MQTT for IOT

Internet of Things

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS5055 – 2025I
PROF.: CWILLIAMS@UTEC.EDU.PE

Executive Summary

- **Motivation:** In the Internet of Things, devices must communicate efficiently, reliably, and often with limited resources.

  MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed specifically for **low-bandwidth, high-latency, or unreliable networks**.

It enables IoT devices to:

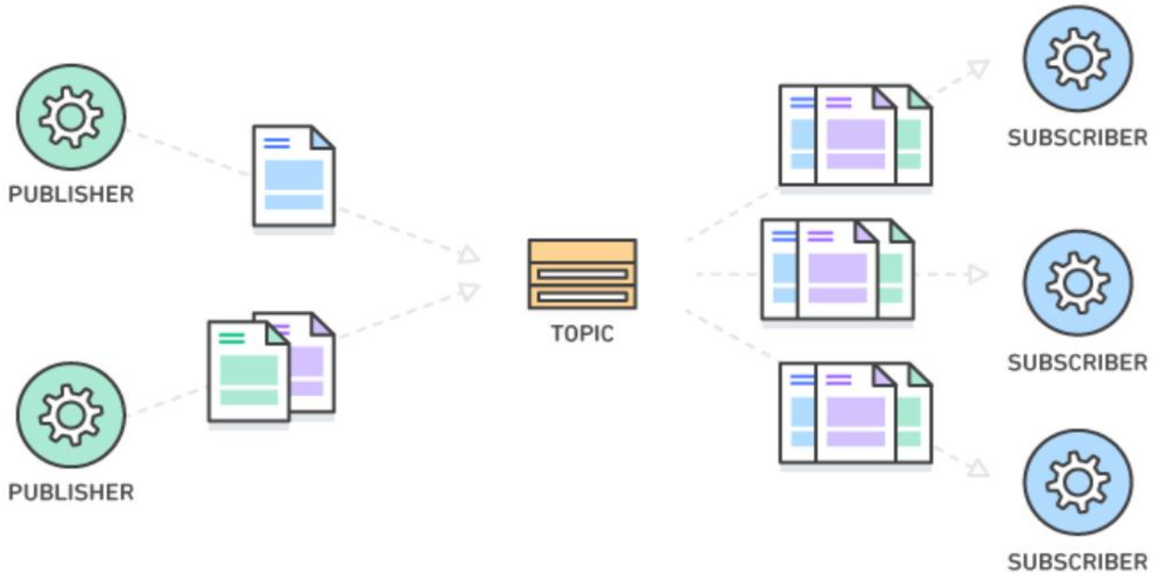
- **Send and receive data in real time**
- **Conserve power and bandwidth**
- **Scale easily to thousands of nodes**

Outline

- Introduction
- MQTT Header
- MQTT QoS
- MQTT implementation
- Conclusion

Recall: Publish/subscribe model

- **Publish-Subscribe**
  - It’s a message where editors do not program messages to specific clients, called subscribers, but they categorize published messages in clases without inputs from the subscribers.



Recall: Publish/subscribe model

- **On an IoT Project:**
  - **Broker:** Can be a function of the local Gateway ou be located in the cloud. It receives information from clients and redirects or processes.
  - **Publisher:** Connects to the broker to publish data
  - **Subscriber:** devices, applications, services that need data. They connect to the broker to be notified about new data.

Outline

Introduction

MQTT Message format

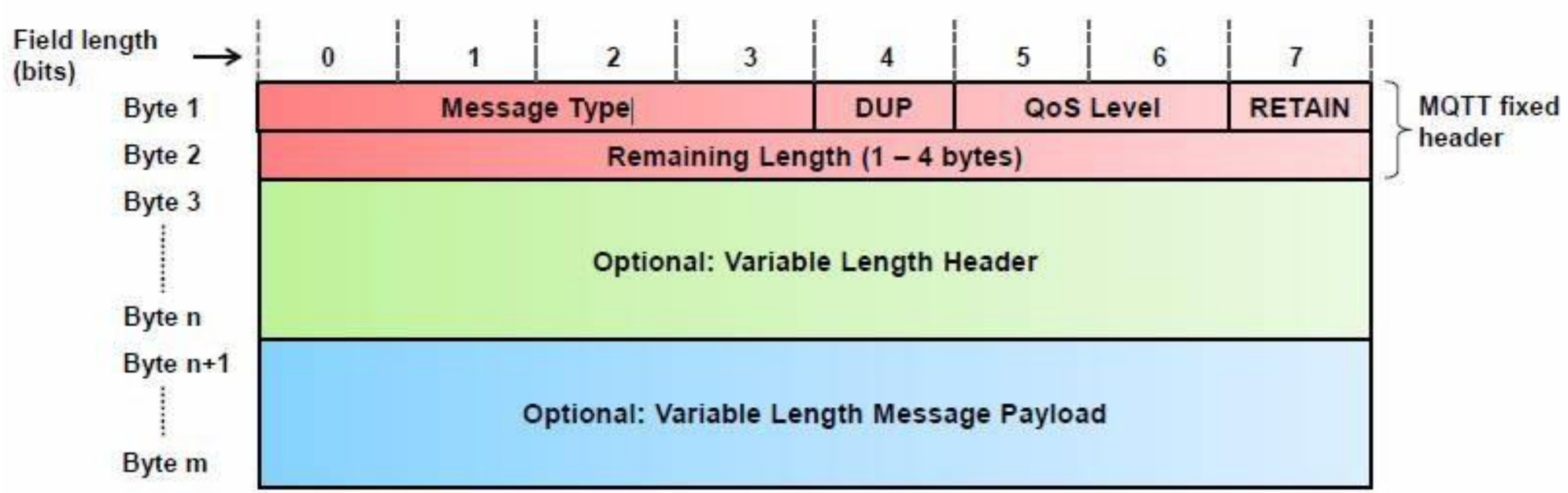
MQTT QoS

MQTT implementation

Conclusion

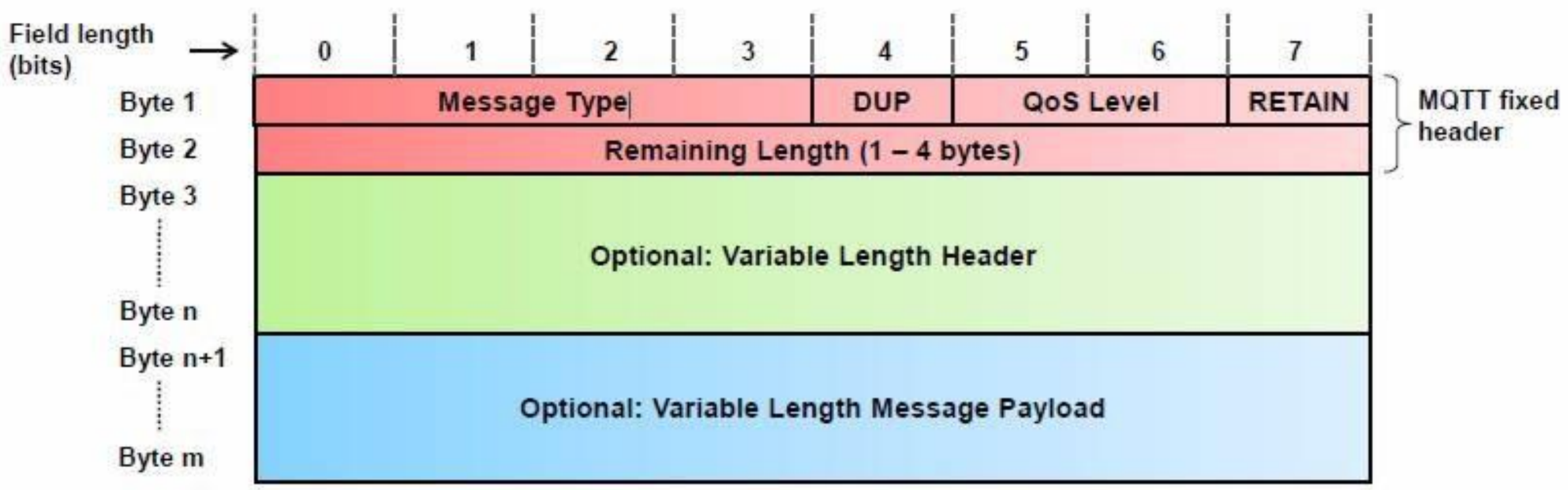
MQTT Message Format

- MQTT is data agnostic!
- MQTT uses TCP How to use TCP?
  - It is not necessary to know about TCP implementation, although it can be helpful
  - Understanding header-payload paradigm is enough
  - MQTT Header:



MQTT Message Format

- Message tipes:
  - CONNECT: Creates a connection with the broker and waits for the stablished connection
  - DISCONNECT: Waits for the client to finish an action and for the TCP/IP connection to be finalized – it stops listening
  - PUBLISH: Returns the data which was sent by the MQTT client



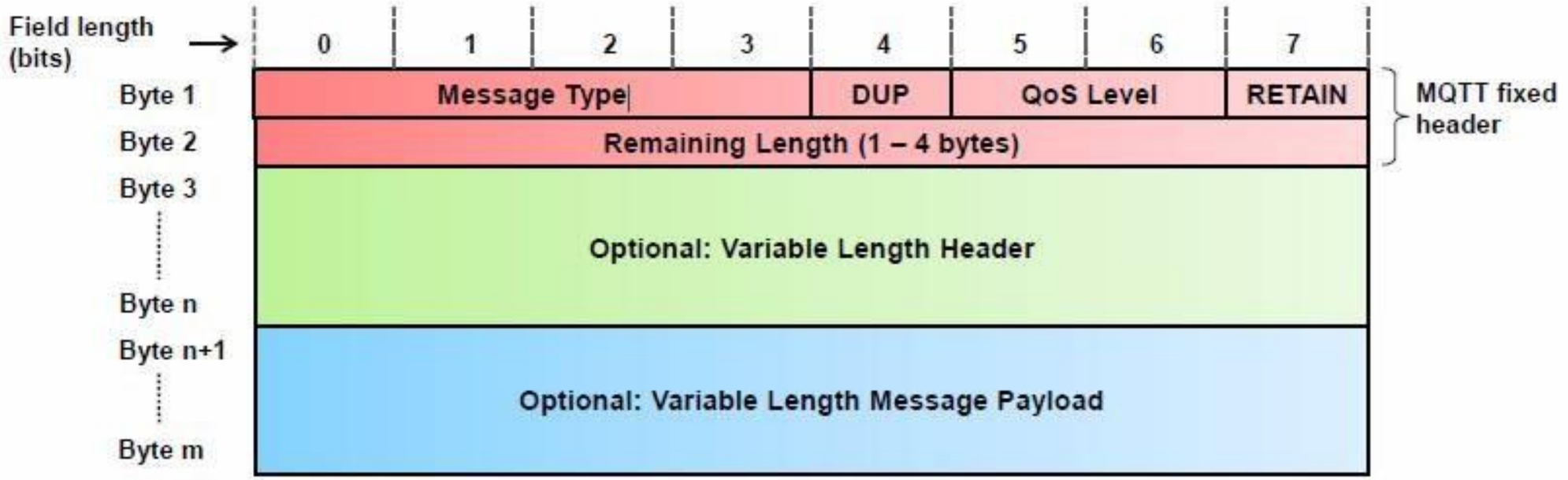
MQTT Message Format

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

MQTT Message Format

- Flags:

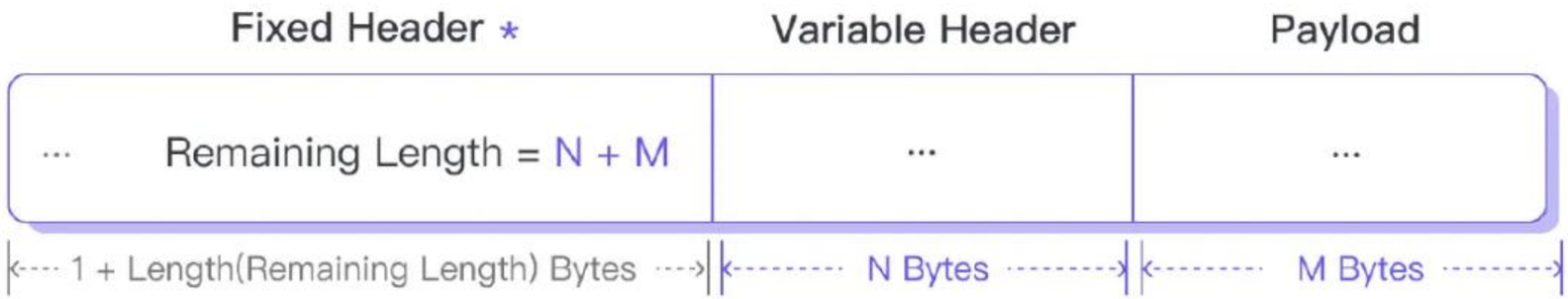
- Bit 4：DUP(Duplicate Delivery Flag), indicates whether the current PUBLISH packet is a retransmitted packet.
- Bit 5,6：QoS, indicates the quality of service level used by the current PUBLISH packet.
- Bit 0：Retain, indicates whether the current PUBLISH packet is retained.



MQTT Message Format

- Remaining Length:

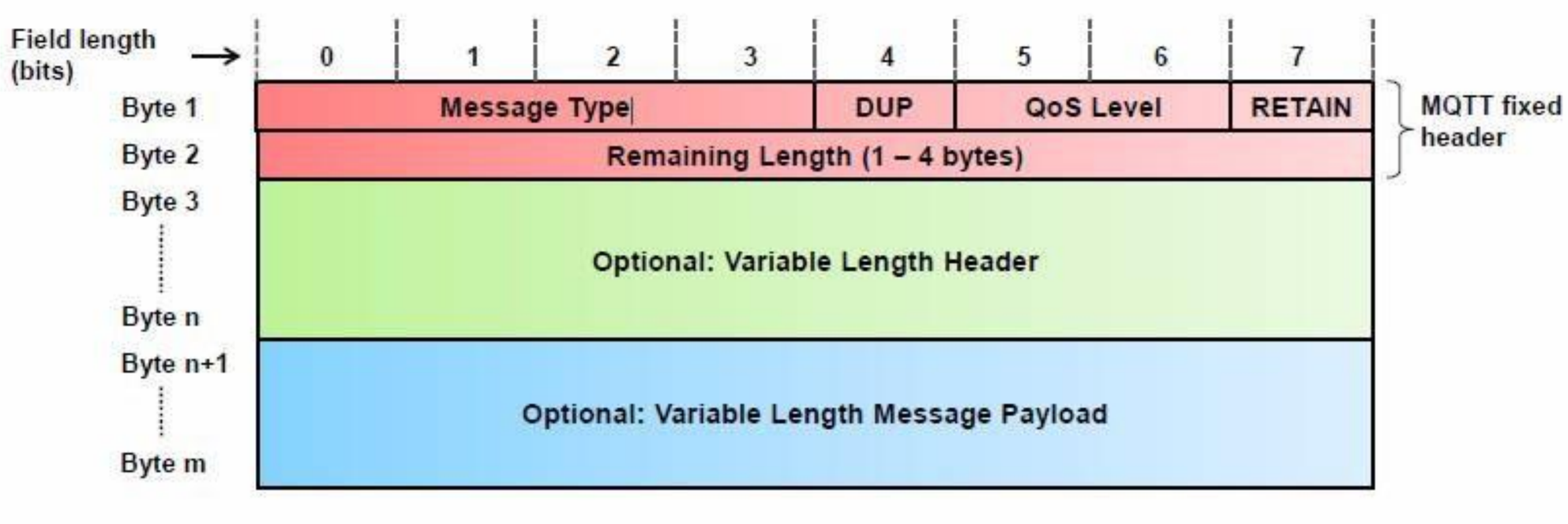
- The Remaining Length field specifies the number of bytes in the rest of the control packet, covering both the Variable Header and the Payload. As a result, the total length of an MQTT control packet is the sum of the Fixed Header size and the Remaining Length.



MQTT Message Format

- Variable Header :

- The Variable Header is an optional part of an MQTT control packet that contains additional information required for processing, depending on the type of packet.



MQTT Message Format

- Variable Header :

- The Variable Header is an optional part of an MQTT control packet that contains additional information required for processing, depending on the type of packet.

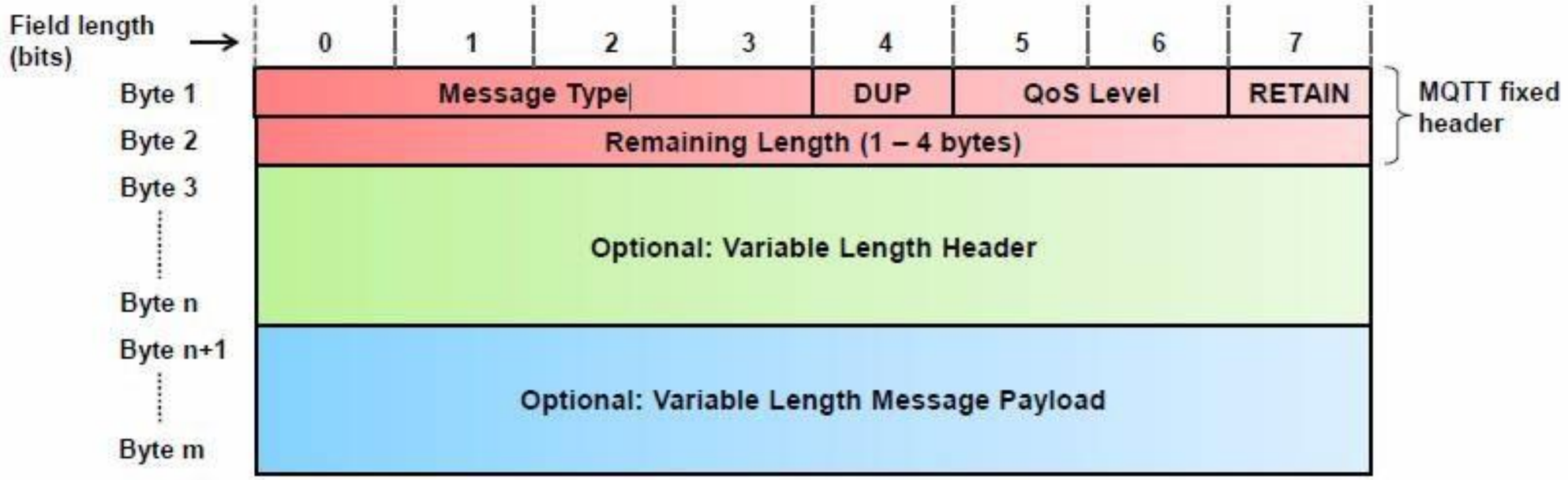
MQTT Message Format

- Payload:

Which serves as the core content of the packet. While the Variable Header provides supporting information, the Payload carries out the main function of the packet. For example :

In a PUBLISH packet, the Payload contains the actual application message, which is its primary purpose.

In a SUBSCRIBE packet, the Payload lists the topics to subscribe to along with their subscription options, fulfilling the main role of the packet..



Outline

- Introduction
- MQTT Message format
- MQTT QoS
- MQTT implementation
- Conclusion

MQTT – Quality of Service

- Quality of Service (QoS) for interchange of messages
  - There are 3 QoS and all connections to broker can specify which one is to be used.
    - <1 QoS 0
    - >1 QoS 1
    - =1 QoS 2

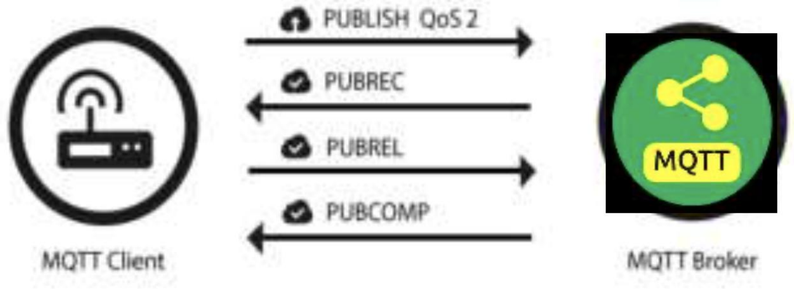
MQTT

- QoS 0: Fire/forgot –r “at most once”
  - The message is sent one time and then forgotten
  - The message is not stored and it does not have feedback to know if it arrived to its destination.
  - It´s the fastest transfer mode
  - If the transmission fails or if the client disconnects, the message is lost completely.

- QoS 1: “at least once”
  - Message delivered at least 1 time with feedback waiting (PUBBACK – acknowledgement)
  - If PUBBACK is not received, it sends the message continuously until feedback PUBBACK changes.

MQTT

- QoS 2: “exactly once”
  - Message is delivered exactly one time.
  - Needs local storage on the emitter and receiver until it is processed.
  - To guarantee this scenario we need a four-part handshake, which are two sets of request-response communications.
    - Message is sent (PUBLISH)
    - Response to receive (PUBREL) and confirmation that the process was concluded and data can be deleted (PUBCOMP)
    - After receiving PUBREL, the receiver can delete the message and when the emitter receives PUBCOMP it can delete de message as well.



Outline

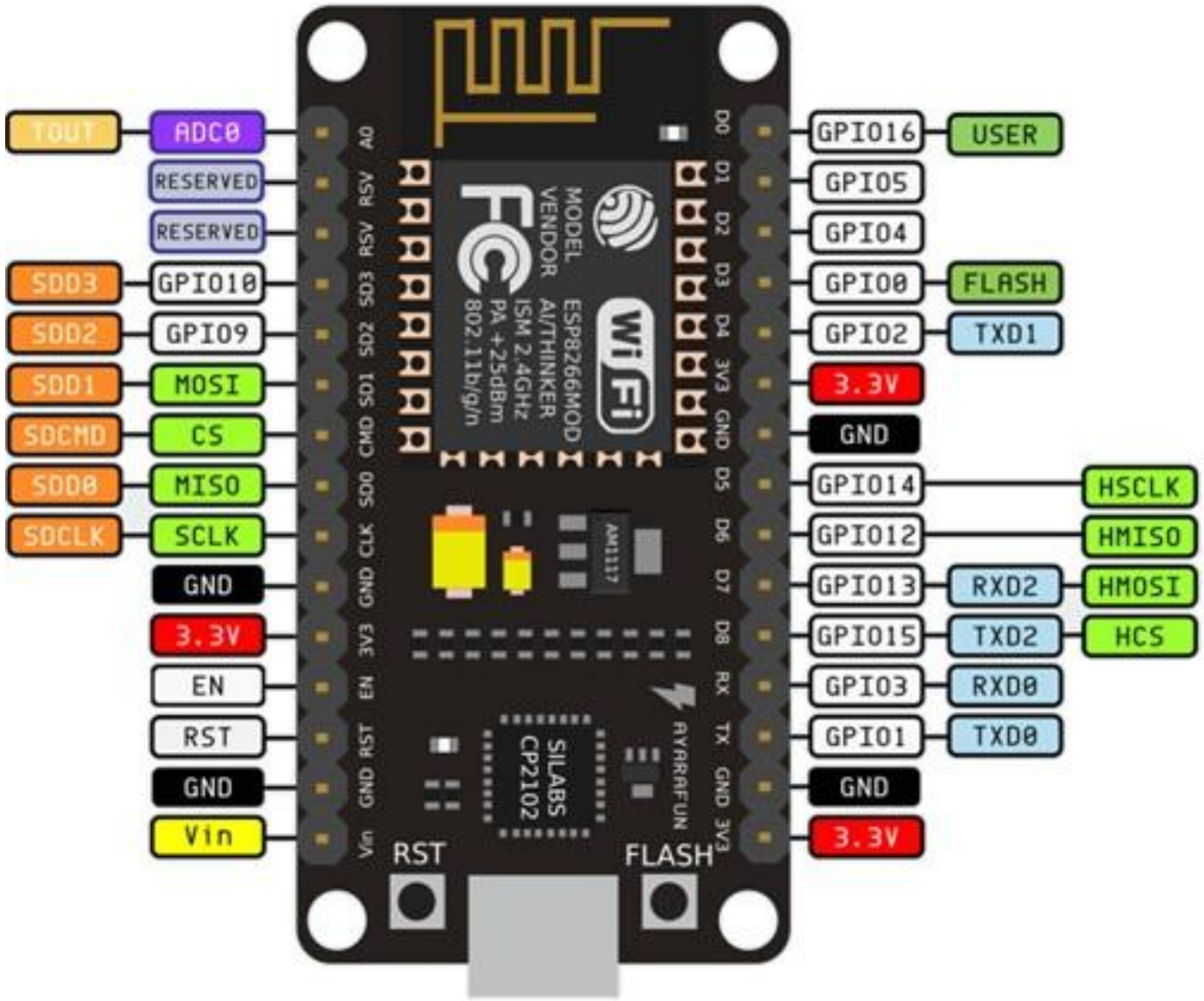
- Introduction
- MQTT Message format
- MQTT QoS
- MQTT implementation
- Conclusion

MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol.

Mosquitto is a specific implementation of the MQTT protocol. It is an open-source MQTT broker developed by the Eclipse Foundation. Mosquitto acts as an intermediary between devices that publish messages and those that subscribe to receive them.

MQTT Mosquitto + ESP32



MQTT Mosquitto + ESP32 (Tipos)

ARDUINO NODEMCU IOT ESP32 WIFI & BLUETOOTH DEVELOPMENT BOARD

38 PINS

NODEMCU ESP32 WIFI + Bluetooth 30 pins (CH340) Type C

30 PINS

ESP32 S3 WROOM-1-N16R8 ESP32-S3-DEVKITC-1

44 PINS

TTGO T-DISPLAY ESP32 1.14 DISPLAY MODULE

24 PINS

WIFI UNO BASED ESP32

32 PINS

NODEMCU ESP32 WIFI + Bluetooth 30 pins (CP2102) Micro

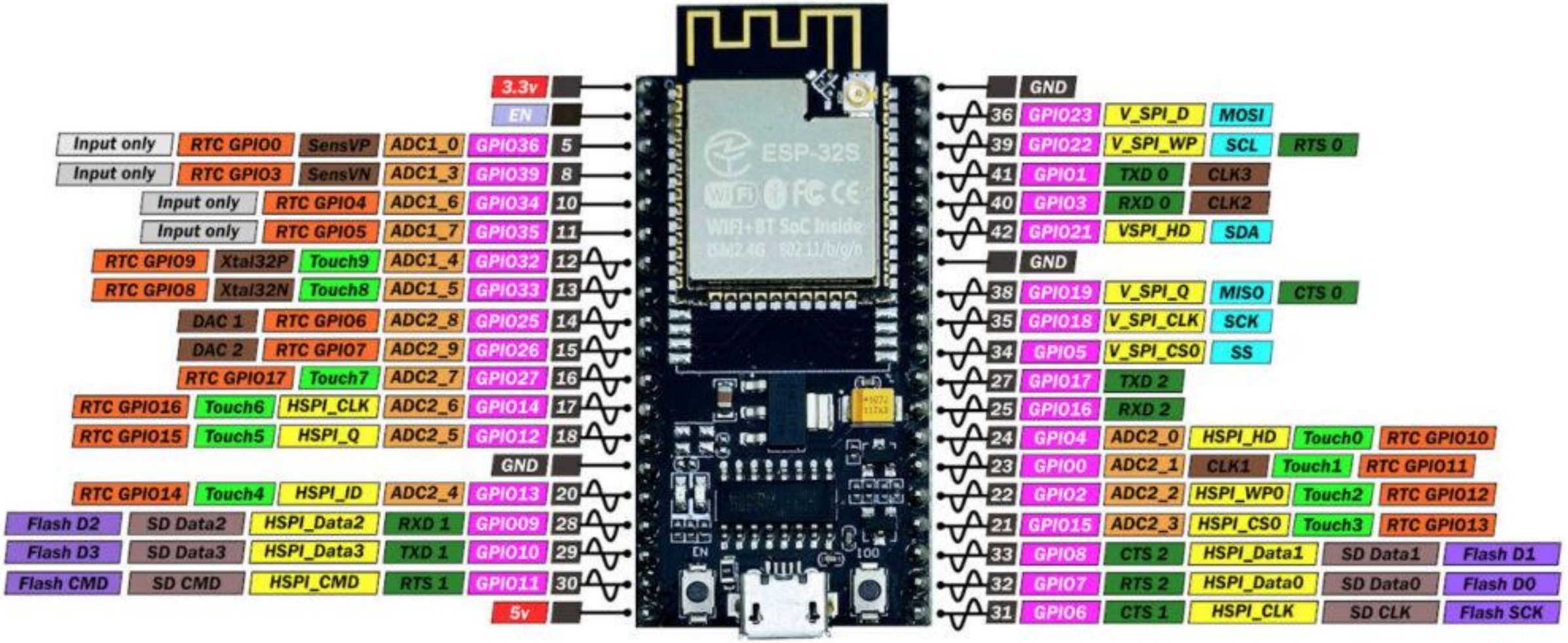
30 PINS

MQTT Mosquitto + ESP32 (Types)

They all share the same chip or variants of the ESP32, but they differ in important aspects such as the number of pins, the type of USB converter, ports, additional features, and size.

Placa	Pines	Características destacadas
Arduino NodeMCU IoT ESP32	38	Modelo genérico, buena compatibilidad, útil para proyectos IoT
NodeMCU ESP32 (CH340, Type C)	30	Usa el chip CH340 para USB, con conector USB-C moderno
ESP32-S3 WROOM (S3-DEVKITC-1)	44	Versión avanzada S3 con más RAM/flash, USB nativo, + GPIO
TTGO T-Display ESP32 (1.14")	24	Incluye pantalla LCD integrada, ideal para interfaces visuales
WiFi UNO Based ESP32	32	Compatible con factor de forma Arduino UNO
NodeMCU ESP32 (CP2102, Micro USB)	30	Similar al de CH340, pero con otro chip USB (CP2102)

ESP32 NODEMCU-32S ESP-32S Kit PINOUT



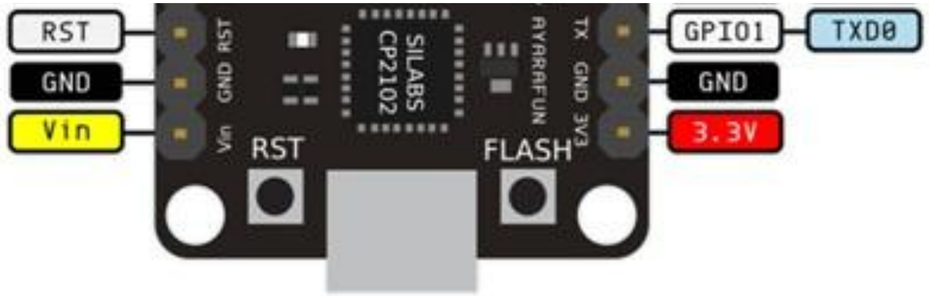
MQTT Mosquitto + ESP32

CAUTION!

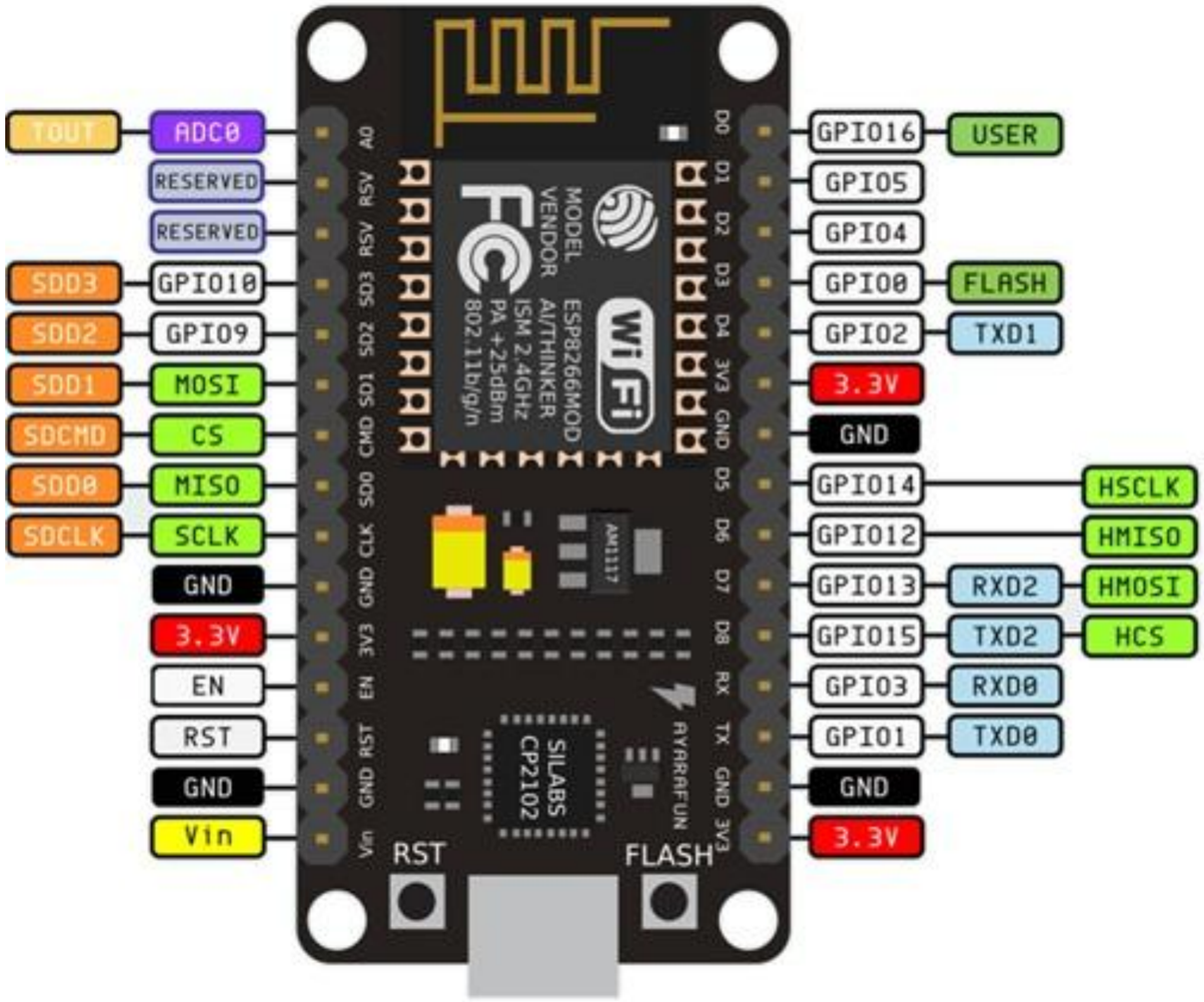
Voltage and current restrictions

The ESP8266 is a 3.3V microcontroller, so its I/O operates at 3.3V as well. The pins are **not 5V tolerant, applying more than 3.6V on any pin will kill the chip.**

The maximum current that can be drawn from a single GPIO pin is **12mA**.



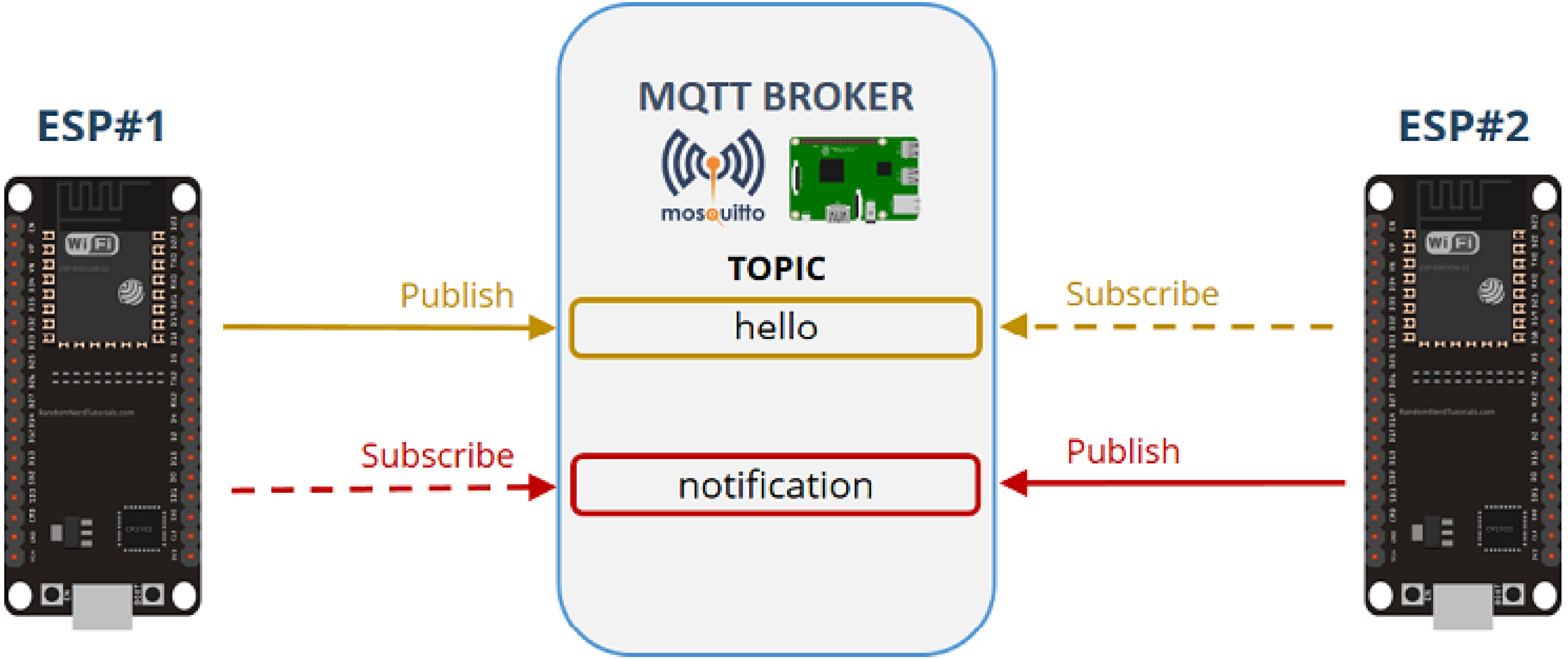
MQTT Mosquitto + ESP32



CAUTION!

Any ESP32 microcontroller (ESP8266, CAM, NodeMCU) operates at 3.3V, which means that the I/O pins also work at 3.3V. The pins do not tolerate 5V voltages, so applying a voltage higher than 3.6V will permanently damage the MCU chip. Additionally, the maximum current that any GPIO can provide is 12 mA.

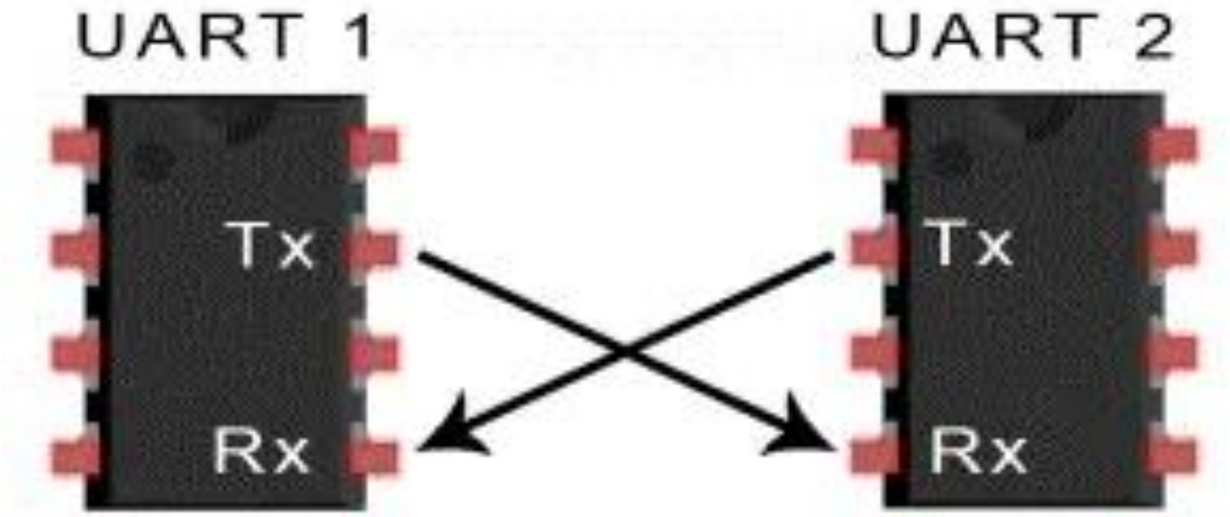
MQTT Mosquitto + ESP32

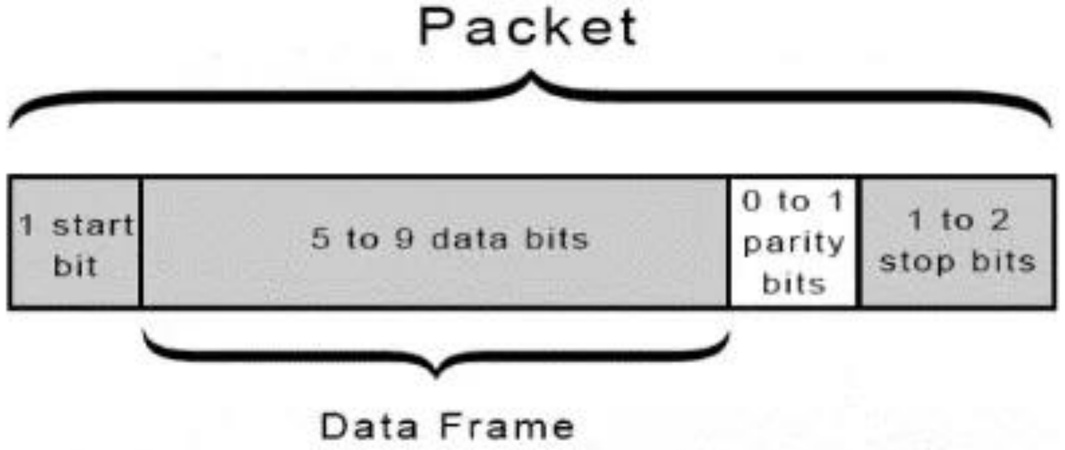


UART (Serial) Communication

UART stands for Universal Asynchronous Receiver/Transmitter. It is a hardware device (or circuit) used for serial communication between two devices.Connecting two UART devices is simple and direct.

The figure below shows a basic UART connection diagram.

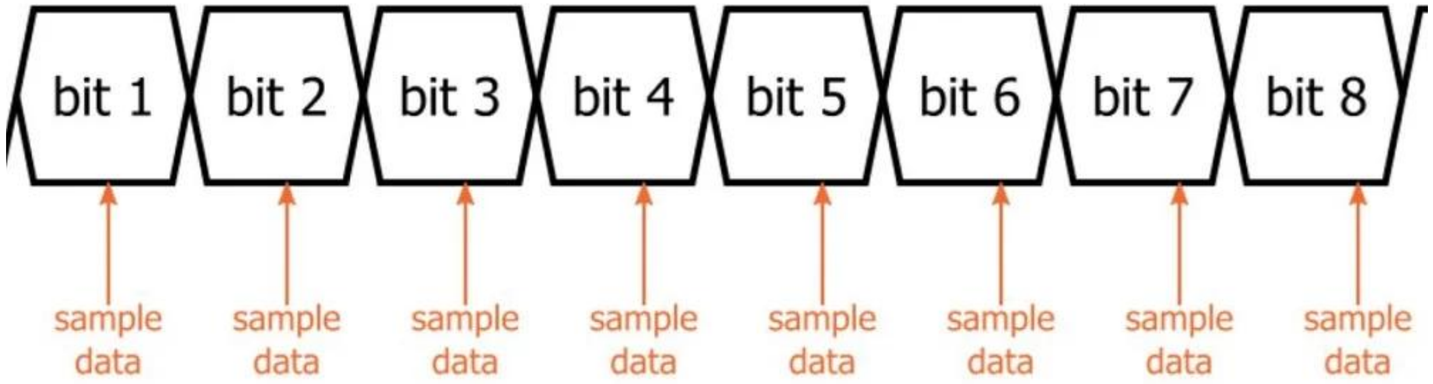




UART (Serial) Communication

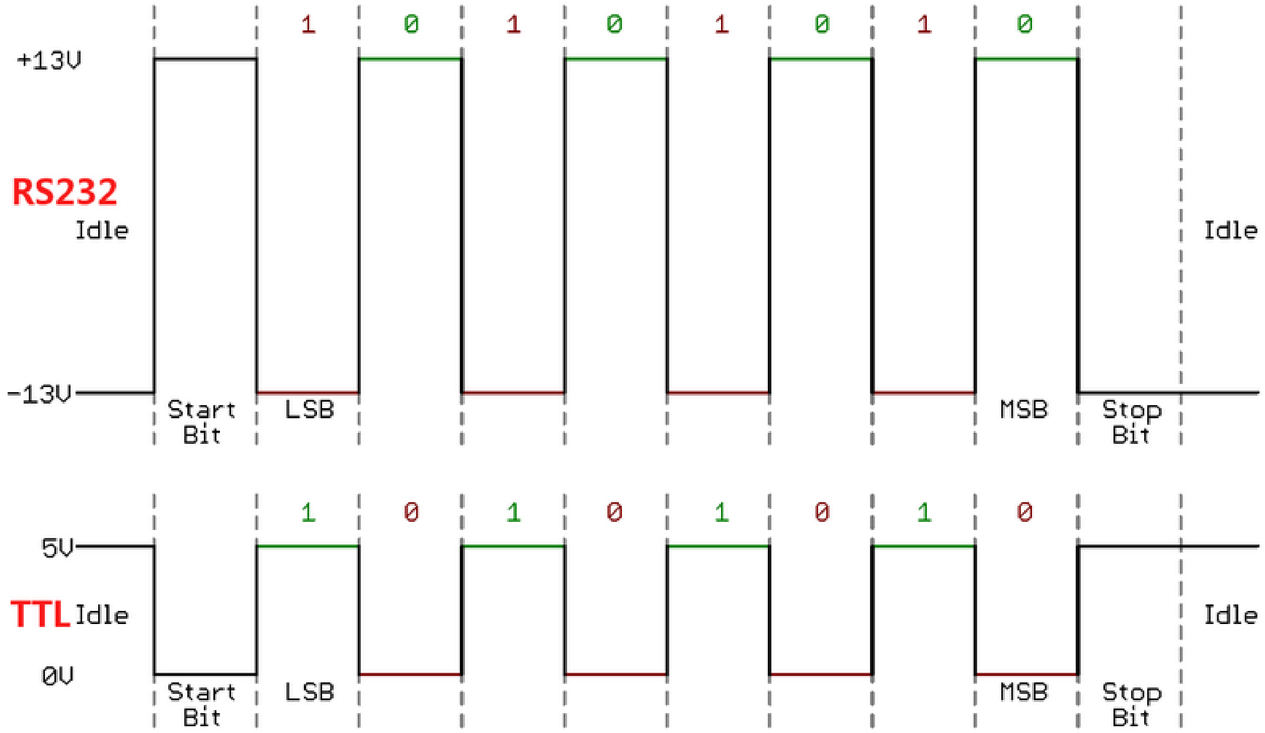
The baud rate is the number of bits per second (bps) that a UART device can transmit/receive. Both UART devices need to be configured with the same baud rate to ensure proper data transmission. Common baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 bps.

IMPORTANT: Both Serial ports used must be set to the same baud rate (e.g., 9600 or 115200).



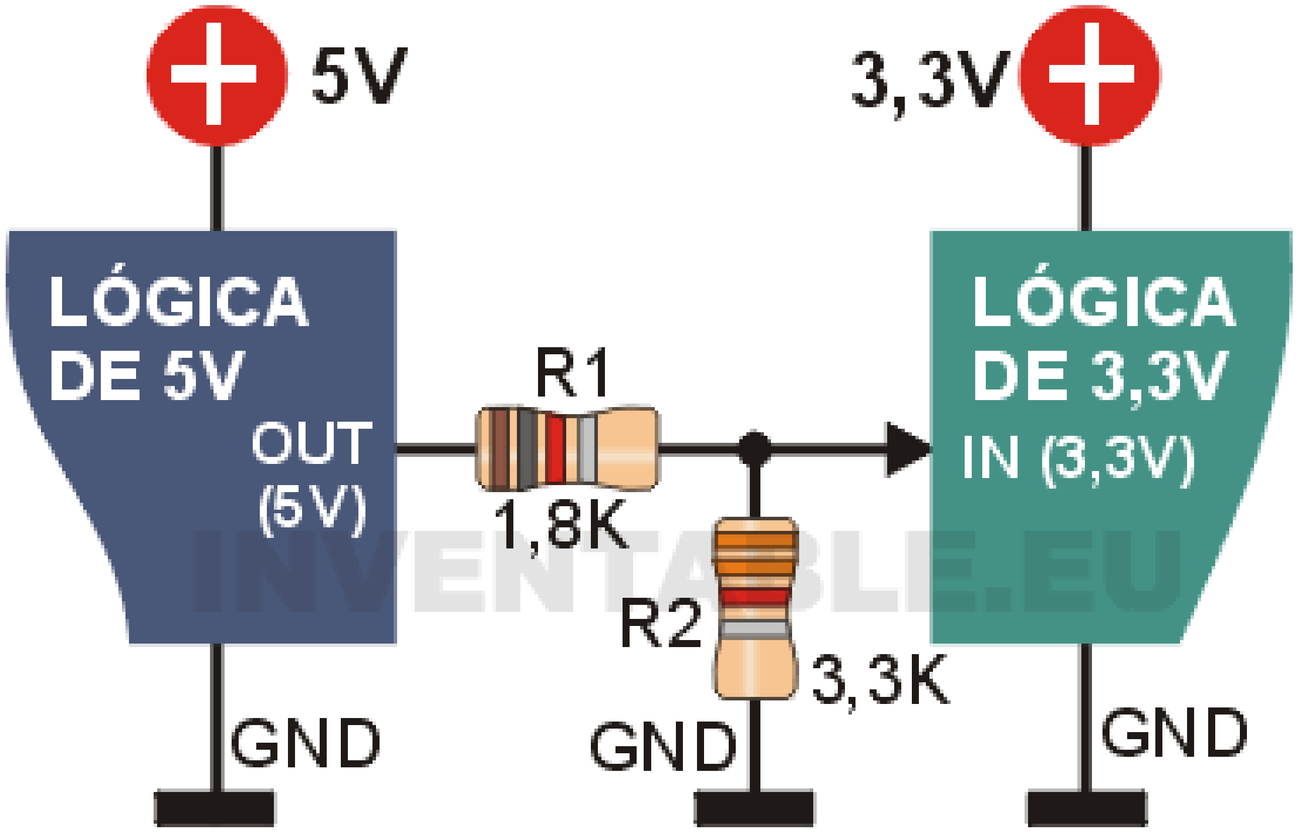
UART (Serial) Communication

The logic levels of UART can differ between manufacturers. For example, an Arduino Uno uses a logic level of 5V, while the RS232 port of a computer uses +/-12V logic levels. Connecting an Arduino Uno directly to an RS232 port will damage the Arduino. If both UART devices do not have the same logic levels, a proper level shifter circuit is needed to connect the devices safely.

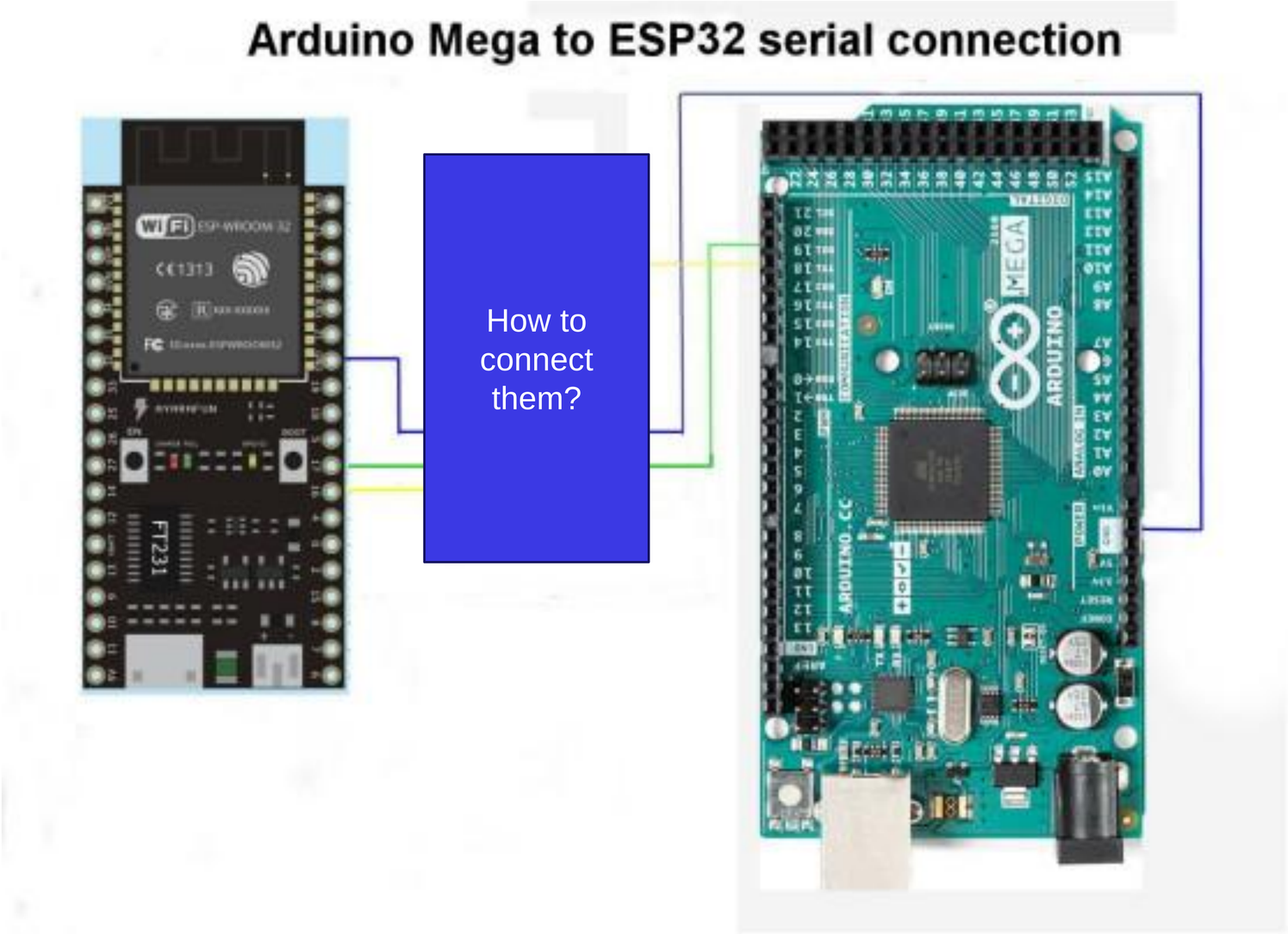


Communication between ESP32 and Arduino

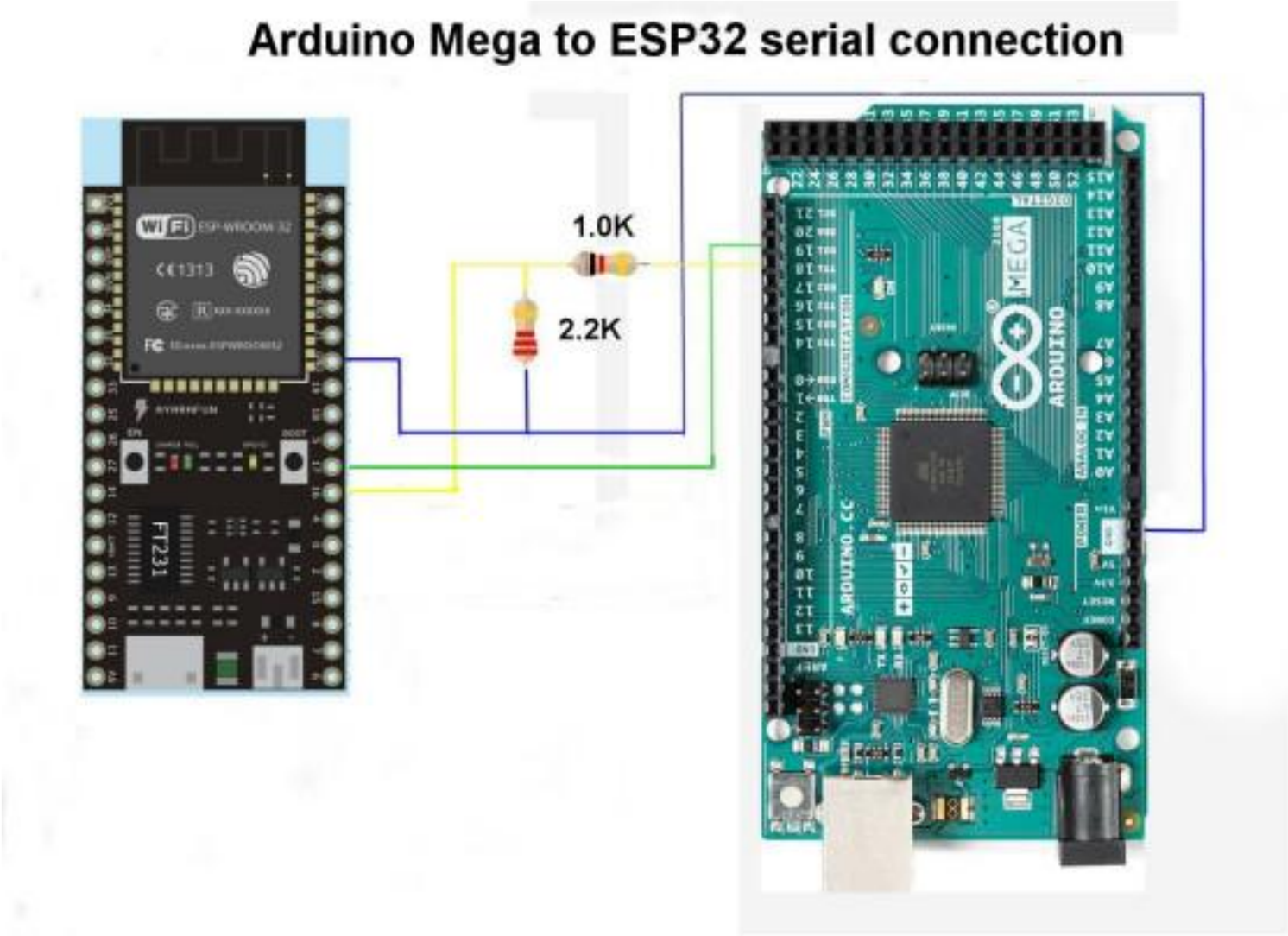
The Arduino (5V) sends 5V signals through TX, but the ESP32 only tolerates 3.3V on its RX. If connected directly, it could be damaged. Therefore, the signal should be reduced using a resistor divider or a level shifter. On the other hand, the ESP32 can send data to the Arduino without issue, as the Arduino interprets 3.3V as HIGH.



Communication between ESP32 and Arduino



Communication between ESP32 and Arduino



MQTT

- Software available
  - Brokers (https://github.com/mqtt/mqtt.github.io/wiki/servers):
    http://mosquitto.org/
    http://www.hivemq.com/
    https://www.rabbitmq.com/mqtt.html
    http://activemq.apache.org/mqtt.html
    https://github.com/mcollina/mosca
- Clients
  http://www.eclipse.org/paho/
- Source: https://github.com/eclipse/paho.mqtt.python
  http://www.hivemq.com/demos/websocket-client/
- Tools
  https://github.com/mqtt/mqtt.github.io/wiki/tools

Outline

- Introduction
- Communication protocols: HTTP and websocket
- CoAP and AMQP
- MQTT
- Conclusion

Conclusion

- There are many ways to communicate our IoT sensors/actuators
- Protocols are categorized in two groups
- Computational power of the “things” is an important feature for choosing communication protocols
- MQTT is important because:
  - Low memory consumption
  - Low processing requirements
  - Low bandwidth consumption