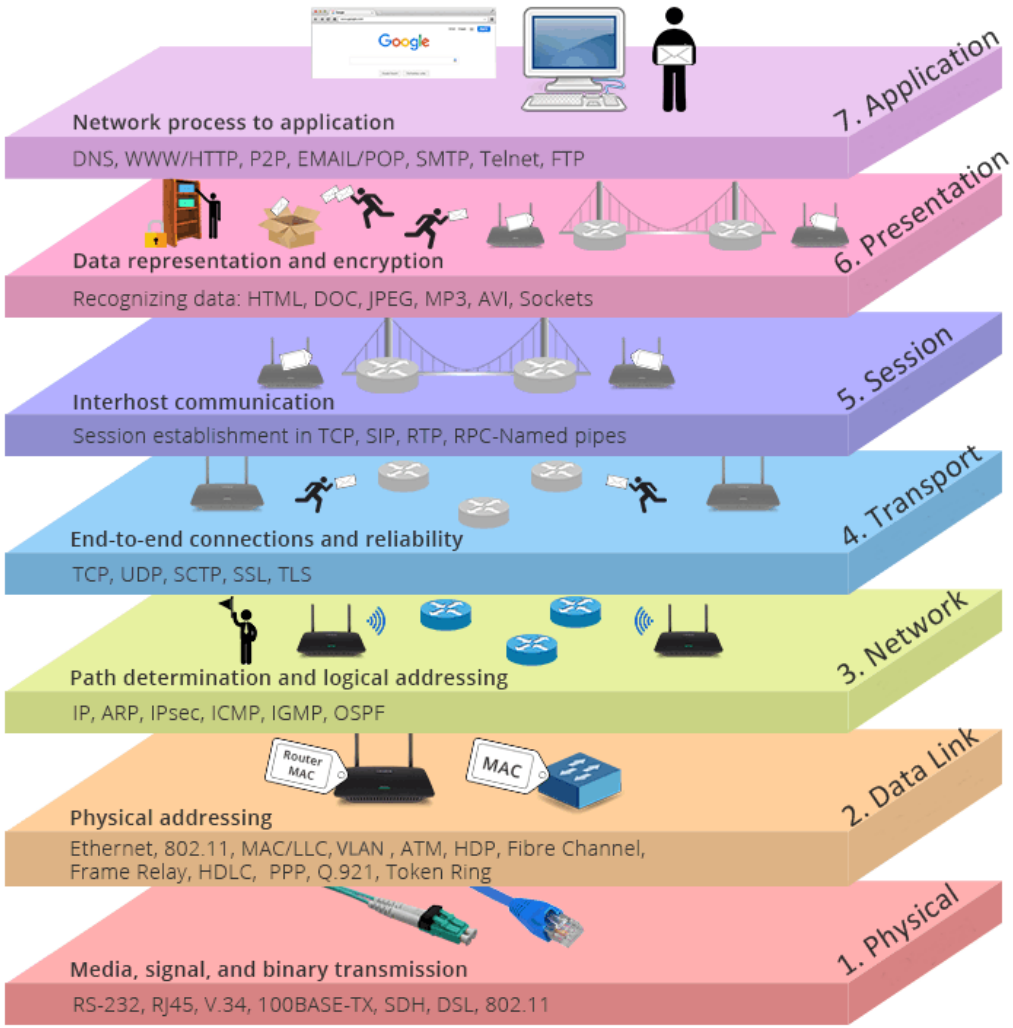
Half-duplex - Each node can transmit and receive, but **not at the same time**.

Full-duplex - In this mode both nodes can transmit data **simultaneously** in **both directions**.

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



2

[2] 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

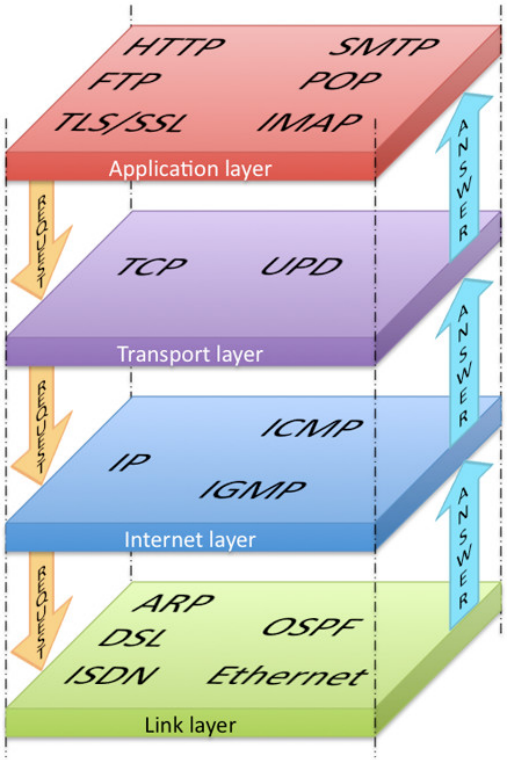
4 **Transport** (Segments) – Transmission of data segments between points on a network, including segmentation

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



[3]

[3] https://linkedin.com/

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[4]
- **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

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

RS-232 vs. RS-485

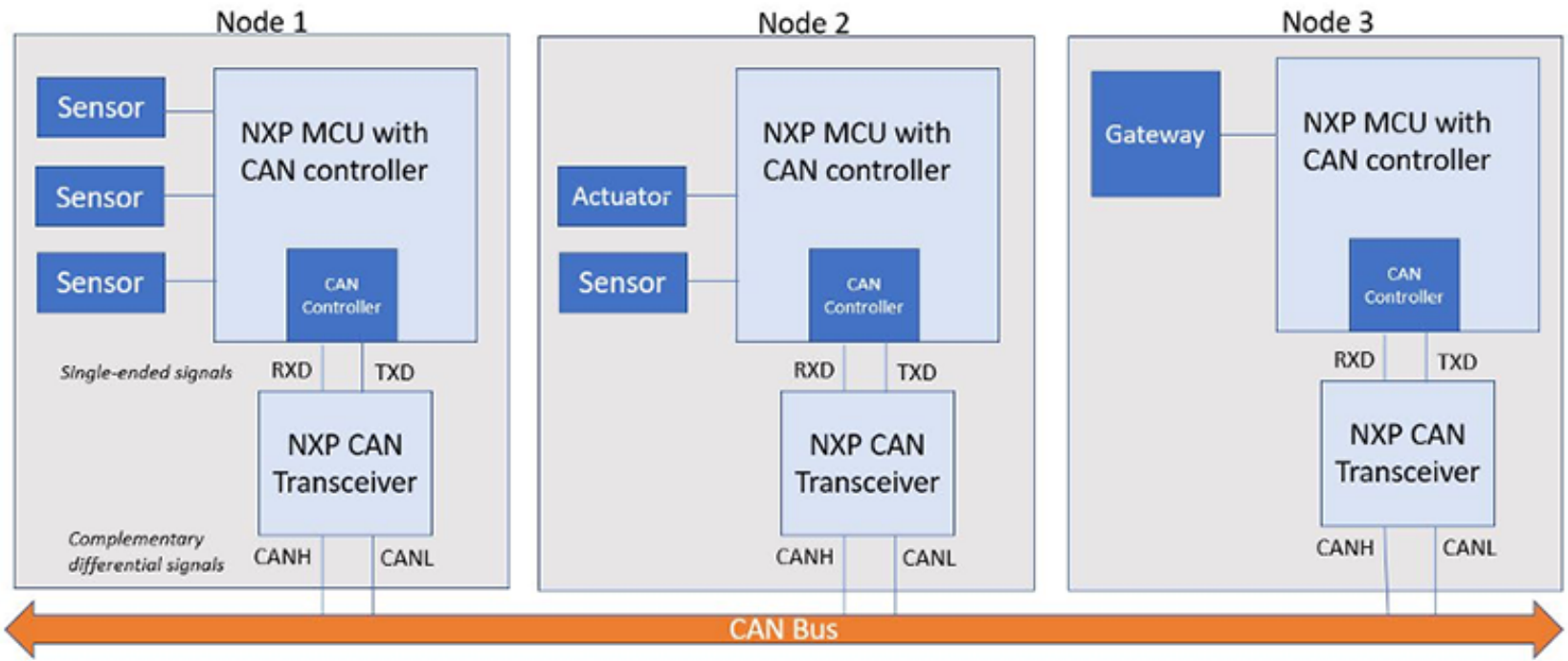
- Communication type:
  - RS-232: Point-to-point connection.
  - RS-485: Multipoint communication on a shared bus.
- Signaling method:
  - RS-232: Single-ended signaling; more noise-prone.
  - RS-485: Differential signaling; better noise immunity.
- Network topology:
  - RS-232: Star or point-to-point topology.
  - RS-485: Bus topology; supports more devices.
- Use cases:
  - RS-232: Short-distance applications (e.g., peripherals).
  - RS-485: Long-distance, robust communication (e.g., industrial systems).

I^2C and SPI

- I^2C (Inter-Integrated Circuit):
  - Two-wire protocol (SDA and SCL).
  - Supports multiple masters and slaves on the same bus.
  - Addressing: Uses 7-bit or 10-bit addressing for devices.
  - Speed: Standard speeds are 100 kbps (standard mode) and 400 kbps (fast mode), with high-speed mode up to 3.4 Mbps.
  - Uses acknowledgment for data transmission reliability.
  - Suitable for short-distance communication in embedded systems (e.g., sensors, EEPROMs).
- SPI (Serial Peripheral Interface):
  - Four-wire protocol (MOSI, MISO, SCK, and SS).
  - Typically point-to-point (one master, one slave).
  - No addressing; each slave device requires a separate select line.
  - Speed: Generally faster than I2C, ranging from 1 Mbps up to several tens of Mbps.
  - Full-duplex communication allows simultaneous data transmission and reception.
  - Ideal for high-speed communication in applications like SD cards and displays.

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.

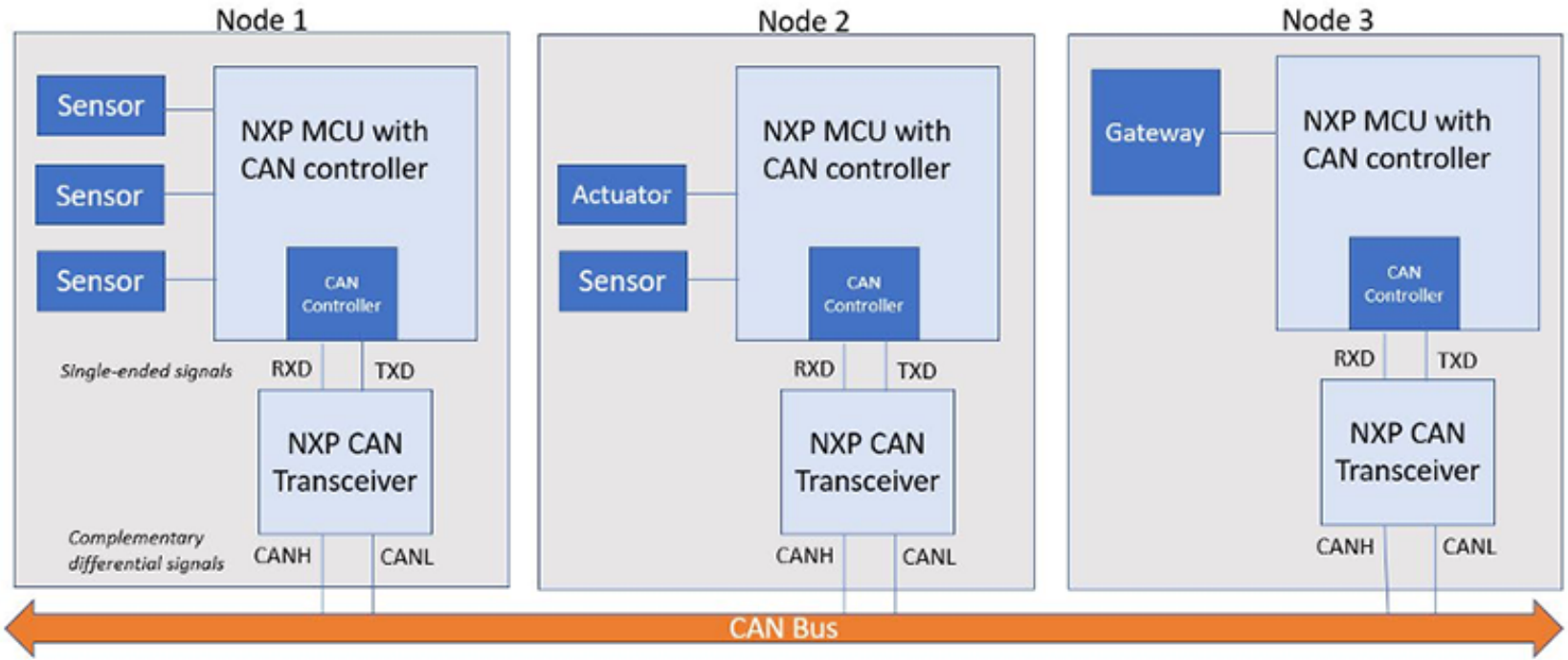


5

[5] 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.



5

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

USB

History

- 1996: USB 1.0 released (1.5 Mbps & 12 Mbps).
- 2000: USB 2.0 introduced (up to 480 Mbps).
- 2008: USB 3.0 with SuperSpeed (up to 5 Gbps).
- 2019: USB 3.2 supports 20 Gbps.

Architecture

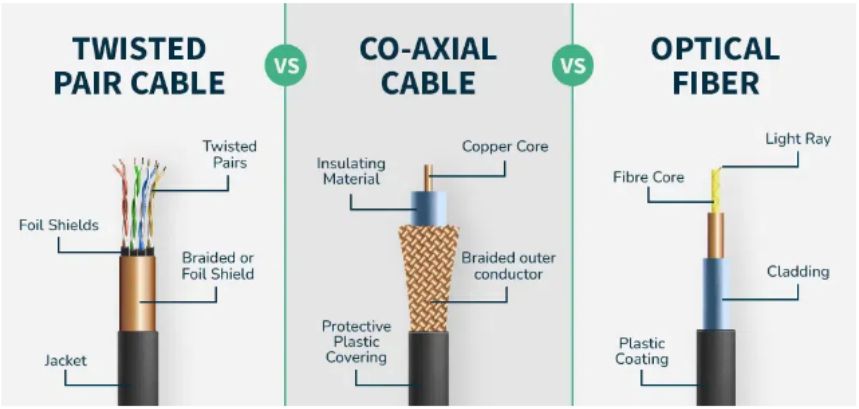
- **Host-device model:**
  - The host initiates communication, manages bandwidth, and controls the bus.
  - Devices respond to the host, can be hubs or endpoints.
- **Tree structure:**
  - A single host can connect multiple devices via hubs, forming a tree-like topology.
  - Each device has a unique address assigned by 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:**
    - Supports different communication types
- Communication types:
  - *Control Transfers* for command and status.
  - *Bulk Transfers* for large data volumes.
  - *Isochronous Transfers* for time-sensitive data.
  - *Interrupt Transfers* for devices needing quick responses.
- 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



[6]

- 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.

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

Ethernet - basic data frame

10/100 IEEE 802.3™ Frame

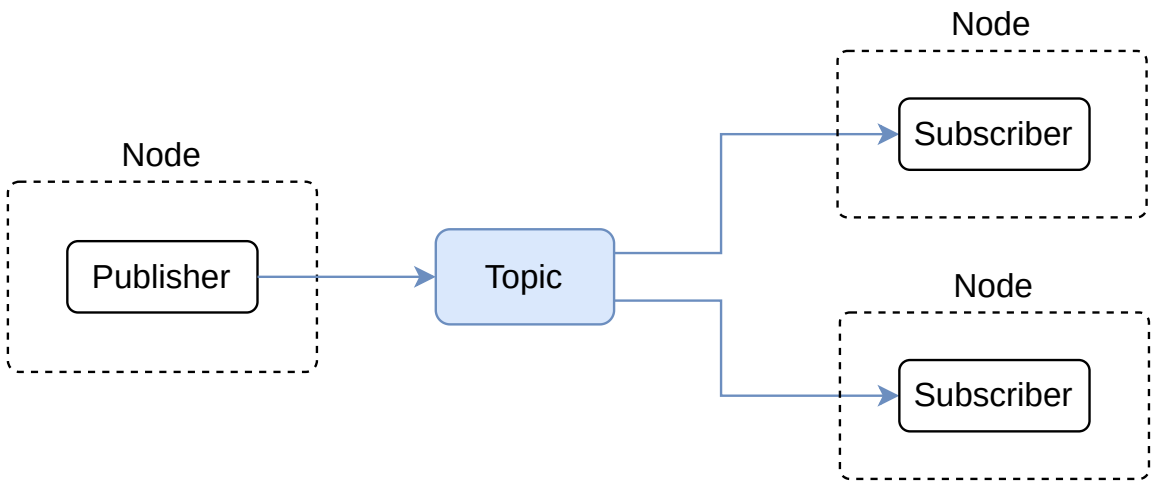
7 octets	Preamble
1 octet	Start Frame Delimiter (SFD)
6 octets	Destination Address (DA)
6 octets	Source Address (SA)
2 octets	Length (≤ 1500) Type (≥ 1536)
46 octets to 1500 octets	Client Data (Payload)
	Pad (if necessary)
4 octets	Frame Check Sequence (FCS)

[7]

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

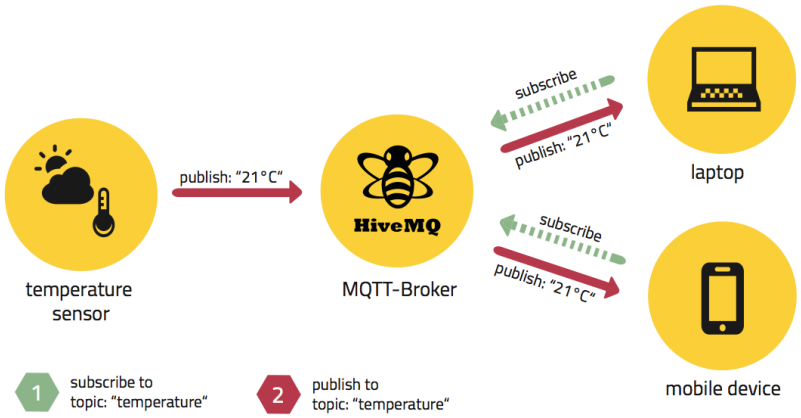
Communication models: Publish-Subscribe Model

- Publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions.
- Notifications are sent asynchronously
- Message-based communication



Distributed Communication - MQTT

- MQTT (MQ Telemetry Transport) is a lightweight publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments.
- One of the main protocols for IoT (Internet of things) deployments
- Clients: Paho Python client, Node.js MQTT Clients, C++ Client
- Broker (Server), the most popular is Mosquitto Broker, commercial broker HiveMQ
- It uses TCP/IP

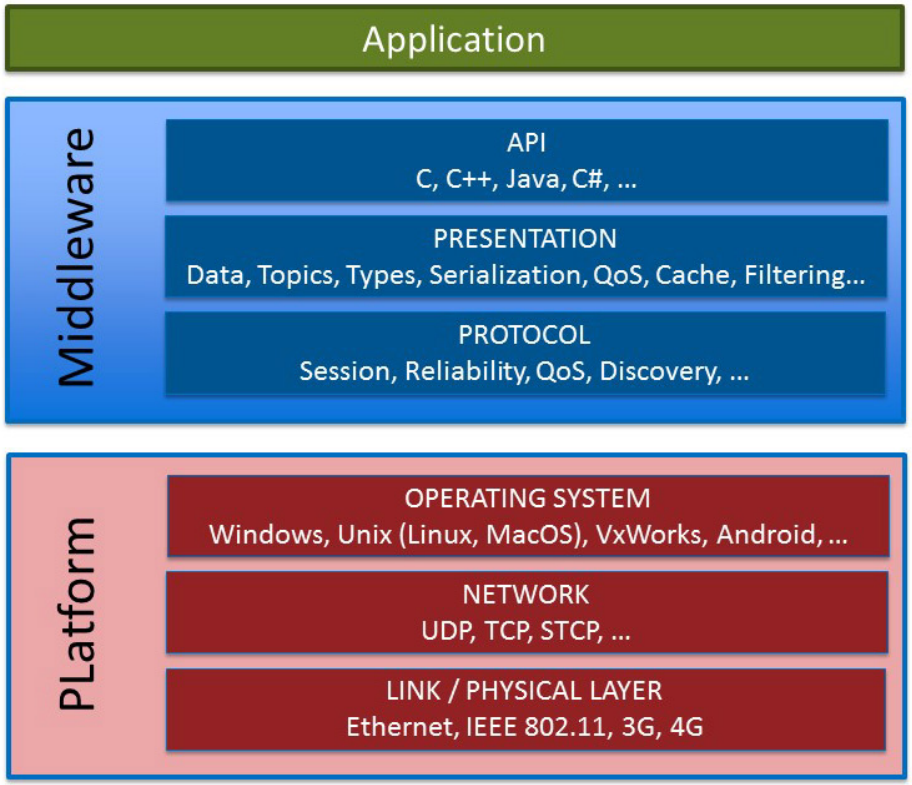


[8]

[8] https://www.eclipse.org/

Data Distribution Service (DDS)

- The DDS Middleware is a software layer that abstracts the Application from the details of the operating system, network transport, and low-level data formats.

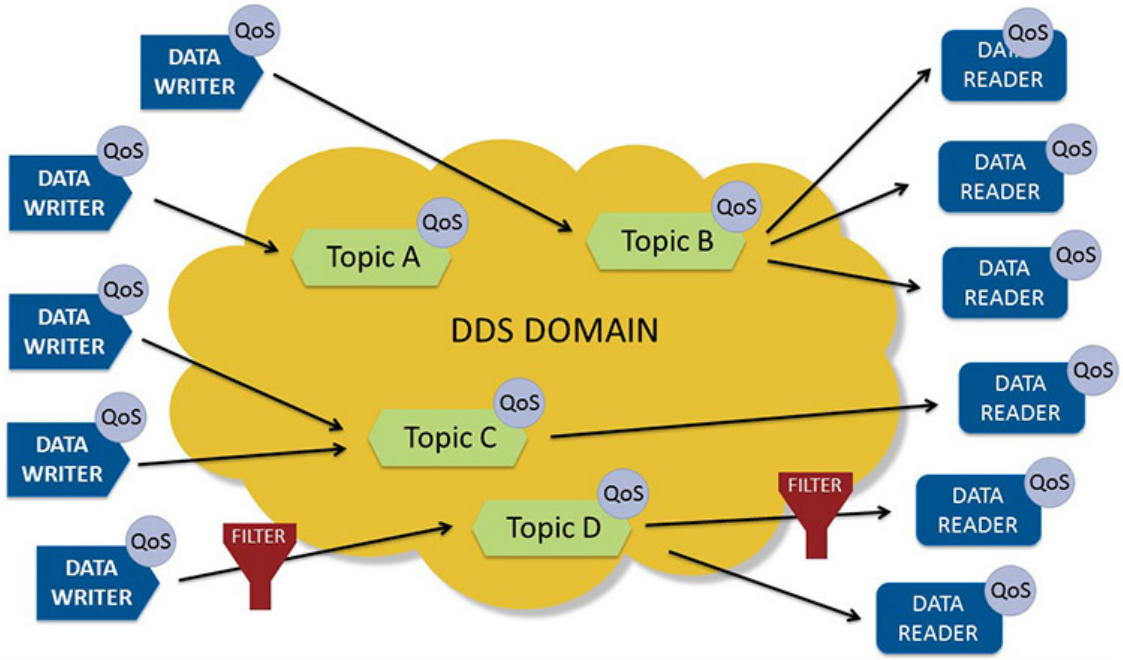


9

[9] https://www.dds-foundation.org/

Data Distribution Service (DDS)

- Implements a publish-subscribe communication pattern.
- Utilizes **Interface Description Language (IDL)** for defining message structures.
- Supports flexible **Quality of Service (QoS)** for reliability, liveliness, and security.



[9]

[9] https://www.dds-foundation.org/

Data Distribution Service (DDS)

- **Dynamic Discovery**: Automatically finds communication endpoints at runtime, enabling plug-and-play functionality.
  - Applications don't need to pre-configure endpoints as DDS discovers them automatically.
  - DDS identifies whether an endpoint is publishing or subscribing, the data type, and communication characteristics.
  - DDS participants can be on the same machine or across a network.
- DDS is an industry standard, implemented by vendors like RTI, eProsima, and Eclipse.[9]

[9] https://docs.ros.org/

Inter-Process Communication (IPC)

What is a process?

A **process** is a program in execution, and each process has its own address space, which comprises the memory locations that the process is allowed to access.

Communication - one host vs multiple hosts

- Processes on the same machine
  - Shared memory
  - Pipes (named, unnamed)
  - Sockets
  - Signals
- Processes on different machines in network
  - Network sockets

Inter-Process Communication (IPC) - sockets

- Relatively low-level support for communication
  - Direct access to internet protocols (Socket API)
- Socket is a communication end-point to which an application can write or read data
- Socket abstraction is used to send and receive messages from the transport layer of the network
- Each socket is associated with a particular type of transport protocol
  - UDP Socket – Connection-less and unreliable communication
  - TCP Socket – Connection-oriented and reliable communication

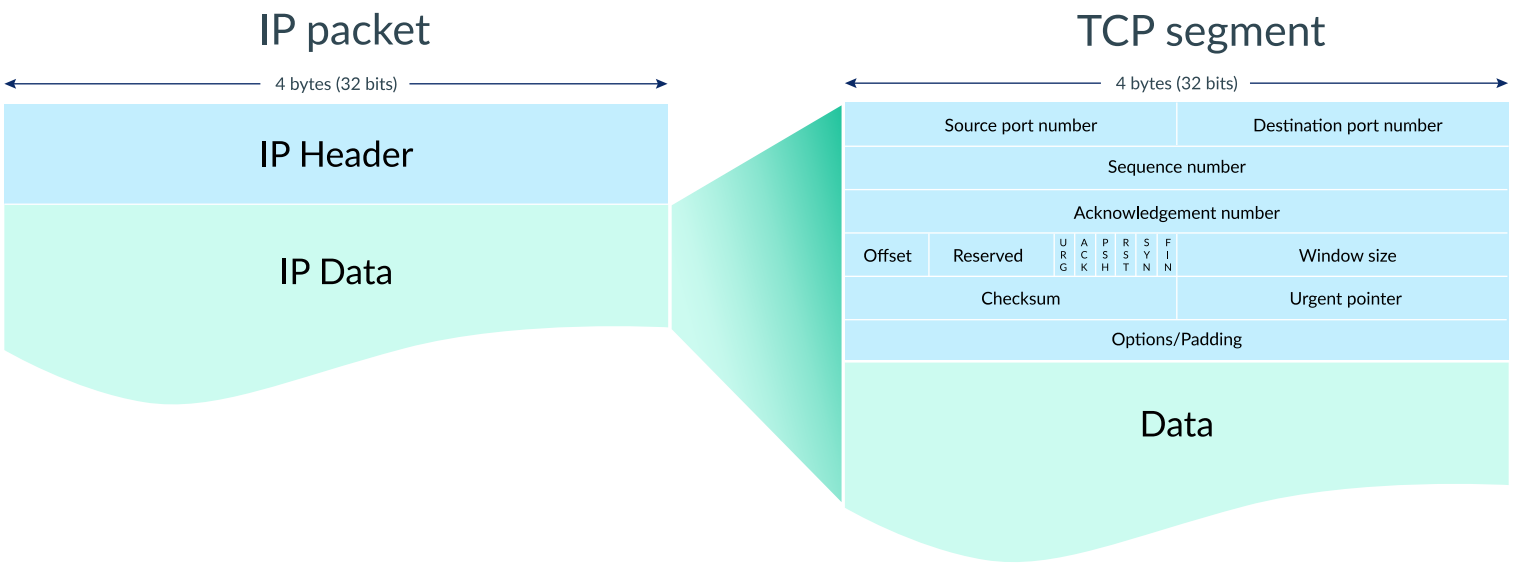
Inter-Process Communication: UDP Sockets

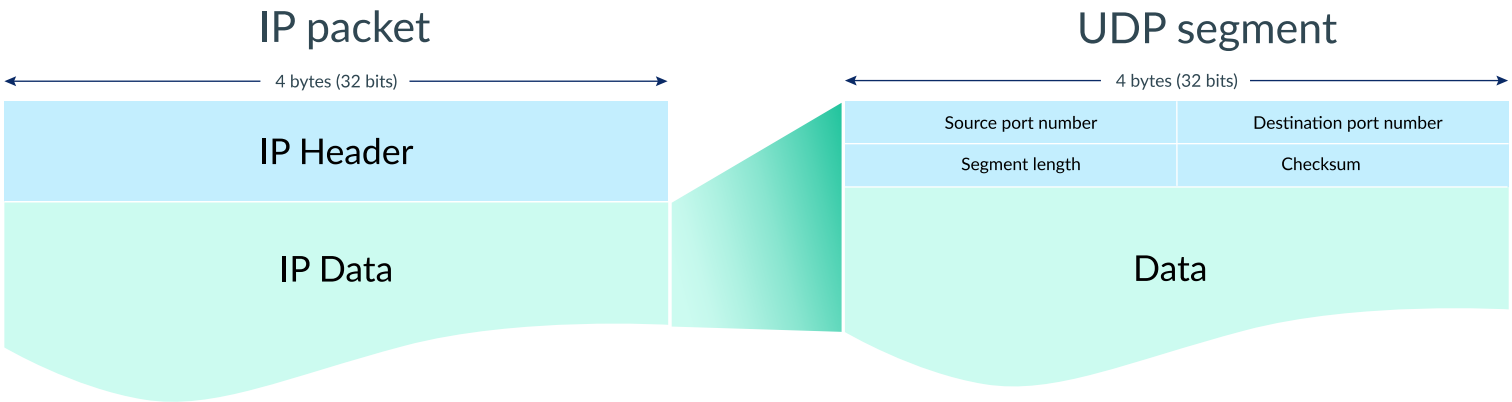
- **Connectionless communication**: UDP enables direct data transmission without establishing a connection, meaning no acknowledgements or retries.
- **Message delivery**: Messages may be delivered out of order, and the communication does not guarantee reliability.
- **Fragmentation required**: The sender must explicitly fragment long messages into smaller chunks before transmission.
- **Buffer management**: The receiver should allocate a buffer large enough to accommodate the sender's message to avoid overflow.

Inter-Process Communication: TCP Sockets

- **In-order delivery**: TCP sockets maintain the order of messages sent between applications.
- **Message size**: Applications can send messages of any size without restrictions.
- **Reliable communication**: Uses acknowledgements and retransmissions to guarantee message delivery.
- **Congestion control**: Regulates the sender's rate to prevent network overload and ensure smooth communication.

TCP vs UDP


IP packet
4 bytes (32 bits)
IP Header
IP Data
TCP segment
4 bytes (32 bits)
Source port number
Destination port number
Sequence number
Acknowledgement number
Offset
Reserved
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options/Padding
Data


IP packet
4 bytes (32 bits)
IP Header
IP Data
UDP segment
4 bytes (32 bits)
Source port number
Destination port number
Segment length
Checksum
Data

ZeroMQ: High-Performance Messaging Library

- **Asynchronous messaging**: Designed for efficient communication in distributed or concurrent applications.
- **Why "zero"?** Zero brokers (brokerless architecture) and minimal latency for optimal performance.
- **Multiple messaging patterns**:
  - Request-Reply
  - Publish-Subscribe
  - Push-Pull

[10]

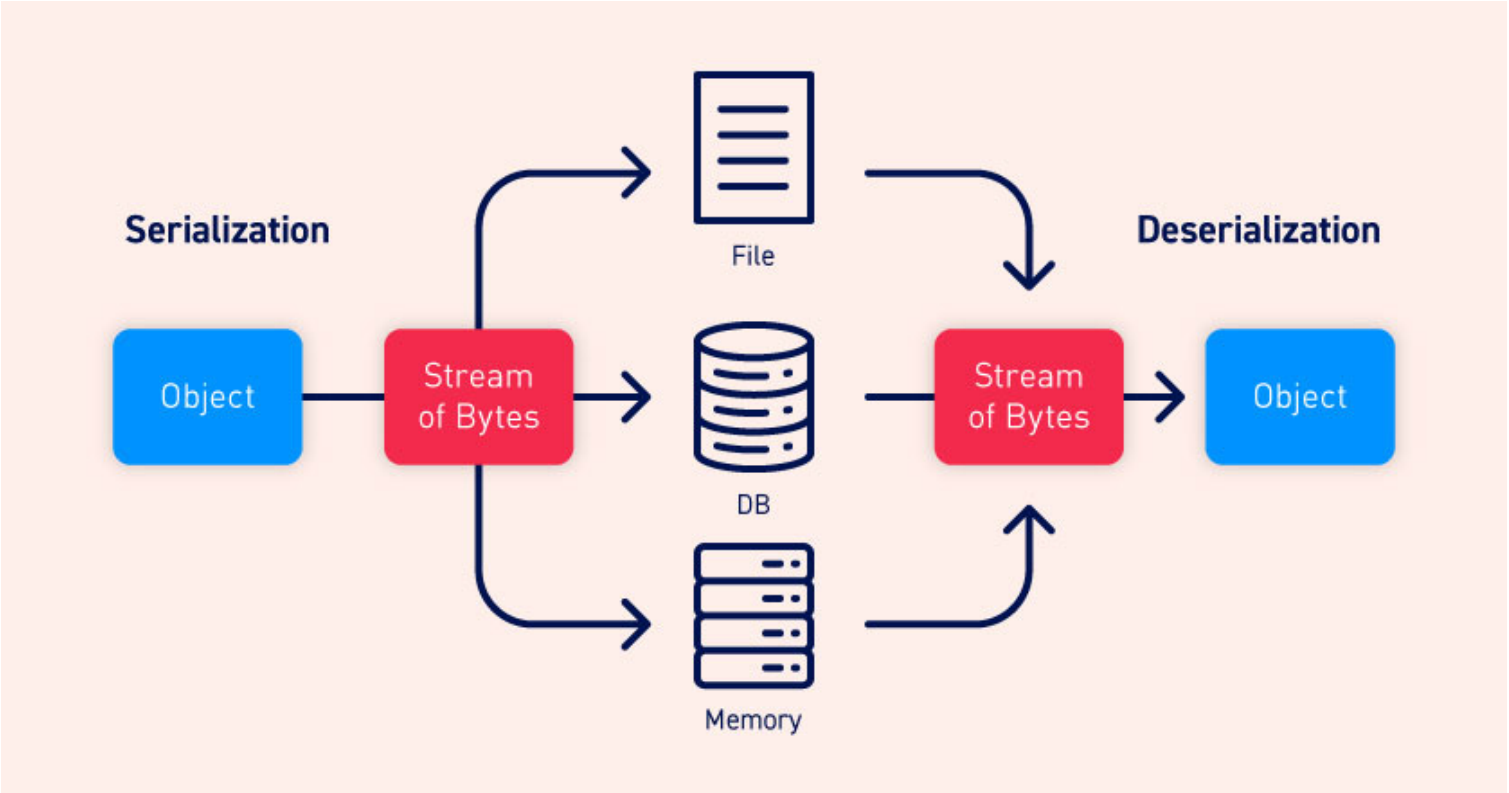
[10] https://zeromq.org/

Data Representation

- Heterogeneity
  - Different operation systems
  - Different programming languages
  - Different hardware architectures
- Problem when data structures must be sent in message in distributed system
- Different hosts may use different data representations
  - Sizes of integers, floating points, characters (ASCII vs Unicode)
  - Big vs. Little endian
  - Data structure layout in memory - padding of arrays
  - Pointers and structured data
    - Pointer representation might differ
    - Trees, lists, etc.

Serialization/deserialization

- **Serialization** is the process of translating a data structure or object state into a format that can be stored or transmitted and reconstructed later

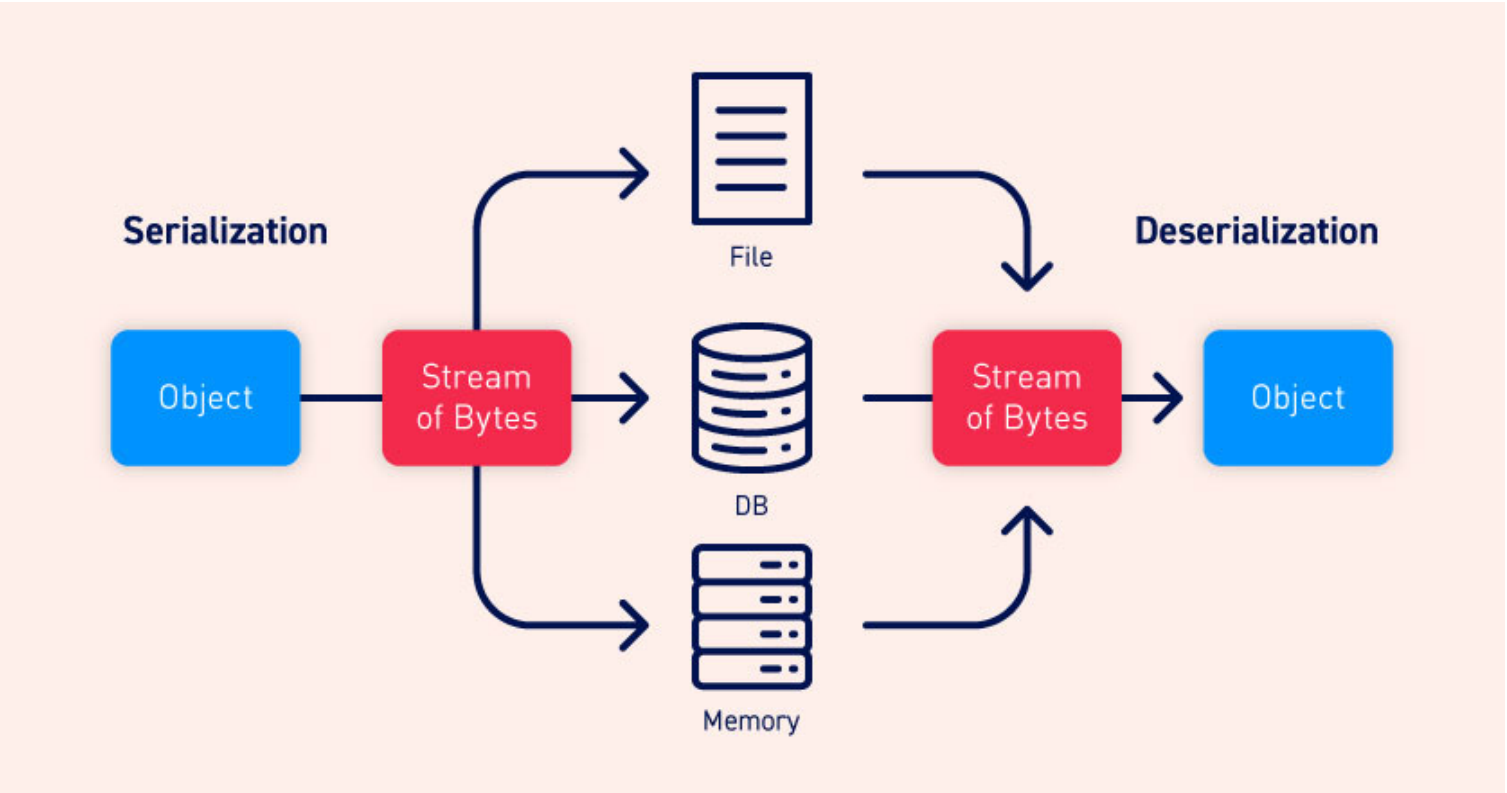


11

[11] https://portswigger.net/

Serialization/deserialization

- **Deserialization** is the opposite operation to serialization. It's extracting a data structure from a series of bytes.



11

[11] https://portswigger.net/

Serialization/deserialization - data schema

Explicit typing

- self-describing data (tags)
- additional information added to message

Implicit typing

- the devices at both ends have knowledge on how to code/decode the message
- depends on the defined protocol specification

Protocol Buffers (protobuf)

- Protocol Buffers are language-neutral, platform-neutral extensible mechanisms for serializing structured data [12]
- Open-source
- Designed to be smaller and faster than XML
- Data structure schemas (messages) described in a *proto* file
- Messages compilers for multiple languages (for example *protoc* for C/C++)

[12] https://protobuf.dev/

Protocol Buffers - example message

```
1  syntax = "proto3";
2
3  message Person {
4    string name = 1;
5    int32 id = 2;
6    message PhoneNumber {
7      optional string number = 1;
8    }
9    repeated PhoneNumber phones = 3;
10 }
11
12 message AddressBook {
13   repeated Person people = 1;
14 }
```

Protocol Buffers - example

- Generate Python code for the message

```
protoc -I=. --python_out=. ./person.proto
```

- Import and use message in Python

```
import person.person_pb2 as person_pb2

person_msg = person_pb2.Person()
person_msg.name = "John"
person_msg.id = 5

# Write message to binary file
with open("person.bin", "wb") as f:
    data = person_msg.SerializeToString()
    f.write(data)

# Read message from binary file
with open("person.bin", "rb") as f:
    msg = person_pb2.Person().FromString(f.read())
```

Summary

- OSI model
- Embedded interfaces
- MQTT, ZeroMQ
- Data Distribution Service (DDS)
- Inter-Process Communication (IPC)
- Data representation, serialization (protobuf)

The end

Thank you for your attention!