# A 01 - MQTT & IoT protocol stack

## Conventional internet protocol suite

| Conventional Internet protocol suite | | |
|---|---|---|
| Layer | Protocol | Features |
| Application | HTTP | Application-level access to information, client/server |
| Transport | TCP/UDP | Communication channels with guarantees (TCP);<br>just an interface to IP (UDP) |
| Network | IP | Addressing & routing;<br>best-effort |

- |application | HTTP | client/server (or public/subscribe parading in MQTT)
- |transport | TCP/UDP: tcp as extra feature of IP, provide reliability (retransmission, may fail if network turbo congested), udp: interface to IP|
- |network|IP|best effort: the way IP provides addressing/routing (ok or accept it)

## IoT and internet

- things must be connecte to the internet to become IoT devices
- so they must adopt TCP/IP + some application layer (e.g.: HTTP)
- IoT constraints: low power, memory, lossy, with constrained resources, expected to remain alive for years

## IoT requirements

# A 01 - MQTT & IoT protocol stack

## Conventional internet protocol suite

| Conventional Internet protocol suite | | |
|---|---|---|
| Layer | Protocol | Features |
| Application | HTTP | Application-level access to information, client/server |
| Transport | TCP/UDP | Communication channels with guarantees (TCP);<br>just an interface to IP (UDP) |
| Network | IP | Addressing & routing;<br>best-effort |

- |application | HTTP | client/server (or public/subscribe parading in MQTT)
- |transport | TCP/UDP: tcp as extra feature of IP, provide reliability (retransmission, may fail if network turbo congested), udp: interface to IP|
- |network|IP|best effort: the way IP provides addressing/routing (ok or accept it)

## IoT and internet

- things must be connecte to the internet to become IoT devices
- so they must adopt TCP/IP + some application layer (e.g.: HTTP)
- IoT constraints: low power, memory, lossy, with constrained resources, expected to remain alive for years

## IoT requirements

| IoT requirements | |
|---|---|
| **Network requirements** | **Impact on networking** |
| Scalability / redundancy | Multi-hop, mesh networking |
| Security | Configurable, with different security levels for different devices capabilities |
| Addressing | Scalable address space, low-overhead network protocols |
| **Device requirements** | **Impact on application-level** |
| Low power / battery powered | Low duty-cycle communications |
| Limited capacity (memory/processor) | Small footprint, low complexity protocols |
| Low cost | Reliability of the device and further constraints... |

- several IoT requirements that impact solved by IPV6 → scalable address space (addressing network requirement)
- mesh networking: solution for large networks of sensors to provide communication each other
- scalability: in term of big # of device in device and distributed computation
- limited capacity → solution: embedding programming language for IoT and algo with low complexity

# MQTT

- Messages Queuing Telemetery Transport
- an application level protocol for IoT (TCP/IP stack) || on iso/osi cares on session and presentation layer
- lightweight public/subscribe messagging transport protocol
- lightweight:
  - low network bandwith
  - minimal packet overhead (better performance that http)
- widely used
- it is build upon TCP/IP:
  - port 1883 (without security)
  - port 8883 (MQTT over SSL→ adds significant overhead → not really much used)

- problems:
    1. TCP is not the cheapest protocol → UDP is more suitable
    2. big problem: TCP window to deal retransmittion/congestion → large window for IoT → can shrink TCP window
- implements public/subscribe paradingm to user but internally implements client/server architecture
- simple to implement on client side
    - complexity (most of) is on server-side
- provides QoS data delivery
- it is data agnostic: can trasfer any bunch of data → just a sequence of bytes
- it is suitable for Machine to Machine and IoT applications
- widely used to:
    - sensor to satellite
    - home automation
    - e-healt
    - supported by bug player that provides platforms for IoT

## Public/subscribe paradigm

- actors: publishers, subscribers, service broker

- publishers and subscribers are both clients

- they don't know each other

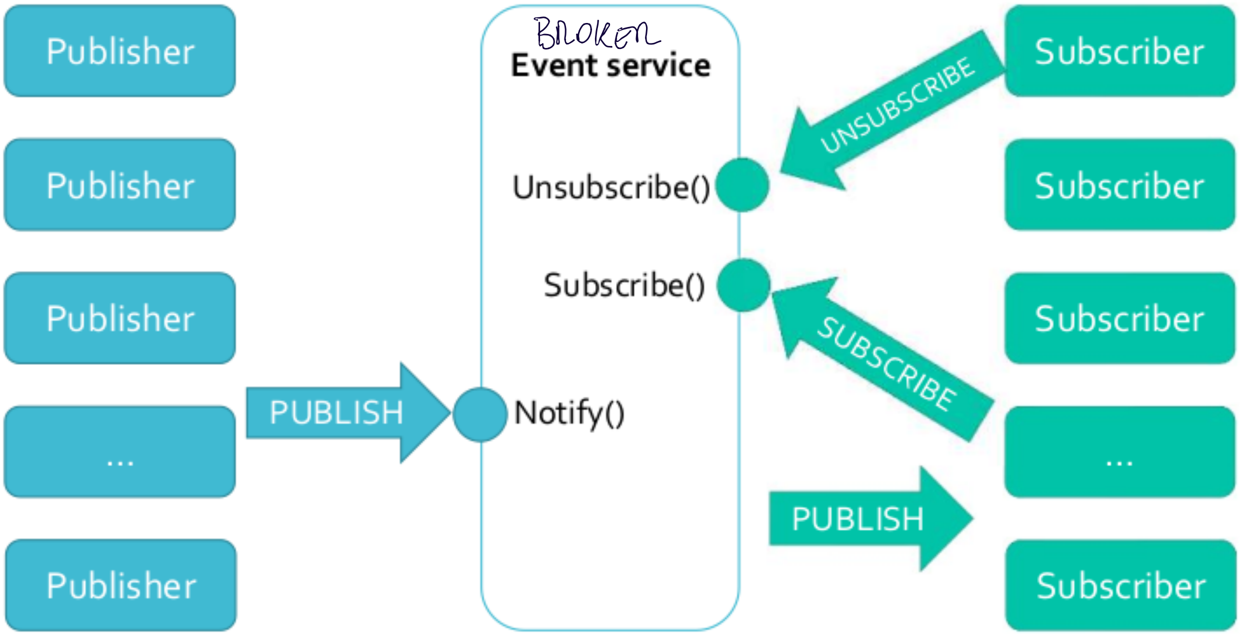
- **publishers**
    - produce events (e.g.: sensors)
    - interact only with broker

- **subscribers**
    - express interest for an event (a type of data)
    - receice notifications whe the event is generated
    - interact only with the broker

- publishers and subscribers are fully decoupled (respect client/server architecture) in time, space, synchronization
    - dont'share routing

- **event service** (aka broker):
  - it's the server
  - known both by publishers and subscribers
  - keep TCP channel to receive all incoming messages from subscribers (allowind decryption)
  - manages the requests of subscription/unsubscription

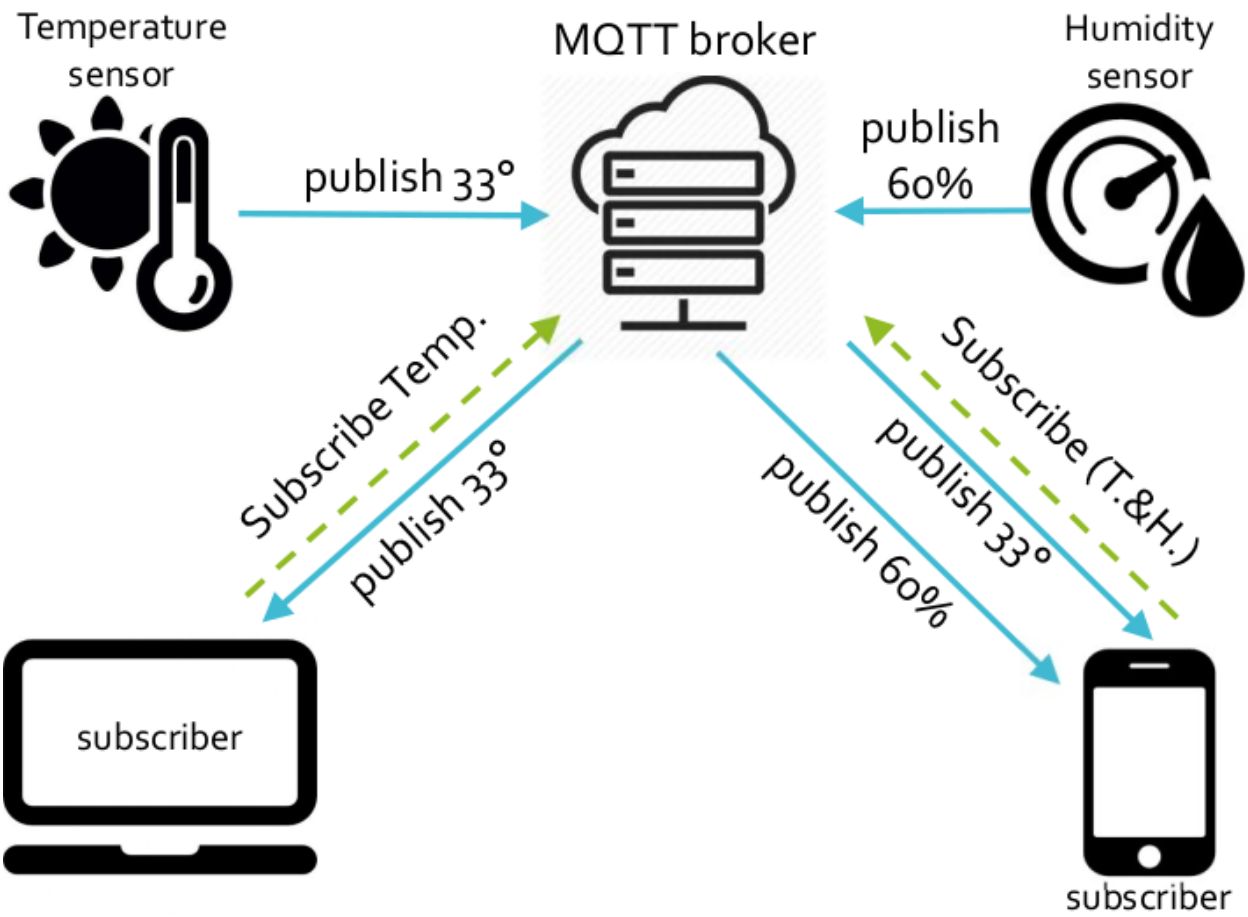
- pub/sub paradigm interaction can be implemented in many different ways:
  - in basic interaction schema:
    - the event service (broker) ~~us~~ is an~~d~~ independent agent
    - operations supported by broker:
    1. publish
    2. subscribe
    3. notify
    4. unsubscribe




- **Event service** (AKA broker): storage and management of subscriptions

- one publisher - - publish (communication)- - > a notify - - publish (forwared to) - - > one subscriber

- example:




- 

- when temperaature sensor publish 33° to MQTT broker and humidity sensor done it with 60%

- PC gets only subscribe temperature from publisher (33°)

- Smartphone gets both (temperature and humidity)

## Publishers/subscribers features:

1) - **space decoupling**(same, no buffer buffer between pub/sub):
    - pub and sub
        - don't need to knows each other and don't share anything
            - don't know IP adress and port of each other
            - don't know how many peer they have
2) - **time decoupling**:
    - pub/sub don't need to run at the same time
    - pub can disapper and reconnect later
3) - **synchronization decoupling**:

    - operations on both pub/sub are not halted during publish or receiving (don't wait for each other)

4) - **scalability**:
    - better than client/server (decoupled)
    - all up to broker
    - operation on broker can be parallelised and are event-driven
    - scalability to a very large number of devices may require parallelization of the broker:
        - manage subset of pub/sub → scale good with MQTT

5) - **filtering**:
    - messages can be filtered at the broker (3 string cases), based on:
        1. **subject topic**: part of the message, client subscribe for specift topic(topics are strings)
        2. **content**: client subscribe for a specific query (e.g.: temperature > 30°); data cannot being encrypted (it may → need keys → need introduce third part)
        3. **type**: filtering events based on both *content* and *structure*, type refers to the typle/class of data;