IV-Edge to Cloud Protocol

IV	**Edge to Cloud Protocol**	HTTP, WebSocket, Platforms. HTTP - MQTT -.Complex Flows: IoT Patterns: Real-time Clients, MQTT, MQTT-SN, Constrained Application Protocol (CoAP), Streaming Text Oriented Message Protocol (STOMP), Advanced Message Queuing Protocol (AMQP), Comparison of Protocols. • Working of protocol • In which application to use it • Comparison with other protocols	8	CO4

Book :

David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Robert Barton, Jerome Henry, IoT Fundamentals Networking Technologies, Protocols, and Use Cases for the Internet of Things CISCO , Ch 6

Analytics for the Internet of Things (IoT) Intelligent Analytics for Your Intelligent Devices.Andrew Minteer,Packet , ch 2

https://stomp.github.io/stomp-specification-1.2.html
http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
https://drive.google.com/file/d/1zXyc_IyobI5rk7ZdTQD6W118tKFeiCxA/view?usp=sharing

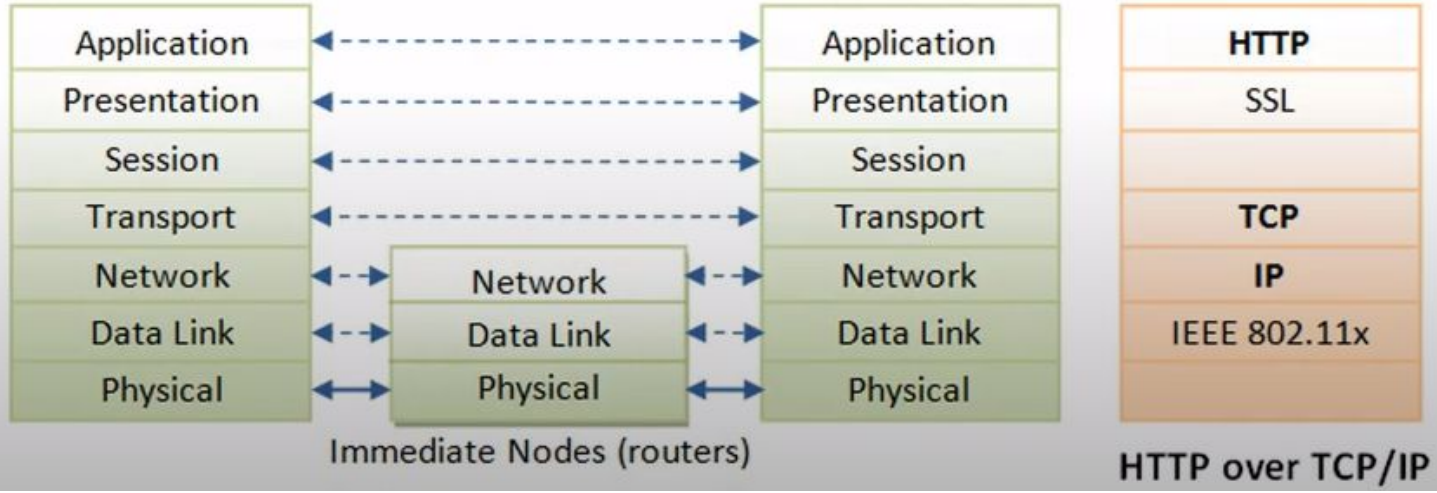
Blooms level : L1,L2,L3

HTTP

- HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).
- HTTP is an *asymmetric request-response client-server* protocol
- HTTP is state-less

HTTP over TCP/IP

- HTTP runs over TCP/IP in most of the scenario to ensure pacl delivery guarantee.
- TCP/IP is a Transport and Network Layer protocol used to communicate between two machines



How HTTP works in IoT

- REST – **RE**presentational **S**tate **T**ransfer is a standard/**design** to communicate between systems on the Web whereas **HTTP** is a **implementation** based on REST standard.
- If any architecture in the web follows REST standards it is said to be RESTful

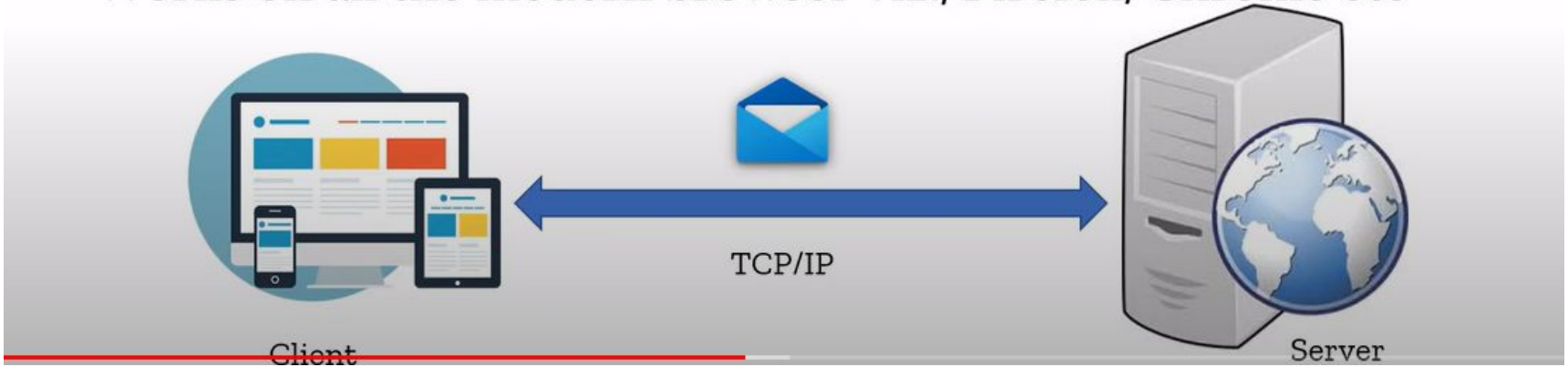
Client

REST API
GET
PUT
DELETE

Smart bulb supporting HTTP

WebSocket

- **WebSocket** is a communication protocol , providing full-duplex communication channels over a single TCP/IP connection between the client and the server.
- Located at layer 7 in the OSI model and depend on TCP at layer 4
- Works on all the modern browser viz., Firefox, Chrome etc



Why WebSocket ?

- It uses standard internet and web technologies
- WebSocket aren't blocked by firewalls and can traverse proxies
- Useful for real time communications with out any delay in IoT applications
- Very less bandwidth than HTTP polling
- Faster than HTTP

Why not HTTP for IoT?

- HTTP requires a connection as it runs on TCP
- HTTP is document centric. But in IoT all we want is to send data, not documents

Edge to Cloud Protocol

- MQTT - Message Queuing Telemetry Transport
- MQTT-SN - publish/subscribe messaging protocol for wireless sensor networks (WSN),
- CoAP - Constrained Application Protocols
- STOMP - Streaming Text Oriented Message Protocol
- AMQP - Advanced Message Queuing Protocol

https://pressbooks.bccampus.ca/cellulariot/chapter/chapter-4/

Why do we need different protocols for IoT devices?

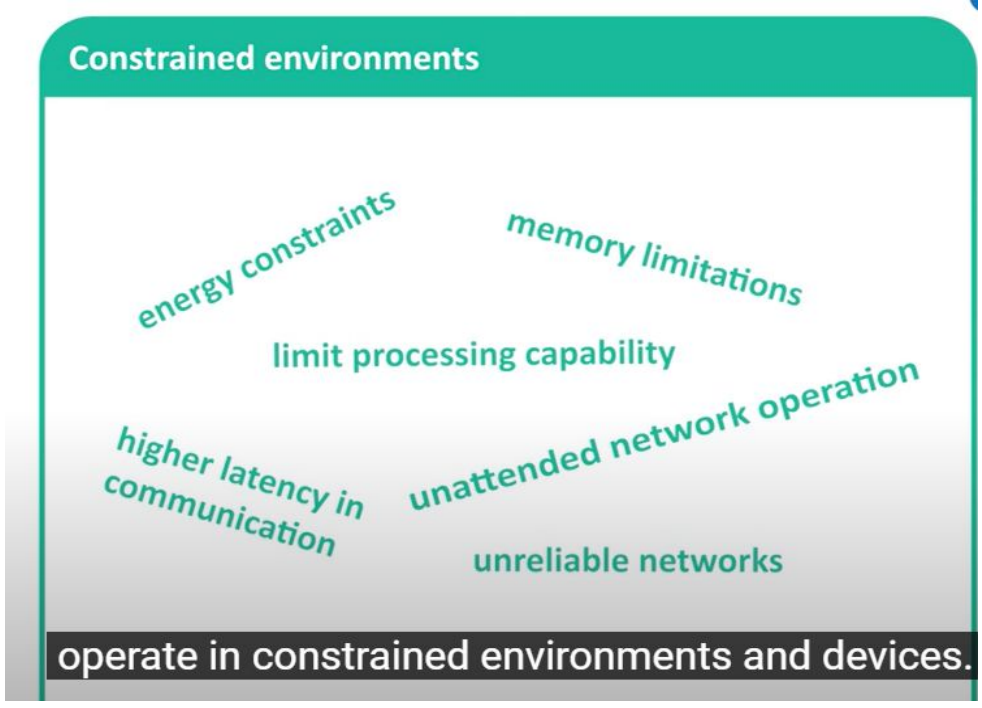
- The rapid development of the Internet and the associated technologies that go with it has led to a surge in IoT devices.
- Almost everything, from your spectacles to your shoes, is now connected to the Internet.
- But not every device has the same processing capacity.
- Add to that, Internet connectivity still remains a major issue across the globe.
- Even powered with the latest 5G Internet, your device is likely to suffer some sort of connectivity issues.
- While such limitations have been a good thing, in the sense that they fuelled faster development of revolutionary technologies, they have also given rise to new ways of communication over the Internet.

- Since all IoT devices cannot communicate with each other on the same capacity and capability, and have different system based requirements, we require different IoT protocols to enable smooth communication and transactions.

- There are also some product-specific issues that developers come across such as the size of the device, which limits the number of chips you can fit into it. Take for instance, the latest wireless Bluetooth earbuds. The Bluetooth connectivity chips used in those devices need to be small enough to fit inside an ear and strong enough to stay connected to your phone or computer at a longer distance.

Summary

- processing capacity
- Internet connectivity still remains a major issue
- IoT devices cannot communicate with each other on the same capacity and capability
- different system based requirements
- require different IoT protocols to enable smooth communication and transactions.
- product-specific issues
- Size
- Frequency of the data
- Critical data
- Continuity in connection
- etc



Constrained Networks

- Low power
- Low bandwidth
- Lossy
- High Latency

Constrained Devices

- Small Embedded Devices
- Runs on batteries and need to be energy efficient
- Small ROM and RAM

e talking of small, constrained embedded devices running on batteries and with very low ROM and RAM.

Why not HTTP for IoT?

- HTTP requires a connection as it runs on TCP
- HTTP is document centric. But in IoT all we want is to send data, not documents

HTTP

HTTP
TCP
IP

CoAP

CoAP
UDP
IP

Features of CoAP

- A connectionless protocol
- A lightweight protocol
- An asynchronous protocol
- A RESTful protocol
- Has a small and simple 4 byte header

Why CoAP is ideal for IoT?

- CoAP is connection-less as it runs on UDP
- It is meant for machine to machine communication by design

These questions will be asked in case of every protocol…

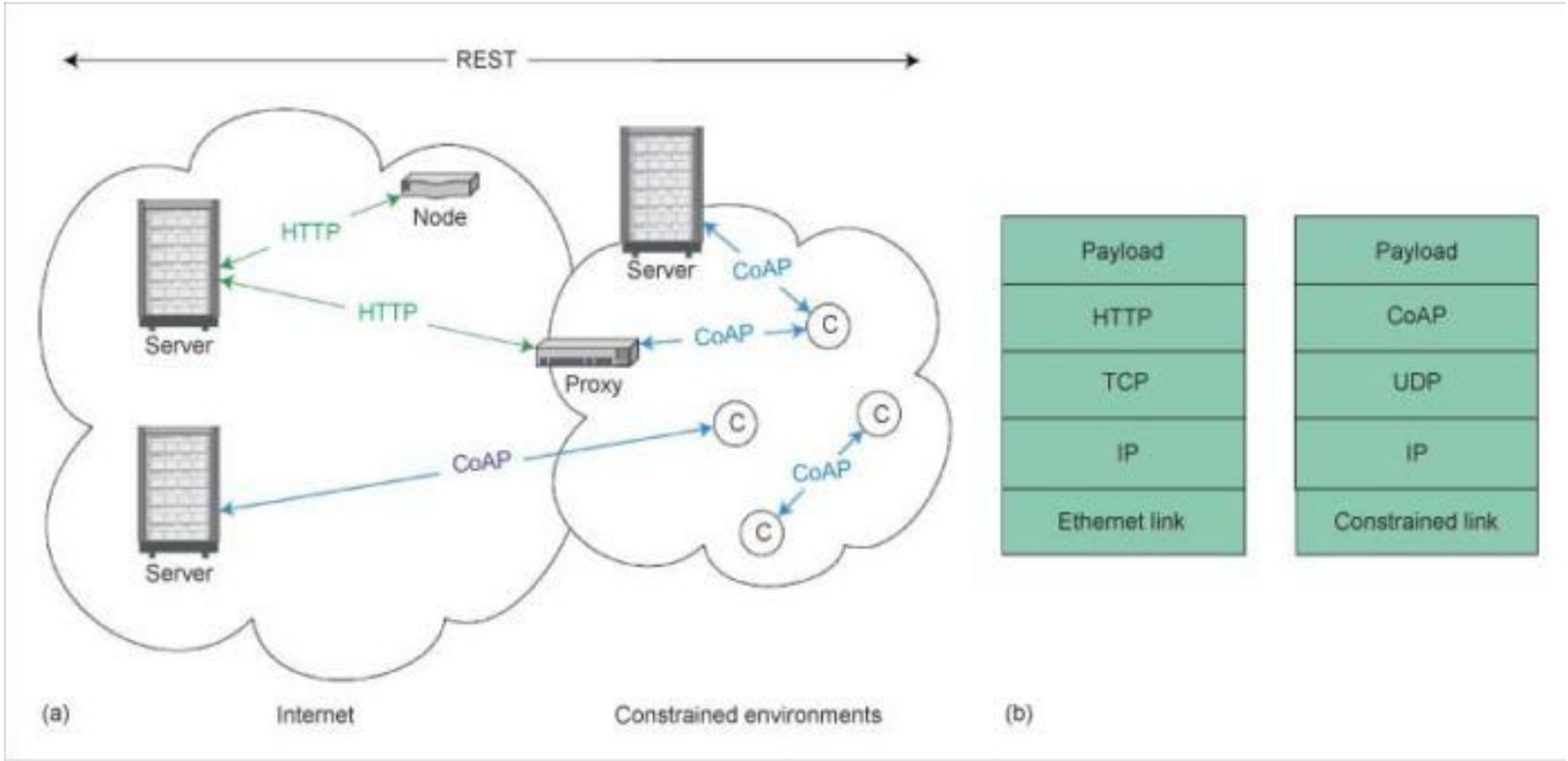
- Security
- Lightweight
- Speed
- Connection
- Confirmation - guaranteed delivery
- etc

Is CoAP reliable?

- Comes with optional reliability, even if built on top of UDP
- Comes with 4 kinds of messages - confirmable and non-confirmable, acknowledgement and reset
- Request and Response are carried on confirmable and non-confirmable messages

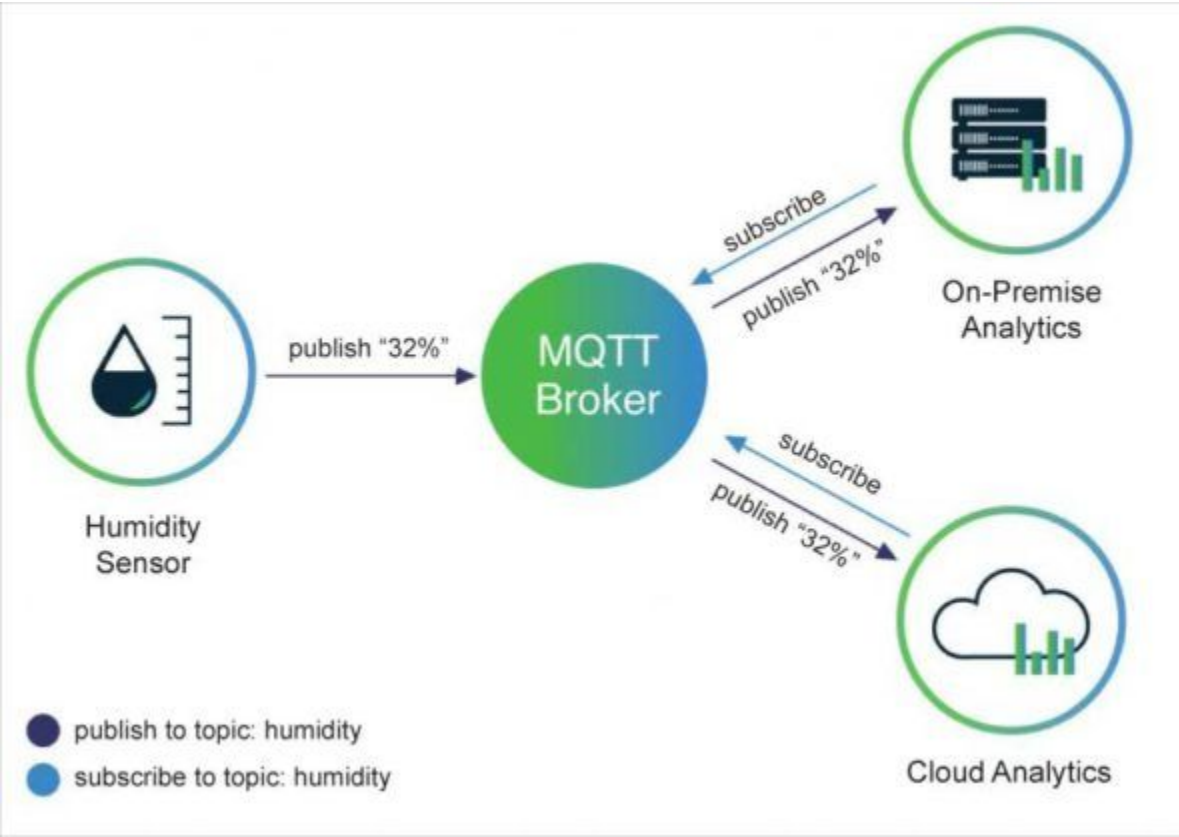
Constrained application protocol (CoAP)

- When it comes to communication over the Internet, there is **no protocol like the HTTP.**
- With its underlying REST framework and simplicity, HTTP is **highly reliable and easy to implement.**
- The trouble however is that it can **prove heavy for most IoT devices, as these generally require a more lightweight protocol.**
- So can there be a protocol that has all the goodness of HTTP in addition to being lightweight and easy to implement?
- That is the problem that the **IETF (Internet Engineering Task Force)** Constrained RESTful environments working group tried to solve by creating CoAP in 2013.
- CoAP uses the user datagram protocol (UDP) for establishing communication between IoT devices, unlike HTTP which uses TCP.
- CoAP uses a RESTful architecture and uses the HTTP GET, POST, PUT and DELETE methods.
- The beauty of CoAP, which distinguishes it from HTTP, is that UDP allows the IoT devices to **communicate without having to establish a prior connection**.
- This means that an IoT device using CoAP can **relay multiple messages to multiple devices without having to wait for a handshake.**
- This makes the communication **faster while continuing to use very low bandwidth**.
- To mitigate the risk of messages getting dropped due to a lack of connection, **CoAP uses a confirmable/non-confirmable feature, using which messages that have been delivered can be marked confirmable.**
- CoAP also uses **datagram transport layer security (DTLS)** for secure exchange of messages in the transport layer.


REST
HTTP
Node
Server
HTTP
Server
CoAP
CoAP
C
Proxy
C
C
CoAP
CoAP
C
Server
(a)
Internet
Constrained environments
(b)
Payload
HTTP
TCP
IP
Ethernet link
Payload
CoAP
UDP
IP
Constrained link

Message queuing telemetry transport (MQTT)

- Perhaps the **most widely used protocol in industry level IoT products**, MQTT uses a publish/subscribe model for transacting data between different devices.
- In this model, devices are divided into three types, namely, the **producers, consumers and brokers.**
- Producers publish data under what are called topics.
- A topic is basically a channel under which related data can be published by the producer.
- The consumer devices subscribed to that particular channel then receive the message.
- The entire transmission of data from a publisher to its subscribers is made possible with the help of brokers.
- Brokers receive messages from producers that they then filter into designated topics and share with the subscribers for each of the topics.
- MQTT uses the TCP/IP protocol to enable communication between multiple devices.
- What makes it ideal for industry level products is that it simplifies sending and receiving messages between multiple devices.
- In the example, the humidifier is the producer that publishes data to the broker under separate topics. The consumers, which in this example are the on-premise analytics and the cloud, receive the data from the broker for their respective topics. So the producer publishes the data to the broker, and the broker publishes the data to the subscribers.



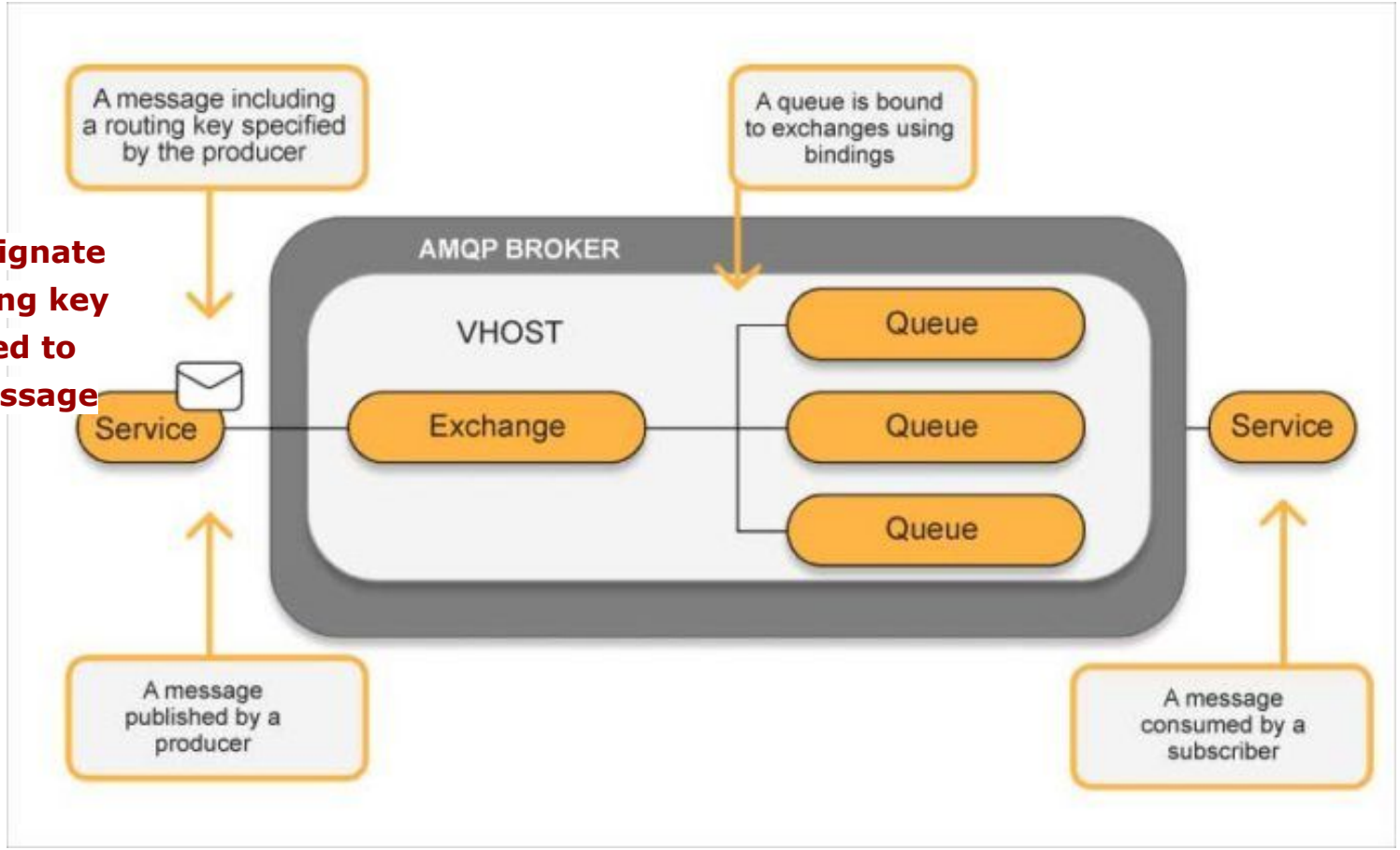
- Scalability
- Latency
- Lack of security encryption are the major drawbacks

Advanced message queuing protocol (AMQP)

- AMQP is **another publish/subscribe model based protocol** that is primarily focused **on guaranteed completion of transactions**. Designed back in 2003, it was used mostly in the financial sector for obvious reasons.
- AMQP makes use of **advanced queuing and distribution techniques to ensure security, reliability and speed**.
- the **publisher publishes the message to the AMQP broker with a predesignated routing key attached to the message**.
- Utilising the routing key, **the broker queues the message for delivery to the designated consumer.**
- These messages are **bound to exchanges with the help of bindings**.
- There are, however, a few drawbacks of the AMQP model which has restricted its mass adoption in the IoT space.
    - For one, it is a **heavy protocol that is not ideal for smaller IoT devices** with a **lack of bandwidth and power**.
    - Second, **the guaranteed delivery of messages is not always a high priority for all IoT projects.**

AMQP

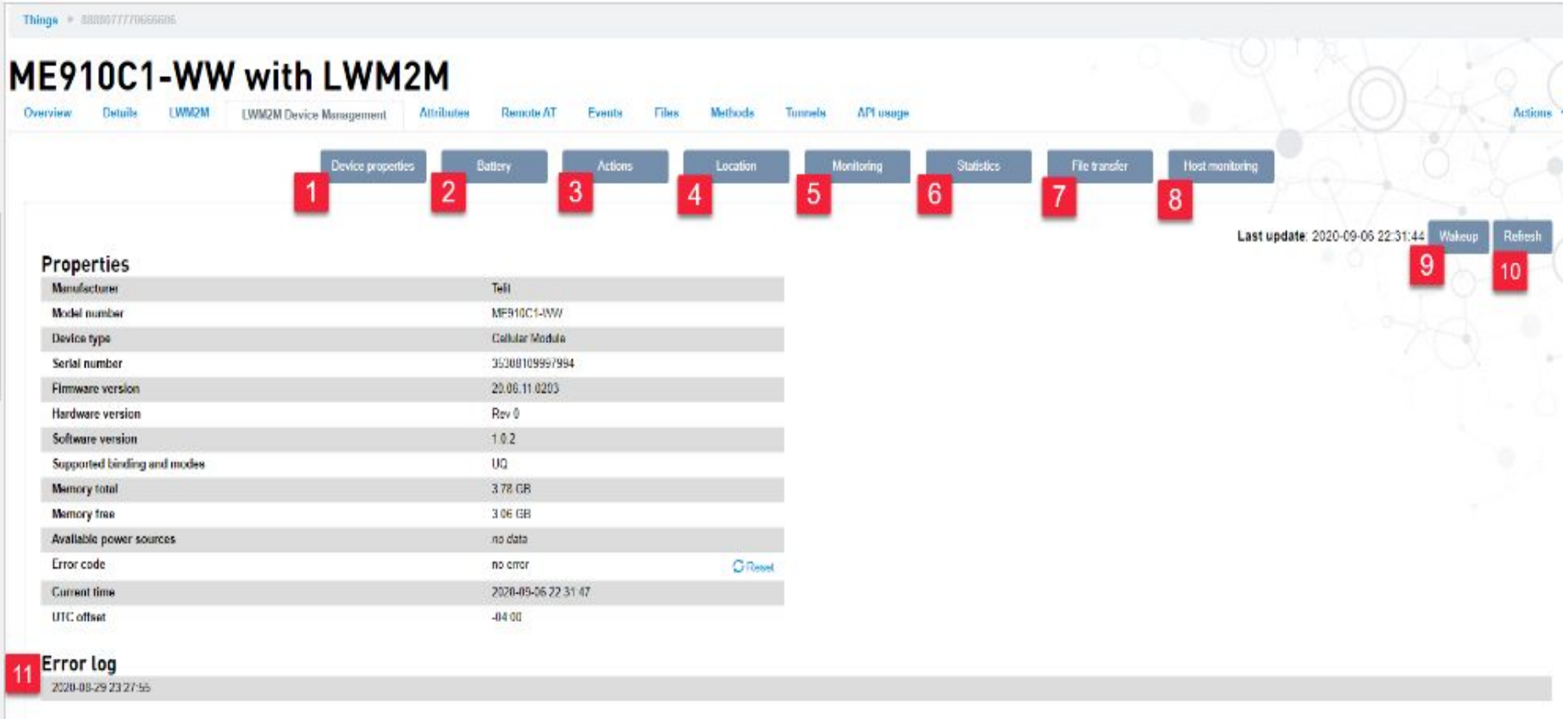
predesignated routing key attached to the message



Lightweight M2M (LwM2M)

- Designed by the Open Mobile Alliance (OMA) in 2014, LwM2M has been specifically designed to **tackle the challenges of modern day IoT systems.**
- With over 75 billion working IoT devices estimated across the world by 2025, LwM2M has been designed for s**calability, speed and reliability in a constrained environment.**
- Lightweight M2M (LWM2M) **is a traffic and resource-optimized protocol to remotely manage IoT devices**. The protocol is standardized by the Open Mobile Alliance.
- As shown in Figure 7, the **LwM2M server stack consists of all the essential components that an IoT network requires including CoAP protocol, UDP protocol, SMS bearer and an efficient payload system.**
- It also has DTLS security inbuilt. Additionally, LwM2M provides a **device level bootstrapping facility, which gives you the ability to customise the usage of the protocol as per your requirements on a local level.**
- It also provides a **client registration interface, and access to resources and instances**. This gives the user access to customise the settings on a granular level without having to revert to factory settings.

The LWM2M Device Management tab is available for all the LWM2M devices.



1 Device properties

The device properties list the following resources of the LWM2M device.

Resources	Description
Manufacturer	Human readable manufacturer name
Model number	A model identifier (manufacturer specified string)
Device type	Type of the device (manufacturer specified string: e.g. smart meters)
Serial number	The serial number of the device
Firmware version	The current firmware version on the device
Hardware version	The current hardware version of the device
Software version	The current software version on the device
Supported binding and modes	Indicates which bindings and modes are supported in the LWM2M Client device. For example, UQS (UDP with Queue Mode and SMS).
Memory total	Total amount of storage space which stores data and software in the LWM2M Device (expressed in kilobytes).
Memory free	Estimated current available amount of storage space which stores data and software in the LWM2M Device
Available power source	The available power sources. • 0 – DC power

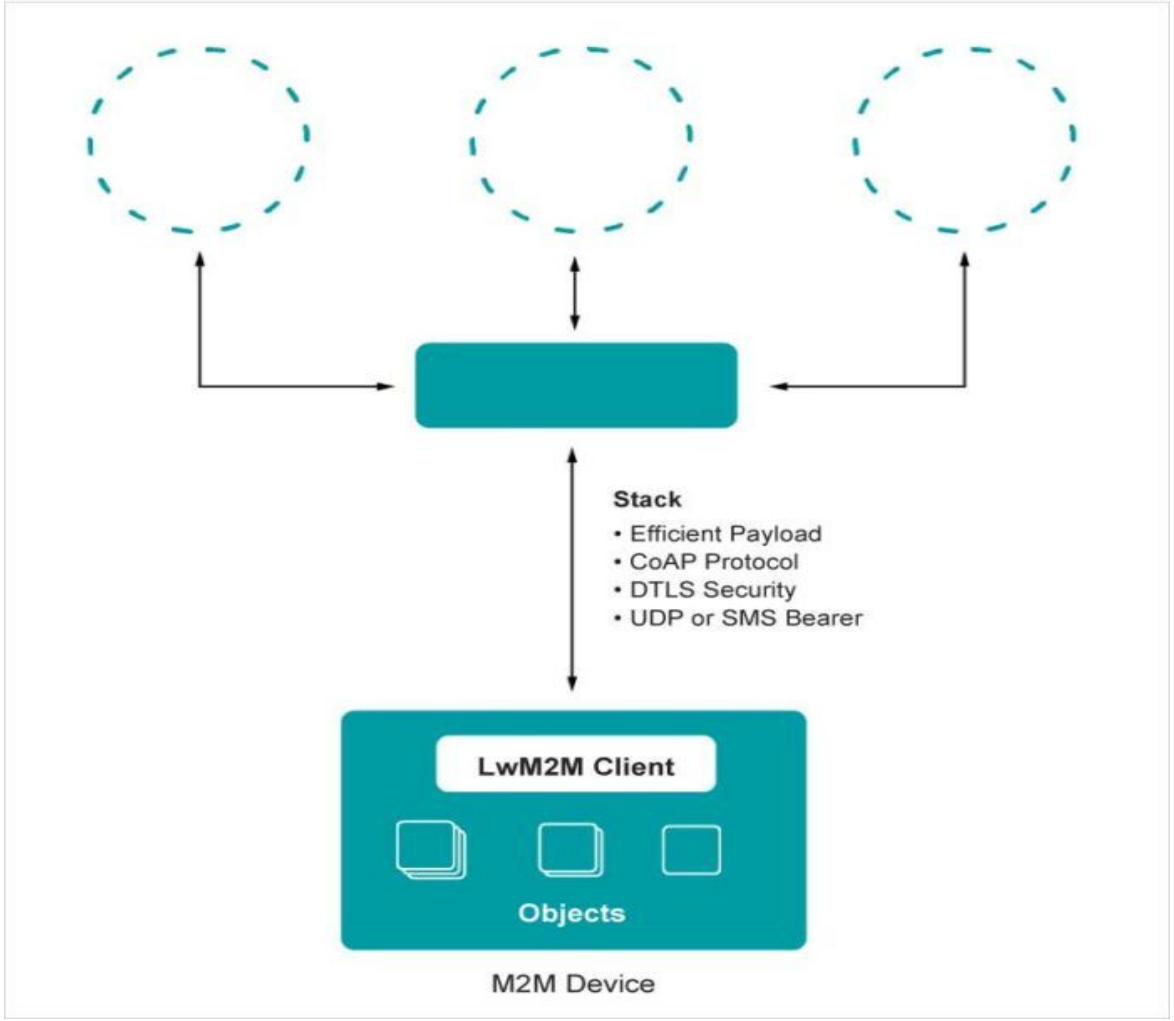
Available power source	The available power sources. • 0 – DC power • 1 – Internal Battery (0 or 1 only) • 2 – External Battery • 4 – Power over Ethernet • 5 – USB • 6 – AC (Mains) power • 7 – Solar
Current time	The current UNIX time of the LWM2M client device. The LWM2M server writes the time to the LWM2M client to get it synchronized.
UTC offset	Indicates the UTC offset currently in effect for this LWM2M Device. UTC+X [ISO 8601]

Battery status	The status of the device internal battery. The possible values include: • *normal* - The battery is operating normally and not on power • *charging* - The battery is currently charging • *charge complete* - The battery is fully charged and still on power • *damaged* - The battery is damaged • *low battery* - The battery is low on charge • *not installed* - The battery is not installed • *unknown* - The battery information is not available
Power source voltage (mV)	Present voltage for each Available Power Sources Resource Instance.
Power source current (mA)	Present current for each Available Power Source.

3 Actions

The Actions contain all the executable resources for the device.

Resources	Description
Reboot	Reboot the LWM2M Device to restore the Device from unexpected firmware failure.
Factory Reset	Perform factory reset of the LWM2M Device to make the LWM2M Device have the same configuration as at the initial deployment.
Reset Error Code	Delete all error code Resource Instances and create only one zero-value error code that implies no error.


Stack
• Efficient Payload
• CoAP Protocol
• DTLS Security
• UDP or SMS Bearer
LwM2M Client
Objects
M2M Device

Which IoT protocol is right for your project?

- What is interesting about IoT, unlike other tech projects, is that the **requirements vary a lot from project to project**, not only **in terms of hardware but also software.**
- For instance, an **internal lighting system might require a connection with the cloud** while **another similar lighting system might require a more localised solution**.
- In some projects, **scalability and speed are the top priorities** while for others, **guaranteed completion of transactions and security might be the focus**.
- As discussed in earlier sections, **protocols play a critical role in facilitating such requirements**.
- **If the hardware is the body, the protocol is the brain of the device.**
- Keeping this in mind, **it is recommended to do a detailed audit and review of the project requirements.**
- It is also important to research industry level IoT projects similar to yours and see which protocol they use. Here's a list of points to consider in this regard.

1. If using a RESTful framework with all the goodness of HTTP is a major requirement for your project, **then you cannot go wrong with CoAP**. CoAP provides all the features of HTTP, with the **additional speed and reliability provided by UDP and message labelling.**

2. If your IoT devices have **major battery related constraints and you require a lightweight protocol**, then **MQTT is more suited for your project**.

It is important to keep in mind that **MQTT does suffer with latency issues**; so use this protocol only **if high latency is not a major concern for your project**.

3. If your **project is data-intensive and requires fast transaction of messages between multiple devices**, then **XMPP (Extensible Messaging and Presence Protocol) might be a protocol worth considering**. But do this only **if the security of the data is not a major concern as XMPP does not have any inbuilt security features.** Needless to say, **if your project uses XML, then the go-to protocol should be XMPP**.

4. If what **you are working on are large, high-performance sensor networks such as the ones used in self-driving cars and smart air-traffic control systems**, then **DDS is probably the ideal choice**.

However, due diligence is necessary before selecting any protocol, and this is even more true when you are dealing with such advanced projects. **DDS is a heavyweight protocol, and requires your devices to have a decent battery life and bandwidth to be able to function.**

The Data Distribution Service for Real-Time Systems (DDS) is an Object Management Group (OMG) Machine-to-machine middleware "m2m" standard that aims to **enable scalable, real-time, dependable, high-performance and interoperable data exchanges between publishers and subscribers**.

5. **AMQP is a protocol that can be considered for projects that are end-to-end applications — for instance, heavy industrial machinery that has the capacity to handle data-intensive tasks without the concern for power consumption.** If **transaction completion is an absolute necessity for your project, then AMQP is the most reliable protocol for it.**

6. **LwM2M is the ideal protocol** for **telemetry and device management based IoT projects.** You can create i**ndustry-level large scale IoT projects with high scalability, speed and security.** You can consider using it for a wide array of projects with varying requirements and consumer bases.

https://www.opensourceforu.com/2022/02/choose-the-most-optimal-iot-protocol-for-your-project/

https://www.techtarget.com/iotagenda/tip/Top-12-most-commonly-used-IoT-protocols-and-standards