IoT Communications & Protocols

SINGAPORE INSTITUTE OF TECHNOLOGY

SingaporeTech.edu.sg

Objective

- What is Internet of Things
- Various IoT Communication Technologies
- Various IoT Communication Protocols

Definition of IoT

"An IoT system is a network of networks where, typically, a massive number of objects, things, sensors or devices are **connected** through communications and information infrastructure to provide value-added services via **intelligent** data processing and management for different **applications** (e.g. smart cities, smart health, smart grid, smart home, smart transportation, and smart shopping)."

(IEEE Internet of Things Journal)

Internet-of-Things (IoT): Components

Physical object (“thing”)

+

Controller (“brain”)

+

Sensors

+

Actuators

+

Networks (Internet)

Internet-of-Things (IoT): Another Perspective

COLLECT	CONNECT	COMPREHEND

Data vs. Knowledge

"The ultimate goal is transforming the raw data to insights and actionable knowledge and/or creating effective representation forms for machines and human users and creating automation."

This usually requires data from multiple sources, (near) real-time analytics and visualisation and/or semantic representations.

Key Considerations for Edge Computing Solutions

Performance:

- Latency and Response Time: Minimize latency for real-time applications.
- Data Processing and Analysis: Ensure sufficient local processing power.

Scalability:

- Device Scalability: Handle a growing number of connected devices.
- Geographic Scalability: Support expansion across different locations.

Security & Privacy:

- Data Security: Protect sensitive data at rest and in transit.
- Privacy Compliance: Adhere to data privacy regulations (e.g., PDPA).

Connectivity:

- Network Connectivity: Reliable and high-bandwidth network access.
- Bandwidth Management: Optimize bandwidth usage for efficient data transfer.

Hardware:

- Hardware Constraints: Select appropriate hardware for edge environments.
- Energy Efficiency: Minimize power consumption for sustainable operation.

Data Management:

- Data Storage: Local storage capacity and data synchronisation strategies.
- Data Management: Efficient data organisation and retrieval.

Reliability:

- Robustness: Ensure system resilience and fault tolerance.
- Reliability: Guarantee consistent and dependable performance.

User Experience:

- User Interface: Simple and intuitive interface for easy interaction.
- User Experience: Provide a seamless and positive user experience.

Compliance & Cost:

- Regulatory Compliance: Meet industry-specific regulations.
- Cost-Effectiveness: Optimize solution costs for maximum ROI.

Customisation:

- Customization: Adapt the solution to specific business needs.
- Flexibility: Enable future modifications and upgrades.

Characteristics of IoT Communications

- **Wireless**
- Low cost
- Low power (long battery duration)
- Low processing capacity
- Low storage capacity
- Extensive network of connections (scalability)
- Various bitrate/bandwidth requirements
- Various range
- Various QoS requirements
- Small size devices (typically)
- Simple network architecture and protocols (arguably)

Wireless Communications

Technology	Frequency	Data Rate	Range	Power Usage	Cost
2G/3G	Cellular Bands	10 Mbps	Several Miles	High	High
Bluetooth/BLE	2.4Ghz	1, 2, 3 Mbps	~300 feet	Low	Low
802.15.4	subGhz, 2.4GHz	40, 250 kbps	> 100 square miles	Low	Low
LoRa	subGhz	< 50 kbps	1-3 miles	Low	Medium
LTE Cat 0/1	Cellular Bands	1-10 Mbps	Several Miles	Medium	High
NB-IoT	Cellular Bands	0.1-1 Mbps	Several Miles	Medium	High
SigFox	subGhz	< 1 kbps	Several Miles	Low	Medium
Weightless	subGhz	0.1-24 Mbps	Several Miles	Low	Low
Wi-Fi	subGhz, 2.4Ghz, 5Ghz	0.1-54 Mbps	< 300 feet	Medium	Low
WirelessHART	2.4Ghz	250 kbps	~300 feet	Medium	Medium
ZigBee	2.4Ghz	250 kbps	~300 feet	Low	Medium
Z-Wave	subGhz	40 kbps	~100 feet	Low	Medium

Source: https://blog.helium.com/802-15-4-wireless-for-internet-of-things-developers-1948fc313b2e

IoT Stack vs Web Stack

	IoT Stack		Web Stack
TCP/IP Model	IoT Applications	Device Management	Web Applications
Data Format	Binary, JSON, CBOR		HTML, XML, JSON
Application Layer	CoAP, MQTT, XMPP, AMQP		HTTP, DHCP, DNS, TLS/SSL
Transport Layer	UDP, DTLS		TCP, UDP
Internet Layer	IPv6/IP Routing 6LoWPAN		IPv6, IPv4, IPSec
Network/Link Layer	IEEE 802.15.4 MAC IEEE 802.15.4 PHY / Physical Radio		Ethernet (IEEE 802.3), DSL, ISDN, Wireless LAN (IEEE 802.11), Wi-Fi

Source: https://www.linkedin.com/pulse/emerging-open-standard-protocol-stack-iot-aniruddha-chakrabarti/

IoT Communication Technologies

- Bluetooth Low Energy
- ZigBee
- HaLow
- LoRa
- LoRaWAN

IoT Communication Technologies

Image @ https://industrytoday.com/best-uses-of-wireless-iot-communication-technology/

Bluetooth

- Started with Ericsson's Bluetooth Project in 1994 for radio communication between cell phones over short distances
- Named after Danish king Herald Blatand (AD 940-981)[1]
- Intel, IBM, Nokia, Toshiba, and Ericsson formed Bluetooth SIG in May 1998
- **Version 1.0A** of the specification came out in late **1999**
- **IEEE 802.15.1** approved in early 2002 is based on Bluetooth.
- Later versions handled by Bluetooth SIG directly
- Key Features at **time of launch** in 1999-2000:
  - Lower Power: 10 mA in standby, 50 mA while transmitting
  - Cheap: $5 per device
  - Small: 9 mm2 single chips

1. https://www.bluetooth.com/about-us/bluetooth-origin/#:~:text=Surprisingly%2C%20the%20name%20dates%20back,earned%20him%20the%20nickname%20Bluetooth.

History of Bluetooth

ERICSSON intel NOKIA

4.0 Bluetooth®

Bluetooth 5
Go Faster. Go Further.

Bluetooth 5.2

Bluetooth Core Specification v5.1

1994-97 | 2006 | 2010 | 2011-2012 | 2015 | 2016 | 2019 | 2020

TraceTogether

2020

Bluetooth Versions

Bluetooth Versions	Remarks
Bluetooth 1.1	IEEE 802.15.1-2002 (1 Mbps)
Bluetooth 1.2	IEEE 802.15.1-2005. Completed Nov 2003. Extended SCO, Higher variable rate retransmission for SCO + Adaptive frequency hopping (avoid frequencies with interference)
Bluetooth 2.0	+ Enhanced Data Rate (EDR) (Nov 2004): 3 Mbps using DPSK. For video applications. Reduced power due to reduced duty cycle
Bluetooth 2.1	+ EDR (July 2007): Secure Simple Pairing to speed up pairing
Bluetooth 3.0	+ High Speed (HS) (April 2009): 24 Mbps using Wi-Fi PHY + Bluetooth PHY for lower rates
Bluetooth 4.0	(June 2010): Low energy. Smaller devices requiring longer battery life (several years). New incompatible PHY Bluetooth Smart or BLE
Bluetooth 4.1	4.0 + Core Specification Amendments (CSA) 1, 2, 3, 4
Bluetooth 4.2	(Dec 2014): Larger packets, security/privacy, IPv6 profile
Bluetooth 5	(2016): 4x wireless range, 2x speed, and broadcasting to two devices simultaneously.

Topology

Piconet v4.0

Up to 7 active slave per master..
Up to 255 'parked' slaves per master

Scatter net v4.1

Multiple piconets joined to form scatter net

Bluetooth Mesh v5.0

Bluetooth Mesh: https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-1/

Bluetooth Smart Protocol Stack

GATT defines how services and characteristics are organised and exchanged between devices.

ATT: It defines how data is represented in a BLE server database and the methods by which that data can be read or written.

GAP: responsible for device discovery, determines the network topology of a BLE system

SMP: Pairing, Authentication and Encryption

L2CAP: Multiplexing of different application protocols, Segmentation and reassembly of data packets, Controls peak bandwidth, latency, and delay variation

HCI enables interoperability between hosts and controllers produced by different manufacturers

Link Layer defines packet structure and control

PHY Layer takes care of transmission and receiving, modulation and demodulation, etc. 40 channels of 2MHz

Bluetooth 5: Powering Next-Gen Connectivity

- **Release Dates**: June/December 2016
- **Core Enhancement**: Enhanced Bluetooth Low Energy
- **Key Features**:
  - **Speed**: 2x Data Rate (2 Mbps)
  - **Range**: 4x Longer Range (800 ft) / 1.6 km Long-Range Mode
  - **Broadcast**: 8x Capacity (255B)
  - **Audio**: Dual-Audio & Enhanced Quality
  - **Power**: +20 dBm (100 mW) LE Transmit
  - **Compatibility**: Backward Compatible, Full Benefits with Bluetooth 5.0

BLE Mesh: Special Interest Group (SIG) Specification

- **Enhanced Physical Layer (PHY):**
  - 2 Mbps PHY: Faster data transfer for demanding applications.
  - Coded PHY: Increased robustness and range for challenging environments.
- **Increased Bandwidth & Directionality:**
  - 2 MHz Bandwidth: Efficient spectrum use, improved performance.
  - Directional Communication: Low-power beamforming for targeted connections.
- **Scalable & Reliable Mesh Networking:**
  - Enhanced Mesh Protocol: Support for larger networks, improved reliability.
- **IPv6 Integration:**
  - IPv6 Support: Seamless integration with IP-based IoT ecosystems.
  - Benefits: Enhanced security, advanced IoT applications.

Zigbee Overview

- Industrial monitoring and control applications requiring small amounts of data, turned off most of the time (<1% duty cycle), e.g., wireless light switches, meter reading, patient monitoring
- First standard was published in 2004
- Ultra-low power, low-data rate, multi-year battery life
- Power management to ensure low power consumption.
- Zigbee Protocol is less Complex. 32kB protocol stack vs 250kB for Bluetooth
- Range: 1 to 100 m, up to 65000 nodes.
- Tri-Band:
  - 16 Channels at 250 kbps in 2.4GHz ISM
  - 10 Channels at 40 kb/s in 915 MHz ISM band (Americas)
  - One Channel at 20 kb/s in European 868 MHz band
  - 920 MHz in Japan

Ref: http://www.cse.wustl.edu/~jain/cse574-20/

Zigbee Overview

- **IEEE 802.15.4** MAC and PHY (Except for Zigbee Smart Energy 2.0)
  - Higher layer and interoperability by Zigbee Alliance
- Up to 254 devices or 65,536 simpler nodes
- Named after zigzag dance of the honeybees
  - direction of the dance indicates the location of food
- Multi-hop ad-hoc **mesh network**
  - Multi-Hop Routing: message to non-adjacent nodes
  - Ad-hoc Topology: No fixed topology. Nodes discover each other
  - Mesh Routing: End-nodes help route messages for others
  - Mesh Topology: Loops possible

Ref: http://www.cse.wustl.edu/~jain/cse574-20/

ZigBee Product Interoperability

- Ensures Network interoperability but does not imply application layer interoperability
- There are multiple Compliant Platforms to choose from
- Products with the same application profiles interoperate end to end
- ZigBee has published a set of Public Application Profiles ensuring end product interoperability

ZigBee Device Types

- ZigBee Coordinator (ZC)
  - One required for each ZB network.
  - Initiates network formation.
- ZigBee Router (ZR)
  - Participates in multi-hop routing of messages.
- ZigBee End Device (ZED)
  - Does not allow association or routing.
  - Enables very low-cost solutions

ZigBee Network Topologies

ZigBee Stack Architecture

Smart Energy & Home Automation

Urgent demand for Smart Energy + compatibility with mainstream Home Automation systems enables customer choice

Z-Wave Vs. Zigbee: How are they different?

- Z-wave has a tightly controlled product ecosystem that caters to the smart home and smart building space, whereas Zigbee can be used for several applications
- Zigbee uses the global standard 2.4GHz ISM frequency band, whereas Z-Wave uses the 915 MHz ISM band (in the U.S.) and the 868 MHz RFID band (in Europe).
  - 2.4 GHz band can be subject to intense interference from Wi-Fi and Bluetooth systems, whereas the sub-GHz bands in Z-Wave do not face the same interference issues
- Lots of providers make Zigbee radios, but Z-Wave uses a proprietary radio system from Sigma designs
- Z-Wave uses frequency-shift keyed modulation (FSK), whereas Zigbee modulation is carried out through direct sequence spread spectrum (DSSS)

Extra Reading: https://www.electronicdesign.com/technologies/communications/article/21796052/whats-the-difference-between-zigbee-and-zwave

Wi-Fi: a New Contender of IoT

- Some low-power protocols do not currently enjoy ubiquitous access to the Internet

Wide deployments

Compatibility with Internet

IoT Wi-Fi (home) Vision

Wi-Fi 802.11ah

- aka "Wi-Fi HaLow" by WiFi Alliance.
- IEEE spec for Low-rate long-range IoT applications.
- Spectrum: Sub-Giga Hertz license-exempt spectrum. Not including TV white spaces (700 MHz for 802.11af).
  - 902-928 MHz (USA)
  - 863-868.6 MHz (Europe)
  - 916.5-927.5 MHz (Japan)
  - 755-587 MHz (China)
  - 917.5-923.5 MHz (Korea)

- Sub-GHz frequency ⇒ Longer range than 2.4 GHz, Less congested, better penetration
- Low bit rate for IoT, Short data transmissions, Power savings, Efficient MAC
- Goal: Support at least 4X devices per AP than legacy 802.11
- 802.11ah PHY down clocked by 10X when compared to 802.11ac
  - 2/4/8/16 MHz channels in place of 20/40/80/160 MHz in ac

Wi-Fi 802.11ah

Wi-Fi CERTIFIED HaLow™ for IoT

Features

- Sub-1 GHz spectrum operation
- Narrow band OFDM channels
- Several device power saving modes
- Native IP support
- Latest Wi-Fi® security

Benefits

- Long range: approximately 1 km
- Penetration through walls and other obstacles
- Supports coin cell battery devices for months or years
- No need for proprietary hubs or gateways

Source: Wi-Fi Alliance®

Source: https://www.wi-fi.org/news-events/newsroom/wi-fi-certified-halow-delivers-long-range-low-power-wi-fi

802.11 Standards: Frequency Range

IoT- Connectivity Technologies

Multiple standards, different attributes:

IoT - the Connectivity Pyramid

LPWAN requirements

- **Low Power Wide Area** wireless connects low bandwidth, low power devices and provides long-range coverage

Includes cellular (NB-IoT, LTE-M/Cat-M1) *and* non-cellular (Sigfox, LoRa etc) technologies

LoRa Technology

- Long Range Wide Area Network.
- LoRa technology was originally developed by a French company, Cycleo (founded in 2009 as an IP and design solution provider), a patented spread spectrum wireless modulation technology that was acquired by SemTech in 2012 for $5 million
- Originally developed by Cycleo in France.

  Acquired by Semtech corporation, which is the founding member of the **LoRa Alliance**

  Now 160+ members.
- V1.0 spec dated January 2015. Released to public July 2015.
- Rapid Adoption: Products widely available

Transceiver

Arduino Radio Shield

Connectivity Kit for Arduino, Waspmote, Raspberry Pi

Ref: https://www.lora-alliance.org/What-Is-LoRa/Technology

Frequency

- Uses ISM license-exempt band:
  - 915 MHz MHz in US. Power limit. No duty cycle limit.
  - 868 MHz in Europe. 1% and 10% duty cycle limit
  - 433 MHz in Asia
- Singapore: Telecommunications Standards Advisory Committee (TSAC)
  - ≤ 500 mW Effective radiated power (erp) : No approvals required within this power limit and within this frequency range.

866 – 869 MHz	≤ 500 mW (e.r.p.)	Refer to limits given in Table1-a of this TS	EN 302 208	Radio Telemetry, Telecommand, RFID system
920 – 925 MHz	≤ 500 mW (e.r.p.)	Refer to limits given in Table1-a of this TS	FCC Part 15 §15.249 and ANSI C63.10-2013	Radio Telemetry, Telecommand, RFID system

https://www.imda.gov.sg/~/media/imda/files/regulation%20licensing%20and%20consultations/ict%20standards/telecommunication%20standards/radio-comms/imdatssrd.pdf?la=en

- Currently suitable for public deployment in an area
  - All gateways report to the same server
  - A device can talk to any gateway
  - All devices use the same frequency

Ref: http://www.link-labs.com/what-is-lora/
http://www.radio-electronics.com/info/wireless/lora/lorawan-network-architecture.php

LoRa Modulation

- Designed to achieve low power transmissions over long distances
- Uses a form of **Chirp Spread Spectrum (CSS)**
- Data is encoded using the frequency increase/decrease rate
  - Data rate and link condition determines the frequency bandwidth required
- Multiple parallel transmissions with different data rates on the same frequency
- Power level is determined adaptively based on data rate and link condition.
  - Fast communication is used to save battery.
- Range: 2-5km for urban, up to 15km in sub-urban and up to 45km in rural.

Ref: "LoRA Physical Layer and RF Interface," Radio-Electronics, http://www.radio-electronics.com/info/wireless/lora/rf-interface-physical-layer.php

CHIRP SPREAD SPECTRUM (CSS)

- LoRa is a proprietary spread spectrum modulation scheme based on **Chirp Spread Spectrum modulation (CSS).**
- Chirp Spread Spectrum is a spread spectrum technique that uses wideband linear frequency modulated chirp pulses to encode information.
- Spread spectrum techniques are methods by which a signal is deliberately spread in the frequency domain → **power is spread over the entire spectrum**
  - For example, a signal is transmitted in short bursts, "hopping" between frequencies in a pseudo-random sequence.
- A chirp, often called a sweep signal, is a tone in which the **frequency increases (up-chirp)** or **decreases (down-chirp)** with time.

What is LoRaWAN?

- LoRaWAN defines the MAC layer protocol and system architecture for the network while the LoRa physical layer enables the long-range communication link.

Key Features

- Bidirectional communication

  Allows firmware/software updates of end devices
- Low Rate: 0.3 kbps to 22 kbps in Europe, 0.9 kbps in US
- Star of Stars Topology: Gateways are transparent bridges. Server is the brain. Simple devices.
- Secure: EUI128 Device Key, EUI64 Network Key, EUI64 Application Key

LoRaWAN - MAC Function over LoRa PHY

- **Role**: Acts as the traffic controller on top of the LoRa radio system.
- **Centralized Control**: A central server manages the entire LoRaWAN network.
- Key Functions:
  - **Device Assignment**: The server gives each device a unique frequency, spreading code, and initial data rate
  - **Optimized Communication**: The server prevents message duplication, schedules acknowledgments, and adjusts data rates for efficiency
  - **Network Synchronization**: All gateways in the network are synchronized for seamless communication (star topology).
  - **Adaptive Data Rate (ADR)**: The server intelligently determines the best data rate for each device based on its distance and communication needs. This helps to conserve energy and improve network performance.
  - **Robust Security**: LoRaWAN incorporates multiple security layers, including unique keys for devices, networks, and applications (EUI128 Device Key, EUI64 Network Key, EUI64 Application Key), along with encryption and authentication mechanisms to ensure data confidentiality and integrity.

Three classes of End Devices

COMPARING LPWAN TECHNOLOGY OPTIONS

Feature	LoRaWAN	Narrow-Band	LTE Cat-1 2016 (Rel12)	LTE Cat-M 2018 (Rel13)	NB-LTE 2019(Rel13+)
Modulation	SS Chirp	UNB / GFSK/BPSK	OFDMA	OFDMA	OFDMA
Rx bandwidth	500 - 125 KHz	100 Hz	20 MHz	20 - 1.4 MHz	200 KHz
Data Rate	290bps - 50Kbps	100 bit/sec 12 / 8 bytes Max	10 Mbit/sec	200kbps – 1Mbps	~20K bit/sec
Max. # Msgs/day	Unlimited	UL: 140 msgs/day	Unlimited	Unlimited	Unlimited
Max Output Power	20 dBm	20 dBm	23 - 46 dBm	23/30 dBm	20 dBm
Link Budget	154 dB	151 dB	130 dB+	146 dB	150 dB
Batery lifetime - 2000mAh	105 months	90 months		18 months	
Power Efficiency	Very High	Very High	Low	Medium	Med high
Interference immunity	Very high	Low	Medium	Medium	Low
Coexistence	Yes	No	Yes	Yes	No
Security	Yes	No	Yes	Yes	Yes
Mobility / localization	Yes	Limited mobility, No loc	Mobility	Mobility	Limited Mobility No Loc

LoRa & LoRaWAN: A Summary

- **LoRaWAN** is a standardized MAC layer protocol built on top of the LoRa physical layer, which uses a variation of chirp spread spectrum modulation.
- **LoRa**'s Chirp Spread Spectrum (CSS) makes it less sensitive to noise and allows multiple devices to send data simultaneously on the same frequency.
- **LoRaWAN** networks use a centralized server for efficient management and control.
- **LoRa** devices broadcast messages to all gateways, and the gateway with the strongest signal typically replies.
- While **LoRaWAN** offers long range and low power consumption, it can have limitations in downlink traffic performance.

Application/Data Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer
Application 1
Application 2
...
Application N
HTTP
MQTT
...
CoAP
Zigbee
UDP
TCP
Thread
IPv6
RPL
IPv4
6LoWPAN
3GPP
LoraWAN
IEEE 802.15.4
BLE
HaLow 802.11ah
RFID/NFC
Wi-Fi 802.11 b/g/n/ac/ax
LoRa

IoT Communication Protocols

- HTTP
- REST
- CoAP
- MQTT

HTTP

Application/Data Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer
Application 1
Application 2
...
Application N
HTTP
MQTT
...
CoAP
Zigbee
UDP
TCP
Thread
IPv6
RPL
IPv4
6LoWPAN
3GPP
LoraWAN
IEEE 802.15.4
BLE
HaLow 802.11ah
RFID/NFC
Wi-Fi 802.11 b/g/n/ac/ax
LoRa

HyperText Transfer Protocol (HTTP)

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (usually)

Web Content

- Web servers return **content** to clients
  - a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type
- Example MIME types
  - `text/html` — HTML document
  - `text/plain` — Unformatted text
  - `application/postscript` — Postcript document
  - `image/gif` — Binary image encoded in GIF format
  - `image/jpeg` — Binary image encoded in JPEG format

Static & Dynamic Content

- The content returned in HTTP responses can be either static or dynamic
  - **Static content**: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
  - **Dynamic content**: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
- **Bottom line**: all web content is associated with a *file* that is managed by the server

Universal Resource Locator (URL)

- Each *file* managed by a server has a unique name called a *URL*
- URLs for static content:
  - `http://www.singaporetech.edu.sg:80/index.html`
  - `http://www.singaporetech.edu.sg/index.html`
  - `http://www.singaporetech.edu.sg`
    - Identifies a *file* called `index.html`, managed by a web server at `www.singaporetech.edu.sg` that is listening on port 80
- URLs for dynamic content:
  - `http://www.singaporetech.edu.sg:8080/cgi-bin/simi?15432&223`
    - Identifies an executable *file* called `simi`, managed by a web server at `www.singaporetech.edu.sg` that is listening on port 8080, that should be called with two argument strings: `15432` and `223`

HTTP Requests

- HTTP request is a *request line*, followed by zero or more *request headers*
- Request line: `<method> <uri> <version>`
  - `<version>` is HTTP version of request (`HTTP/1.0` or `HTTP/1.1`)
  - `<uri>` is typically URL for proxies and servers
  - `<method>` is either `GET, POST, OPTIONS, HEAD, PUT, DELETE,` or `TRACE`

GET /devices/123/temperature

A **URI** is an **identifier** of a specific resource. Like a page, or book, or a document. A **URL** is **special type of identifier** that also tells you how to access it, such as HTTPs , FTP , etc. .

HTTP Requests

request line

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

- HTTP methods:
  - `GET`: Retrieve static or dynamic content
    - Arguments for dynamic content are in URI
    - Workhorse method (99% of requests)
  - `POST`: Retrieve dynamic content
    - Arguments for dynamic content are in the request body
  - `OPTIONS`: Get server or file attributes
  - `HEAD`: Like `GET` but no data in response body
  - `PUT`: Write a file to the server
  - `DELETE`: Delete a file on the server
  - `TRACE`: Echo request in response body
    - Useful for debugging

HTTP Responses

- HTTP response is a *response line* followed by zero or more *response headers*
- Response line:
- `<version> <status code> <status msg>`
  - <version> is HTTP version of the response
  - <status code> is numeric status
  - <status msg> is corresponding English text
    - 200 OK - Request was handled without error
    - 403 ForbiddenServer lacks permission to access file
    - 404 Not found Server couldn't find the file
- Response headers: `<header name>: <header data>`
  - Provide additional information about response
  - `Content-Type:` MIME type of content in response body
  - `Content-Length:` Length of content in response body

Example:

```
{
 "temperature": 26.5,
 "unit": "Celsius"
}
```

HTTP Responses (example)

version

status code & status msg

header name

header data

response line

response headers

response body

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Example of an HTTP Transaction from an IoT Device via Telnet

Client (IoT Device) Action

```
iot-device> telnet api.iotserver.com 80 # IoT opens connection to server

Trying 192.168.1.100...                 # Server IP resolution

Connected to api.iotserver.com.

Escape character is '^]'.


POST /devices/123/temperature HTTP/1.1  # IoT device sends HTTP
                                        # POST request

Host: api.iotserver.com                 # Required HTTP/1.1
                                        # HOST header

Content-Type: application/json          # Specifies JSON data format

Content-Length: 45                      # Size of payload

                                        # Empty line terminates headers

{ "device_id": "123", "temperature": "26.5" }  # JSON payload with sensor data
```

Server (Cloud API) Response

```
HTTP/1.1 201 Created                    # Server response line
                                        # Success)

Date: Tue, 12 Mar 2024 10:15:30 GMT     # Timestamp of response

Server: IoT-Cloud/1.0                   # Server software

Content-Type: application/json          # Response format

Content-Length: 35                      # Response payload size

Connection: close                       # Server closes connection


{ "status": "success", "id": "98765" }  # Response body confirming
                                        # data saved
```

{REST:API}

REpresentational State Transfer (REST)

- First introduced by Roy Fielding in 2000.
- Web standards-based architecture and uses HTTP Protocol
- REST is not a protocol but an architectural style
- Every component is a resource that is accessed via HTTP protocols.
- REST can adopt several standards, e.g. HTTP, URL, XML, HTML, GIF, JPEG, etc

Software Architecture for REST

https://blog.ndepend.com/rest-vs-restful/

REST & HTTP

- The motivation for REST was to take advantage of the success of the Web
  - URI Addressable resources → URL
  - HTTP Protocol
  - Request → Response → Display Response

- Make use of the HTTP protocol beyond POST & GET
  - HTTP PUT, HTTP DELETE

A URI is an identifier.
A URL is an identifier that tells you how to get to it.

Six Architectural Constraints of REST

1. **Client-Server Architecture** - The client and server act independently.
2. **Statelessness** - The server does not record the state of the client.
3. **Cacheable** - The server marks whether data is cacheable.
4. **Layered System** - The application behaves the same regardless of any intermediaries between the client and server.
5. **Uniform Interface** - The client and server interact in a uniform and predictable way. An important aspect of this is that the server exposes resources.
6. **Code on Demand (optional)** – It allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. Allowing features to be downloaded after deployment improves system extensibility.

REST vs RESTful

https://www.educba.com/rest-vs-restful/

Resources

- The key abstraction of information in REST is a resource
- A resource is a conceptual mapping to a set of entities
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g., “today's weather in Berlin”), a collection of other resources, a non-virtual object (e.g., a person), etc.
- Represented with a global identifier (URI in HTTP)
  - http://www.boeing.com/aircraft/747

Naming Resources

- REST uses URI to identify resources

  - http://localhost/books/ → **Retrieves all books**
  - http://localhost/books/ISBN-0011 → **Retrieves a specific book**
  - http://localhost/books/ISBN-0011/authors → **Retrieves the authors of that specific book**

  - http://localhost/classes
  - http://localhost/classes/cs2650
  - http://localhost/classes/cs2650/students

- As you traverse the path from more generic to more specific, you are navigating the data

REST Verbs

Create
Read
Update
Delete

Verb	Objective	Usage	Multiple Request	Cache
GET	Retrieve items from resource	Links	Yes	Yes
POST	Create new item in resource	Forms	No	No
PUT	Replace existing item in resource	Forms	Yes	No
PATCH	Update existing item in resource	Forms	No/Yes	No
DELETE	Delete existing item in resource	Forms/Links	Yes	No

GET

- How clients ask for the information they seek

- Issuing a GET request transfers the data from the server to the client in some representation

- GET http://localhost/books
  - Retrieve all books

Response: Returns book data.

- GET http://localhost/books/ISBN-0011021
  - Retrieve book identified with ISBN-0011021

- GET http://localhost/books/ISBN-0011021/authors
  - Retrieve authors for book identified with ISBN-0011021

POST, PUT & PATCH

- HTTP POST creates a resource
- HTTP PUT updates the (entire) resource

- POST http://localhost/books/ **Response: 201 Created with the new book details.**
  - Content: {“isbn”: “111-456-789”, “title”: “The Boring Two”, “authors”: …}
  - Creates a new book with given properties

- PUT http://localhost/books/isbn-111 **Response: 200 OK with updated book details.**
  - Content: {“isbn”: “123-456-789”, “title”: “The Boring One”, “authors”: …}
  - Update book identified by isbn-111 with submitted properties

- PATCH http://localhost/books/isbn-111 **Response: 200 OK with updated book title.**
  - Content: {“title”: “The Boring One”}
  - Update book identified by isbn-111 with submitted properties

DELETE

- DELETE http://localhost/books/isbn-111
  - Remove specific book

Response: 204 No Content.

Representations

- How data is represented or returned to the client for presentation.
- Typical formats:
  - JavaScript Object Notation (**JSON**)
  - Extensible Markup Language (**XML**)

```
{ "id": 1, "name": "John Doe", "email": "john@example.com" }
```

```
<user>
  <id>1</id>
  <name>John Doe</name>
  <email>john@example.com</email>
</user>
```

- The above shown are <u>not</u> the <u>resource</u> itself.
- They are just a way to <u>represent</u> the <u>resource</u> which is stored somehow in your server.

Summary – HTTP & REST in IoT

- Definitions
  - HTTP is the foundation of web communication, using **request-response** interactions.
  - REST is an **architectural style** that leverages HTTP for scalable and structured APIs.
- IoT relevance
  - IoT devices use **RESTful APIs** to allow remote monitoring, device control, and data retrieval.
  - Well-suited for **high-bandwidth**, internet-connected IoT applications (e.g., smart homes, industrial dashboards).
- Challenges
  - **High overhead:** (Relatively large) HTTP headers make it inefficient for low-power devices.
  - **Request-response model:** Not ideal for real-time or event-driven IoT applications.
  - **Latency:** REST APIs require a new connection for each request, increasing delays.
  - **Not optimised for constrained networks**: LoRa and Zigbee don't directly support HTTP.
- Overcoming These Challenges
  - **MQTT** (Message Queuing Telemetry Transport)
    - A lightweight protocol designed for low-bandwidth, real-time communication via publish-subscribe messaging.
  - **CoAP** (Constrained Application Protocol)
    - A REST-based protocol optimised for low-power, constrained devices with a smaller message format.

CoAP

RFC 7252

Constrained Application Protocol (CoAP)

- Developed by the Constrained Resource Environments (CoRE) IETF group
- REST-based web transfer protocol (RESTful protocol)
- To communicate over the Internet using UDP instead of TCP (while still providing reliability)
- CoAP has been designed to work on resource-constrained microcontrollers with as low as 10 KB of RAM and 100 KB of code space (RFC 7228).
- CoAP could be used for adapting simple HTTP interfaces into a more compact protocol
  - different message types to request resources from server: GET, PUT, POST, DELETE and **OBSERVE**
  - subset of HTTP functionality re-designed for low power embedded devices such as sensors (for IoT and M2M)
- Supports multicast and congestion control

Example of a CoAP Transaction

https://devopedia.org/constrained-application-protocol

CoAP Architecture

HTTP and CoAP work together across constrained and traditional Internet environments

HTTP protocol stack and CoAP protocol stack

Bormann, Carsten, Angelo P. Castellani, and Zach Shelby. 2012. "CoAP: An application protocol for billions of tiny Internet nodes." IEEE Internet Computing, vol. 1, no. 2, pp. 62–67, Mar/Apr.

TCP vs UDP

- TCP overhead is too high, and its flow control is not appropriate for short-lived transactions
- UDP has lower overhead and supports multicast

Message Header (4 bytes)

Bit:	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
	Ver, T, TKL	Code	Message ID	
	Token (if any)			
	Options (if any)			
	Payload Marker	Payload (if any)		

Ver - Version (1)
T – Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)
TKL- Token Length, if any, the number of Token bytes after this header
Code - Request Method (1-10) or Response Code (40-255)
Message ID – 16-bit identifier for matching responses
Token – Optional response matching token

https://tools.ietf.org/html/rfc7252
https://tools.ietf.org/id/draft-ietf-core-coap-00.html

Message Mapping

```
CLIENT                                                          SERVER
   |                                                               |
   |        ----- CON [0x7d34] GET /temp ------------->            |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   0   |     GET = 1     |           MID=0x7d34          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  11   |   4   |        "temp" (4 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
CLIENT                                                          SERVER
   |                                                               |
   |      <-------- ACK [0x7d34] 2.05 Content ---------            |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 2 |   0   |     2.05=69     |           MID=0x7d34          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Response Code

CoAP Status Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
2.31	Continue
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.08	Request Entity Incomplete
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

HTTP Status Code	Description
1xx	Informational
2xx	Successful 200 – OK 201 – Created 202 – Accepted 204 – No Content
3xx	Redirection 301 - Moved Permanently 305 - Use Proxy 307 - Temporary Redirect
4xx	Client Error 400 – Bad Request 401 – Unauthorized 403 – Forbidden 404 - Not Found 405 – Method Not Found 408 – Request Timeout
5xx	500 – Internal Server Error 501 – Not Implemented 503 – Service Unavailable 504 - Gateway Timeout

Only mostly used HTTP Status Codes are listed here

https://www.slideshare.net/HamdamboyUrunov/the-constrained-application-protocol-coap-part-2

Message Layer Model

Four Message Types:

- **Confirmable (CON)** – requires an acknowledgement
- **Non-confirmable (NON)** – no acknowledgement needed
- **Acknowledgement (ACK)** – Acknowledge a Confirmable message
- **Reset (RST)** - indicates a Confirmable message has been received but context is missing for processing

Request / Response Layer Model

Piggy-backed

Dealing with Packet Loss

Request / Response Layer Model

Separate Response

Non-confirmable R&R

Observation

http://tesi.cab.unipd.it/44294/1/TESI.pdf

Resource Discovery

Block Transfer

Block2 Option added to messages

- nr = incremental block number within original data
- m = more blocks flag
- sz = block size

Other CoAP Stuff

- Intermediaries and Caching
- Security - DTLS (TLS/SSL for Datagrams)
- Alternative Transport (e.g. SMS)
- Message Header Options

MQTT

Message Queuing Telemetry Transport (MQTT)

- MQTT was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) back in 1999, where their use case was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connections. They specified the following goals, which the future protocol should have:
  - Simple to implement
  - Bi-directional asynchronous “push” Communications
  - Provide a QoS Data Delivery
  - Lightweight and Bandwidth Efficient
  - Data Agnostic
  - Continuous Session Awareness

https://www.youtube.com/watch?v=QSwR-JMmNOo&feature=emb_logo
https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/

MQTT

- Built for proprietary embedded systems; now shifting to IoT
- You can send anything as a message; up to 256 MB
- Built for unreliable networks (Continuous Session Awareness)
- Enterprise scale implementations down to hobby projects
- Decouples readers and writers
- Message have a topic, quality of service, and retain status associated with them
- MQTT is a binary based protocol where the control elements are **binary bytes** and not **text strings**.

TCP/IP Port: 1883

When running over TLS/SSL, TCP/IP Port 8883
SSL: Secure Socket Layer
TLS: Transport Layer Security*

MQTT

- MQTT consist of three parts:
  - Broker
  - Subscribers
  - Publishers

https://www.paessler.com/it-explained/mqtt

Publish/Subscribe Concept

Decoupled in space and time:

- **Clients** do <u>not</u> need each others IP address
- **Broker**'s IP and port <u>must</u> be known by clients
- **Namespace** hierarchy used for **topic** filtering
- Possible that a published message is never consumed by any subscriber

MQTT: Simple to implement

Connect

Subscribe

Publish

Unsubscribe

Disconnect

```
client = new Messaging.Client(hostname, port, clientId)
client.onMessageArrived = messageArrived;
client.onConnectionLost = connectionLost;
client.connect({ onSuccess: connectionSuccess });

function connectionSuccess() {
     client.subscribe("planets/earth");
     var msg = new Messaging.Message("Hello world!");
     msg.destinationName = "planets/earth";
     client.publish(msg);
}

function messageArrived(msg) {
     console.log(msg.payloadString);
     client.unsubscribe("planets/earth");
     client.disconnect();
}
```

Eclipse Paho JavaScript MQTT client

Topics

- Each published data specifies a topic
- Each subscriber subscribed to that topic will receive it
- Topic format:

MQTT Message Format

Shortest Message is Two Bytes

MQTT Fixed Header

Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	

Message Types

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
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
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client 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	Client is disconnecting
Reserved	15	Forbidden	Reserved

MQTT: pub/sub decouples sender from receiver

https://www.slideshare.net/BryanBoyd/mqtt-austin-iot-meetup

MQTT: allows wildcard subscription

single level wildcard: **+** multi-level wildcard: **#**

get all the bedroom data → **myhome/bedroom/+**
get the temperature data of three rooms → **myhome/+/temperature**
get all the data → **myhome/#** or **#**

https://www.slideshare.net/BryanBoyd/mqtt-austin-iot-meetup

Best Practices

- Do not use topics beginning with $
- Never use a leading forward slash: /myhome/groundfloor/livingroom
- Embed a unique identifier or the Client Id into the topic: client1/status
- Don’t forget extensibility: think about future topic names
- Use specific topics, not general ones:
  - myhome/livingroom/temperature
  - myhome/livingroom/brightness
  - myhome/livingroom/humidity
- Don’t subscribe to #
- Never use spaces in a topic
- Keep the topic short and concise
- Use only ASCII characters; avoid non-printable characters

https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

MQTT: reliable messaging

https://www.slideshare.net/BryanBoyd/mqtt-austin-iot-meetup

Publishing “QoS 0” – Unreliable

- 0 – unreliable (aka “**at most once**”)
  - OK for continuous streams, least overhead (1 message)
  - “Fire and forget”
  - TCP will still provide reliability

https://www.emqx.com/en/blog/introduction-to-mqtt-qos

Publishing “QoS 1” – At Least Once

- 1 – delivery “**at least once**” (duplicates possible)
  - Used for alarms – more overhead (2 messages)
  - Contains message ID (to match with **ACK**ed message)

Publishing “QoS 2” – Exactly Once

- 2 – delivery “**exactly once**”
  - Utmost reliability is important – most overhead (4 messages) and slowest

MQTT: Flexible Payload

MQTT: Retain Message (state retention)

MQTT: Last Will and Testament (LWT)

Paho-MQTT (Python-client Example)

```
#import the client
import paho.mqtt.client as mqtt

#create new instance
client = mqtt.Client("P1")

#connecting to broker (host, port, keepalive, bind_address)
client.connect("iot.eclipse.org“, 1883, 60, “192.168.1.184”)

#Subscribing to topic “house/bulb1” with QoS=1
client.subscribe(“house/bulb1”, 1)
client.subscribe([(“house/bulb3”,2),(“house/bulb4”,1),(“house/bulb5”,0)])

#Publishing message to topic “house/bulb1” with “OFF” payload and QoS of 1 with retain=FALSE
client.publish(“house/bulb1”, “OFF”, 1, FALSE)
```

subscribing to **multiple topics**

http://www.steves-internet-guide.com/into-mqtt-python-client/

Security

- **Transport Layer Security*** (TLS) and **Secure Sockets Layer** (SSL) are cryptographic protocols which use a handshake mechanism to negotiate various parameters to create a secure communication channel between a client and a server.
  - Mitigate “Man-In-The-Middle-Attacks”
  - Ensures that the communication content cannot be read or altered by third parties*
- **Payload Encryption** is the encryption of application-specific data on the application level
  - End-to-end (E2E) encryption

  - Client-to-broker

MQTT: Brokers

Appliance	Cloud	Open Source
IBM MessageSight 1m connections 15m QoS 0 / sec policies for security, messaging, connection developer VM	**HiveMQ** **IBM IoT Foundation** **Eurotech EDC** **Litmus Loop** Others	**Mosquitto** (C) **Mosca** (Node.js) **Moquette** (Java) **RSMB** (C) [tiny] Others **Eclipse Sandbox** iot.eclipse.org
Commercial	"Freemium"	Free

MQTT 5 – Whats New

- **User Properties**
- **Shared Subscriptions**
- **Payload Format**
- **Request-Response**
- **Topic Alias**
- **Enhanced Authentication**
- **Flow Control**
- Expiry Interval
  - Message
  - Session
- 128 Reason Codes (19 in MQTT 3.1.1) and Reason String
- Improved Client Feedback – List of supported features
- Prohibition of retransmitting MQTT messages for healthy TCP connections
- Passwords without usernames for authentication

User Properties - What

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/

User Properties - Why

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/

Shared Subscriptions

$share/GROUPID/TOPIC

The shared subscription consists of 3 parts:

- A static shared subscription identifier ($share)
- A group identifier
- The actual topic subscriptions (may include wildcards)

Example for a subscriber:

$share/my-shared-subscriber-group/myhome/groundfloor/+/temperature

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part7-shared-subscriptions/

Payload Format

- 0 indicates an “unspecified byte stream”
- 1 represents a “UTF-8 encoded payload.”
  - MIME-like content type descriptor
- When the Payload Format Indicator isn’t provided, it automatically defaults to 0.

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part8-payload-format-description/

Request-Response

Topic Alias

- Payload are relatively small
- Topics are relatively long
- Message throughput is relatively high

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part10-topic-alias/

Enhanced Authentication

- Challenge-response
  - vs credential-based
- Authentication methods:
  - Salted Challenge Response Authentication Mechanism (SCRAM)
  - Kerberos
- Enabled for every message of the entire flow

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part11-enhanced-authentication/

Flow Control

- Flow Control is designed to maintain balanced message processing, preventing the overloading of any participating parties.
- Receive Maximum: Max number of unacknowledged PUBLISH messages the client can accommodate
- "Receive Maximum" value is not specified, the default value of 65,535
- Helps with not overloading low resourced IoT

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part12-flow-control/

Summary - MQTT

- **Lightweight Protocol** for IoT designed for constrained environments.
- **Publish/Subscribe model**: Decouples publishers from subscribers, simplifying communication without requiring direct connections.
- **Three QoS Levels**
- **Flexible and efficient**: Ideal for unreliable or bandwidth-limited networks
- **Security via TLS/SSL**: Secure communication channels to prevent unauthorised access.
- **Widely used** from hobby projects to large-scale IoT solutions.

SINGAPORE INSTITUTE OF TECHNOLOGY

MQTT vs CoAP

RFC 7252

MQTT > CoAP

1. **Simplicity and Ease of Implementation:** MQTT is often perceived as simpler to implement compared to CoAP
2. **Reliability in Unstable Networks**: MQTT QoS levels allow for message delivery guarantees, which is crucial in many IoT applications.
3. **Better Support for High-Latency Networks**: MQTT works well over networks with high latency.
4. **Widespread Adoption and Ecosystem**: MQTT has a broader adoption in the industry.
5. **Scalability**: MQTT's publish-subscribe model scales well for a large number of devices.
6. **Interoperability with Existing Infrastructure**: MQTT’s use of TCP/IP makes it easy to integrate.
7. **Use Cases and Industry Preferences**: Certain industries have preferred MQTT for its reliability and ease of integration, especially in scenarios like telemetry, where data needs to be pushed from a large number of devices to a server.
8. **Strong Community and Standards Support**: MQTT has strong support from organisations like OASIS and the Eclipse Foundation, and it's an ISO standard (ISO/IEC 20922), which adds to its credibility and reliability.

https://www.hivemq.com/blog/mqtt-vs-coap-for-iot/

MQTT < CoAP

1. **Extremely Low-Power Devices**: CoAP is designed for very low-power devices that need to conserve battery life aggressively. It works well in scenarios where devices need to remain in sleep mode for extended periods and wake up only to send or receive brief messages.
2. **Devices with Limited Processing Capabilities**: CoAP is more straightforward and requires less processing power compared to MQTT, which is an advantage in environments with very limited computational resources.
3. **UDP-Based Networks**: CoAP operates over UDP, which is generally lighter on resources than TCP (used by MQTT). In environments where TCP's overhead is undesirable or unsustainable, CoAP is the better choice.
4. **Direct Web Integration**: CoAP's RESTful design and built-in proxying support to HTTP make it inherently suitable for direct integration with web technologies. This is beneficial in constrained environments where devices need to interact directly with web-based services without the additional overhead of a MQTT broker.
5. **Multicast Support**: CoAP supports multicast requests natively, allowing a single message to be sent to multiple devices simultaneously. This is particularly useful in scenarios like firmware updates or mass configuration changes in a constrained network.
6. **Low-Latency Device-to-Device Communication**: In scenarios where devices need to communicate directly with each other with minimal latency, CoAP's lightweight, stateless operation can be advantageous compared to MQTT, which generally requires communication through a broker.

Comparing CoAP with …

	DDS	AMQP	CoAP	MQTT	REST / HTTP	XMPP
TRANSPORT	UDP/IP (unicast + multicast) TCP/IP	TCP/IP	UDP/IP	TCP/IP	TCP/IP	TCP/IP
INTERACTION MODEL	Publish-and-Subscribe, Request-Reply	Point-to-Point Message Exchange	Request-Reply (REST)	Publish-and-Subscribe	Request-Reply	Point-to-Point Message Exchange
SCOPE	Device-to-Device Device-to-Cloud Cloud-to-Cloud	Device-to-Device Device-to-Cloud Cloud-to-Cloud	Device-to-Device	Device-to-Cloud Cloud-to-Cloud	Device-to-Cloud Cloud-to-Cloud	Device-to-Cloud Cloud-to-Cloud
AUTOMATIC DISCOVERY	✓	-	✓	-	-	-
CONTENT AWARENESS	Content-based Routing Queries	-	-	-	-	-
QoS	Extensive (20+)	Limited	Limited	Limited	-	-
INTEROPERABILITY LEVEL	Semantic	Structural	Semantic	Foundational	Semantic	Structural
SECURITY	TLS, DTLS, DDS Security	TLS + SASL	DTLS	TLS	HTTPS	TLS + SASL
DATA PRIORITIZATION	Transport Priorities	-	-	-	-	-
FAULT TOLERANCE	Decentralized	Implementation-Specific	Decentralized	Broker is SPoF	Server is SPoF	Server is SPoF

https://devopedia.org/constrained-application-protocol

つづく