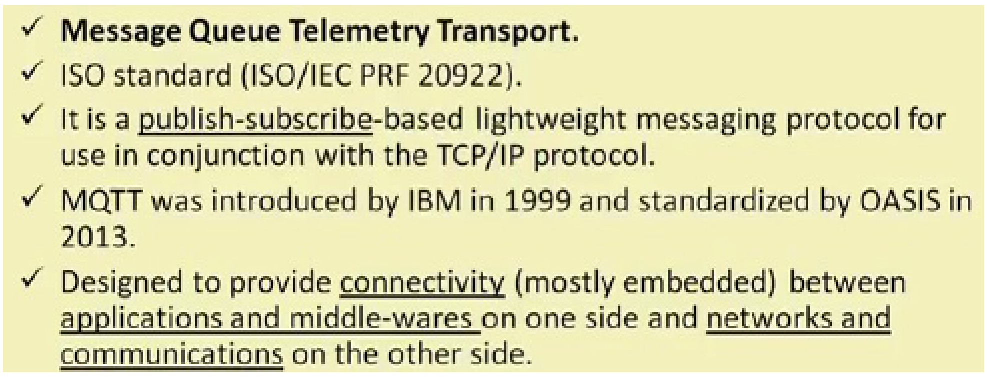
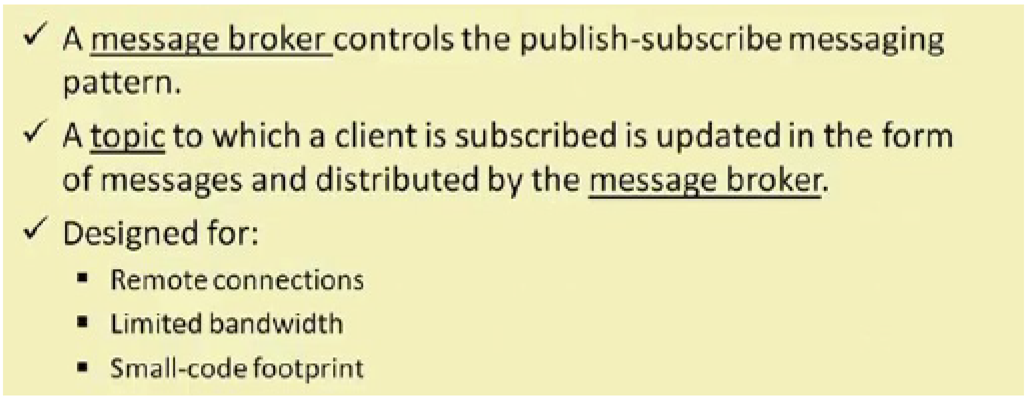
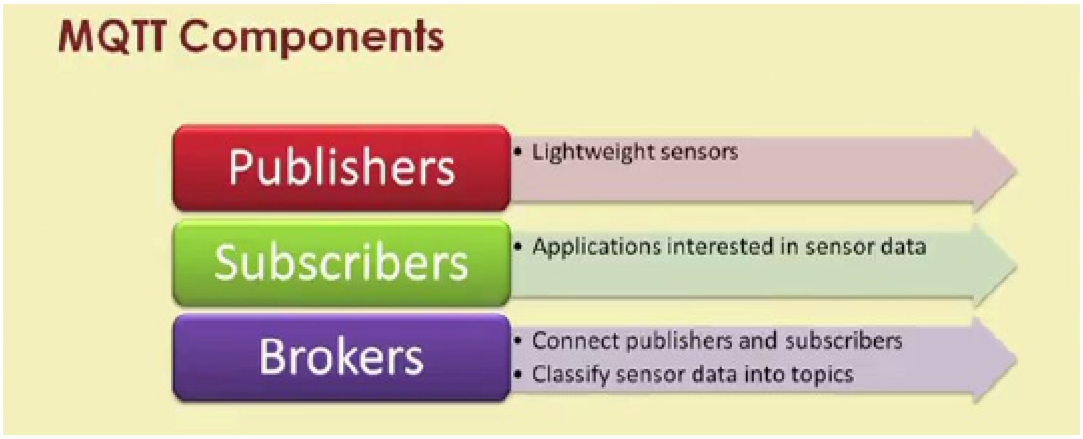
MQTT: Message queue telemetry transport or MQTT is a simple, lightweight publish–subscribe protocol, designed mainly for messaging in constrained devices and networks. It provides a one-to-many distribution of messages and is payload content agnostic. MQTT works reliably and flawlessly over high latency and limited bandwidth of unreliable networks without the need for significant device resources and device power.



- ✓ **Message Queue Telemetry Transport.**
- ✓ ISO standard (ISO/IEC PRF 20922).
- ✓ It is a publish-subscribe-based lightweight messaging protocol for use in conjunction with the TCP/IP protocol.
- ✓ MQTT was introduced by IBM in 1999 and standardized by OASIS in 2013.
- ✓ Designed to provide connectivity (mostly embedded) between applications and middle-wares on one side and networks and communications on the other side.



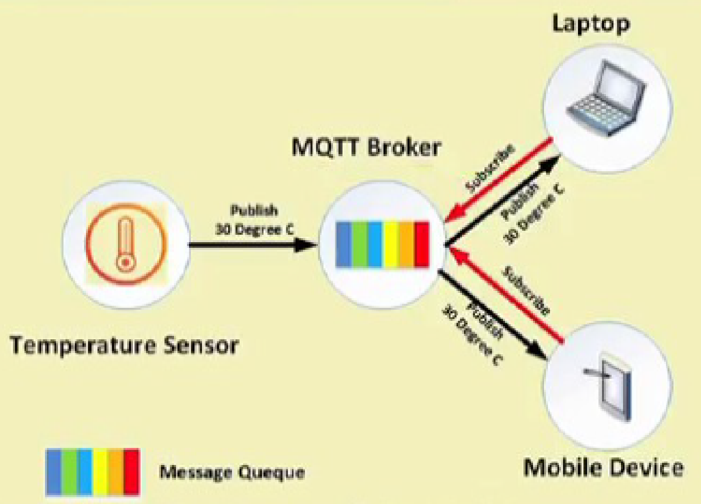
- ✓ A message broker controls the publish-subscribe messaging pattern.
- ✓ A topic to which a client is subscribed is updated in the form of messages and distributed by the message broker.
- ✓ Designed for:
  - Remote connections
  - Limited bandwidth
  - Small-code footprint


MQTT Components
Publishers
• Lightweight sensors
Subscribers
• Applications interested in sensor data
Brokers
• Connect publishers and subscribers
• Classify sensor data into topics

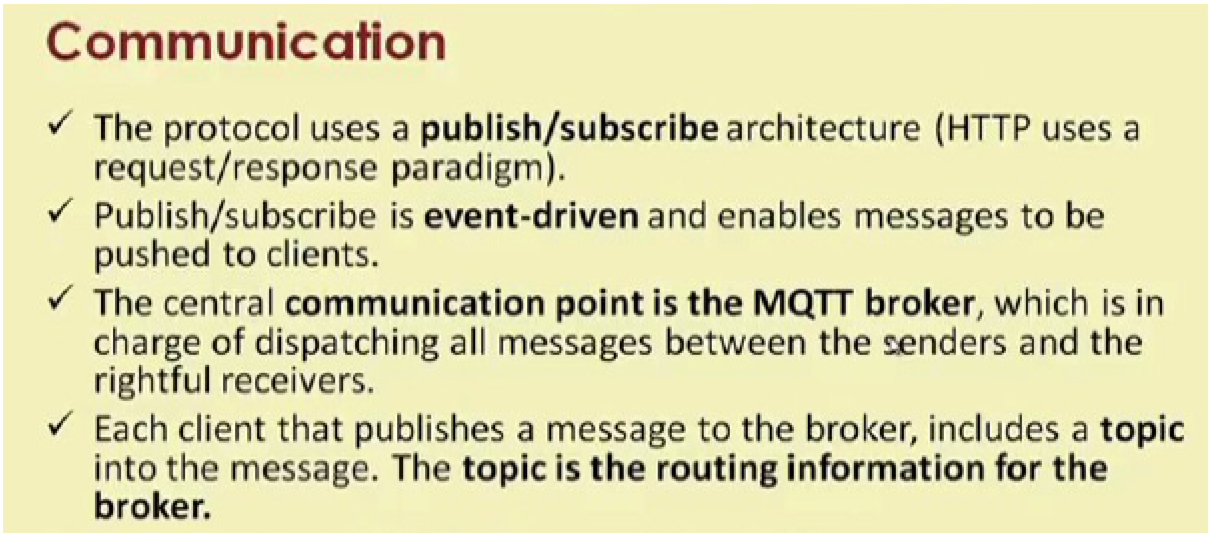
MQTT Methods
Connect
Disconnect
Subscribe
Unsubscribe
Publish

MQTT's control message sizes can range between 2 bytes to 256 megabytes of data, with a fixed header size of 2 bytes. This enables the MQTT to reduce network traffic significantly. The connection credentials in MQTT are unencrypted and often sent as plain text. The responsibility of protecting the connection lies with the underlying TCP layer. The MQTT protocol provides support for 14 different message types, which range from connect/disconnect operations to acknowledgments of data.

The following are the standard MQTT message types:
(i) CONNECT: Publisher/subscriber request to connect to the broker.
(ii) CONNACK: Acknowledgment after successful connection between publisher/subscriber and broker.
(iii) PUBLISH: Message published by a publisher to a broker or a broker to a subscriber.
(iv) PUBACK: Acknowledgment of the successful publishing operation.
(v) PUBREC: Assured delivery component message upon successfully receiving publish.
(vi) PUBREL: Assured delivery component message upon successfully receiving publish release signal.
(vii) PUBCOMP: Assured delivery component message upon successfully receiving publish completion.
(viii) SUBSCRIBE: Subscription request to a broker from a subscriber.
(ix) SUBACK: Acknowledgment of successful subscribe operation.
(x) UNSUBSCRIBE: Request for unsubscribing from a topic.
(xi) UNSUBACK: Acknowledgment of successful unsubscribe operation.
(xii) PINGREQ: Ping request message.
(xiii) PINGRESP: Ping response message.
(xiv) DISCONNECT: Message for publisher/subscriber disconnecting from the broker.

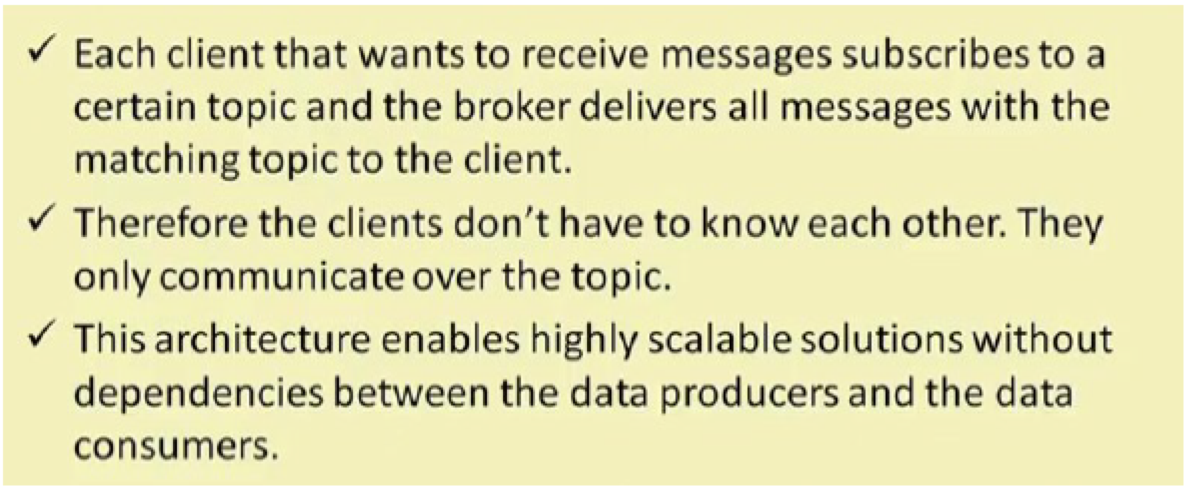


Source: "MQTT 101 – How to Get Started with the lightweight IoT Protocol", HiveMQ (Online)

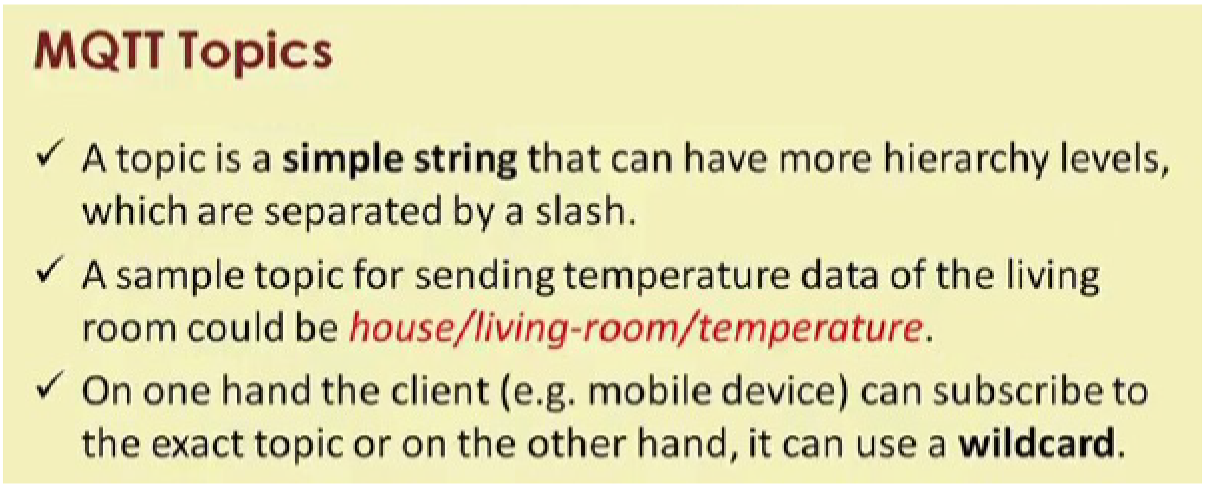


Communication

- ✓ The protocol uses a **publish/subscribe** architecture (HTTP uses a request/response paradigm).
- ✓ Publish/subscribe is **event-driven** and enables messages to be pushed to clients.
- ✓ The central **communication point is the MQTT broker**, which is in charge of dispatching all messages between the senders and the rightful receivers.
- ✓ Each client that publishes a message to the broker, includes a **topic** into the message. The **topic is the routing information for the broker.**

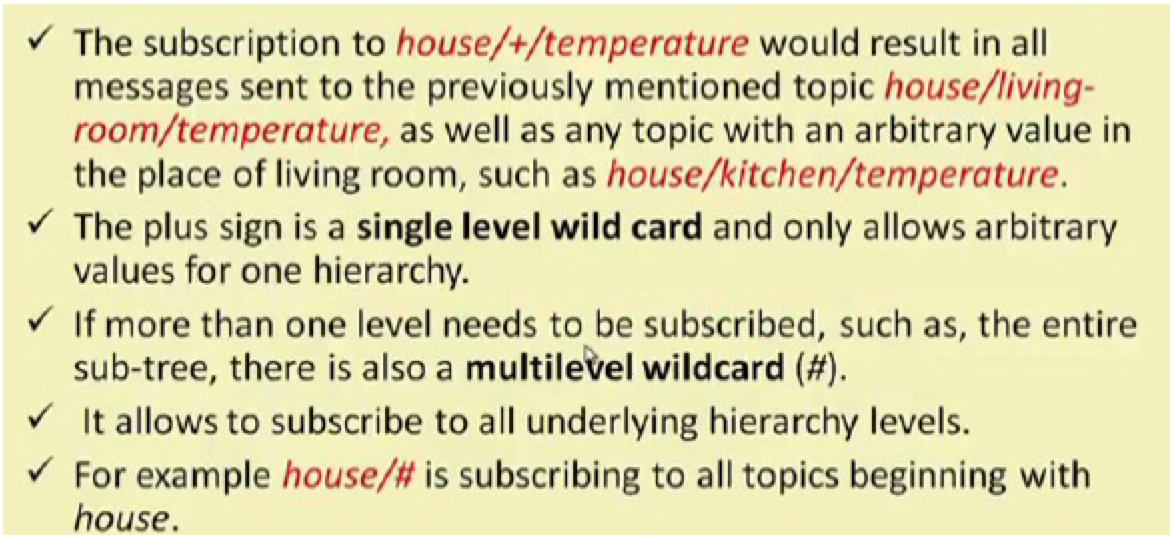


- ✓ Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with the matching topic to the client.
- ✓ Therefore the clients don't have to know each other. They only communicate over the topic.
- ✓ This architecture enables highly scalable solutions without dependencies between the data producers and the data consumers.

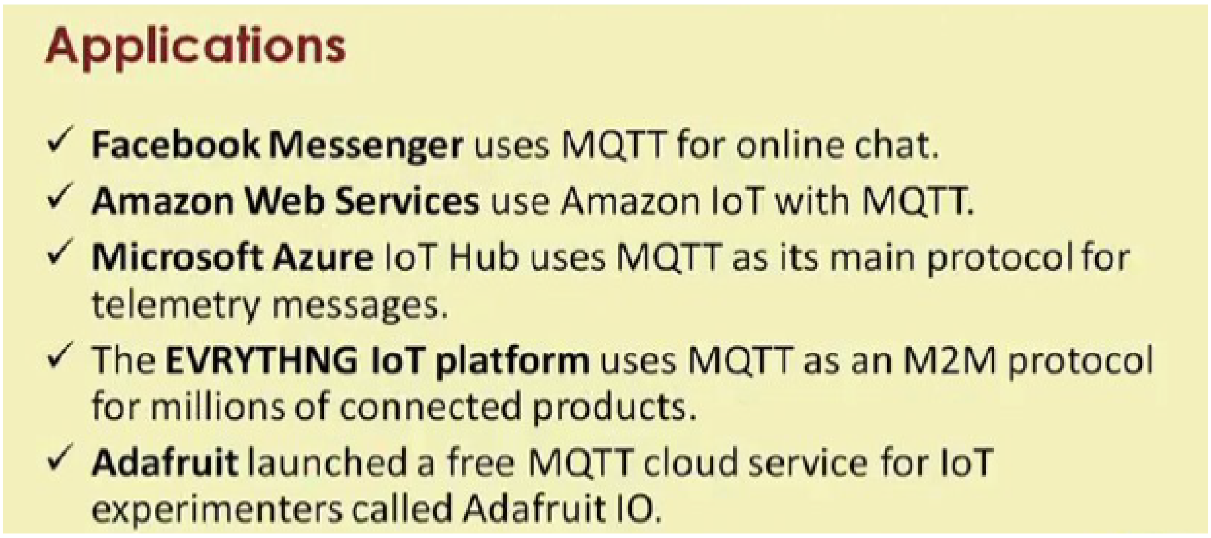


MQTT Topics

- ✓ A topic is a **simple string** that can have more hierarchy levels, which are separated by a slash.
- ✓ A sample topic for sending temperature data of the living room could be *house/living-room/temperature*.
- ✓ On one hand the client (e.g. mobile device) can subscribe to the exact topic or on the other hand, it can use a **wildcard**.



- ✓ The subscription to *house/+/temperature* would result in all messages sent to the previously mentioned topic *house/living-room/temperature,* as well as any topic with an arbitrary value in the place of living room, such as *house/kitchen/temperature*.
- ✓ The plus sign is a **single level wild card** and only allows arbitrary values for one hierarchy.
- ✓ If more than one level needs to be subscribed, such as, the entire sub-tree, there is also a **multilevel wildcard** (#).
- ✓ It allows to subscribe to all underlying hierarchy levels.
- ✓ For example *house/#* is subscribing to all topics beginning with *house*.



Applications

- ✓ **Facebook Messenger** uses MQTT for online chat.
- ✓ **Amazon Web Services** use Amazon IoT with MQTT.
- ✓ **Microsoft Azure** IoT Hub uses MQTT as its main protocol for telemetry messages.
- ✓ The **EVRYTHNG IoT platform** uses MQTT as an M2M protocol for millions of connected products.
- ✓ **Adafruit** launched a free MQTT cloud service for IoT experimenters called Adafruit IO.

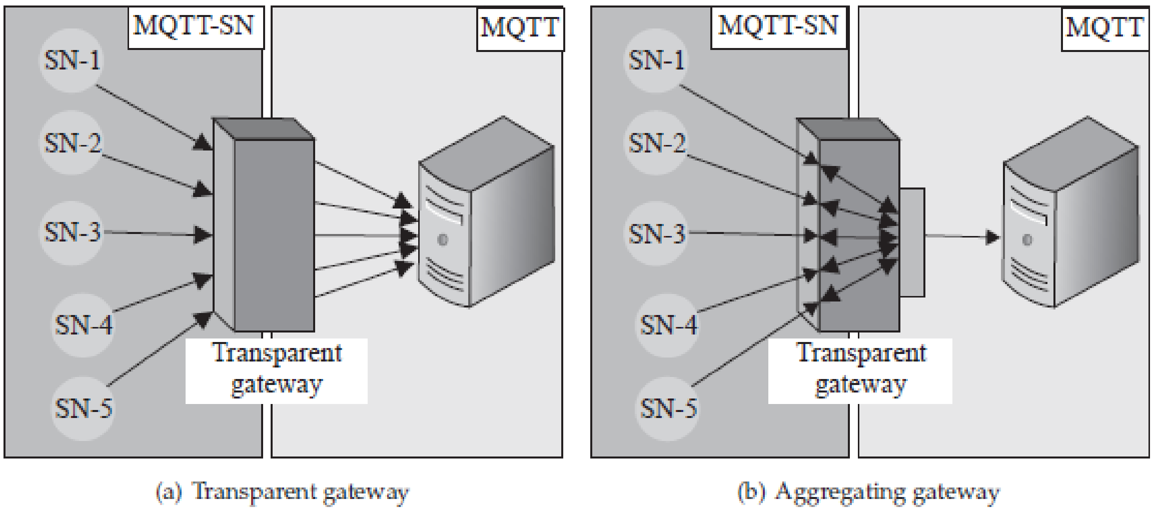
MQTT-SN

The primary MQTT protocols heavily inspire **MQTT for sensor networks or MQTTSN**; however, the MQTT-SN is robust enough to handle the requirements and challenges of wireless communications networks in sensor networks. Typical features of MQTT-SN include low bandwidth usage, ability to operate under high link failure conditions; it is suitable for low-power, low-cost constrained nodes and networks.

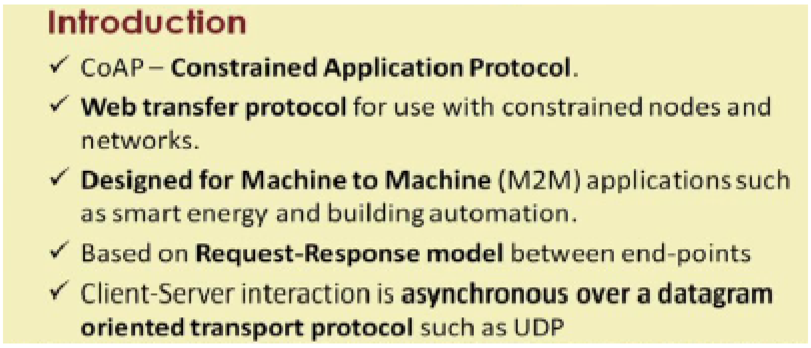
The major differences between the original MQTT and MQTT-SN include the following:

i) The CONNECT message types are broken into three messages in which two are optional and are tasked with the communication of the testament message and testament topic to the broker.
ii) The topic name in the PUBLISH messages are replaced by topic identifiers, which are only 2 bytes long. This reduces the traffic generated from the protocol and enables the protocol to operate over bandwidth-constrained networks.
iii) A separate mechanism is present for topic name registration with the broker in MQTT-SN. After a topic identifier is generated for the topic name, the identifier is informed to the publisher/subscribers. This mechanism also supports the reverse pathway.
iv) In special cases in MQTT-SN, pre-defined topic identifiers are present that need no registration mechanism. The mapping of topic names and identifiers are known in advance to the broker as well as the publishers/subscribers.
v) The presence of a special discovery process is used to link the publisher/subscriber to the operational broker's network address in the absence of a preconfigured broker address.
vi) The subscriptions to a topic, Will topic, and Will message are persistent in MQTTSN. The publishers/subscribers can modify their Will messages during a session.
vii) Sleeping publishers/subscribers are supported by a keep-alive procedure, which is offline, and which helps buffer the messages intended for them in the broker until they wake up. This feature of MQTT-SN is not present in regular MQTT.

Figure shows the two gateway types in MQTT-SN:
1) transparent gateway and
2) aggregating gateway.
The MQTT-SN converts/translates MQTT and MQTTSN traffic by acting as a bridge between these two network types. The transparent gateway creates as many connections to the MQTT broker as there are MQTT-SN nodes within its operational purview; whereas the aggregating gateway creates a single connection to the MQTT broker, irrespective of the number of MQTT-SN nodes under it.

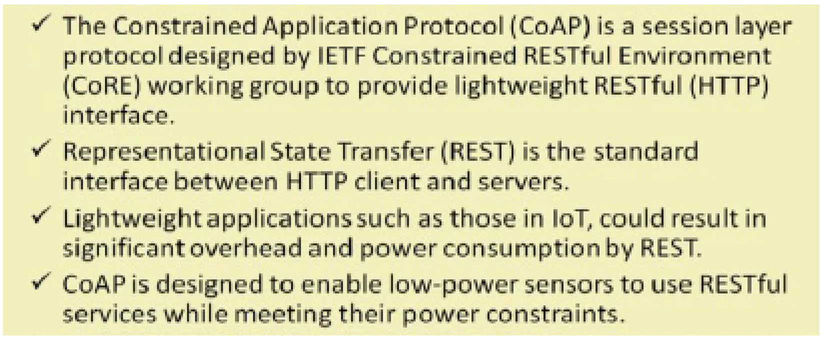


COAP:
The constrained application protocol, or CoAP as it is more popularly known, is designed for use as a web transfer protocol in constrained devices and networks, which are typically low power and lossy. The constrained devices typically have minimal RAM and an 8-bit processor at most. CoAP can efficiently work on such devices, even when these devices are connected to highly lossy networks with high packet loss, high error rates, and bandwidth in the range of kilobits.

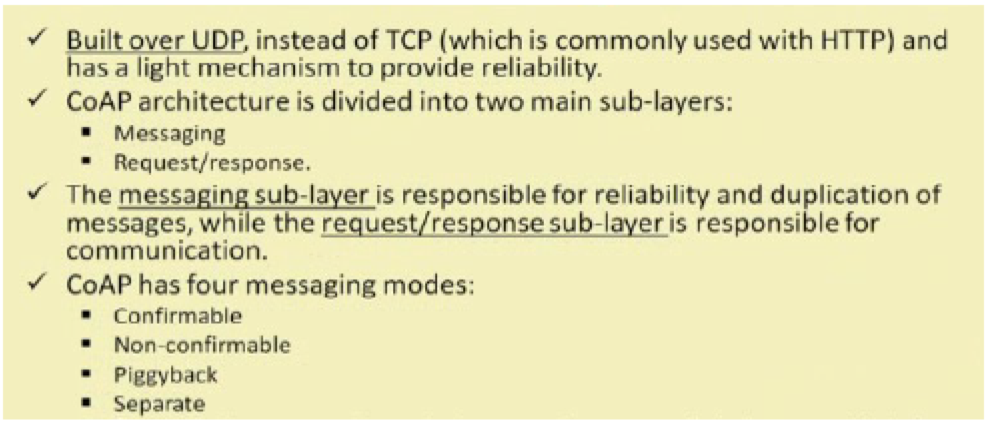


Introduction

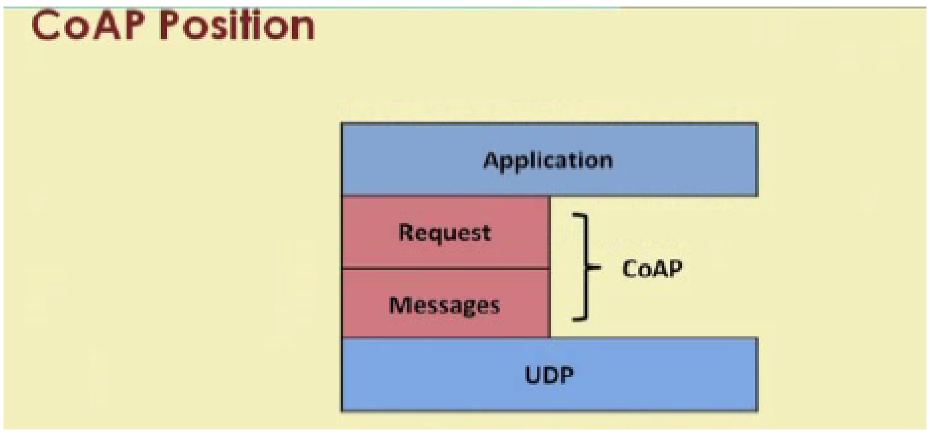
- ✓ CoAP – **Constrained Application Protocol**.
- ✓ **Web transfer protocol** for use with constrained nodes and networks.
- ✓ **Designed for Machine to Machine** (M2M) applications such as smart energy and building automation.
- ✓ Based on **Request-Response model** between end-points
- ✓ Client-Server interaction is **asynchronous over a datagram oriented transport protocol** such as UDP



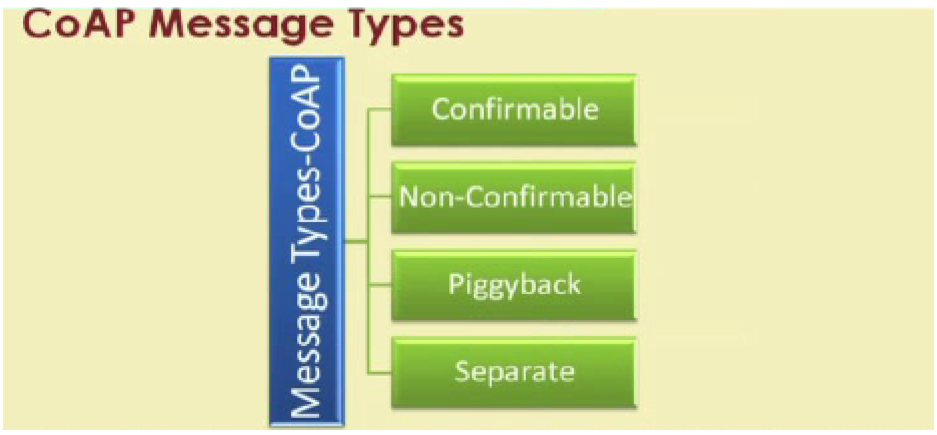
- ✓ The Constrained Application Protocol (CoAP) is a session layer protocol designed by IETF Constrained RESTful Environment (CoRE) working group to provide lightweight RESTful (HTTP) interface.
- ✓ Representational State Transfer (REST) is the standard interface between HTTP client and servers.
- ✓ Lightweight applications such as those in IoT, could result in significant overhead and power consumption by REST.
- ✓ CoAP is designed to enable low-power sensors to use RESTful services while meeting their power constraints.

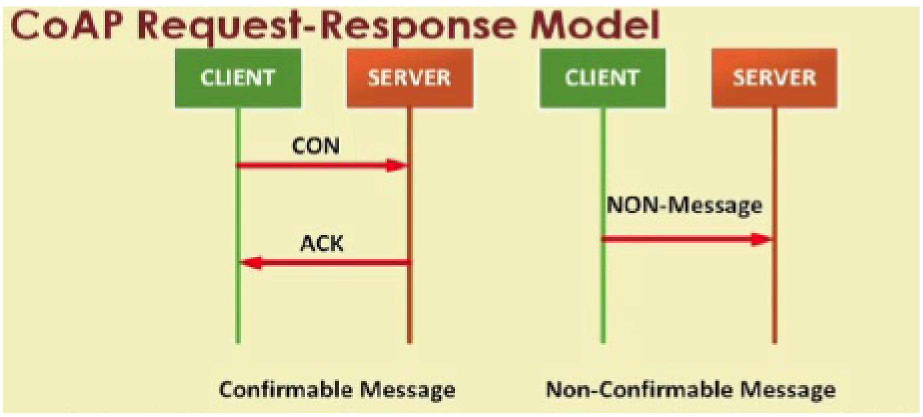


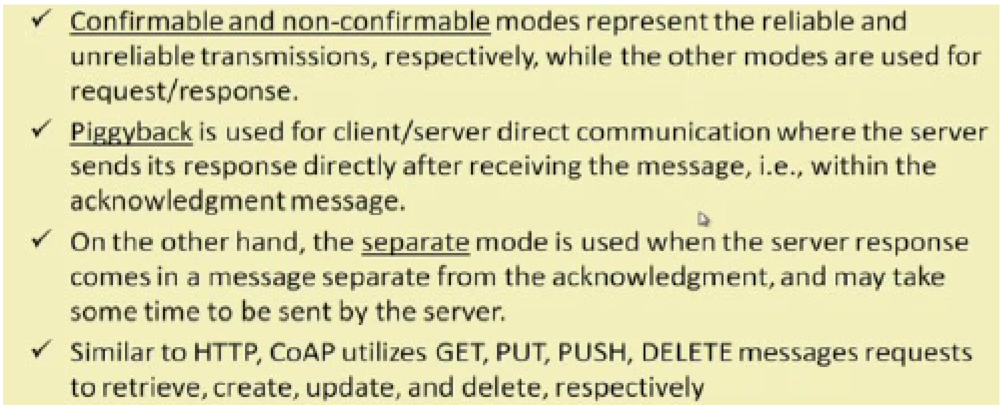
- ✓ Built over UDP, instead of TCP (which is commonly used with HTTP) and has a light mechanism to provide reliability.
- ✓ CoAP architecture is divided into two main sub-layers:
  - Messaging
  - Request/response.
- ✓ The messaging sub-layer is responsible for reliability and duplication of messages, while the request/response sub-layer is responsible for communication.
- ✓ CoAP has four messaging modes:
  - Confirmable
  - Non-confirmable
  - Piggyback
  - Separate



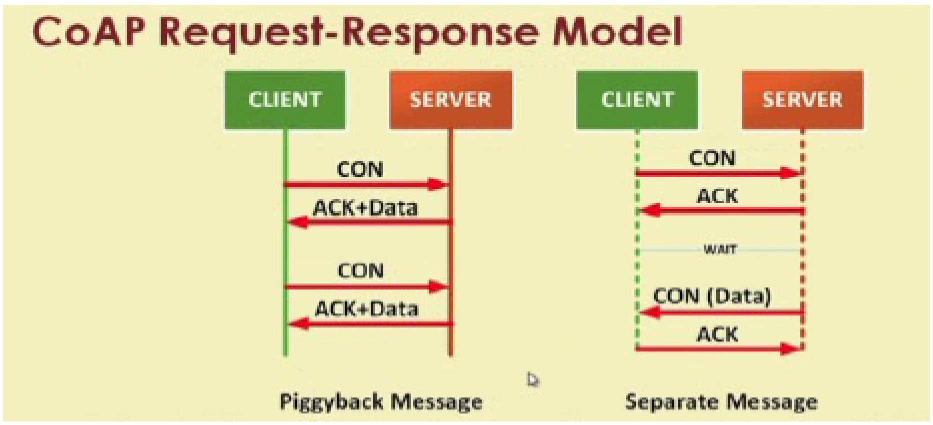
CoAP is a protocol of the session layer.



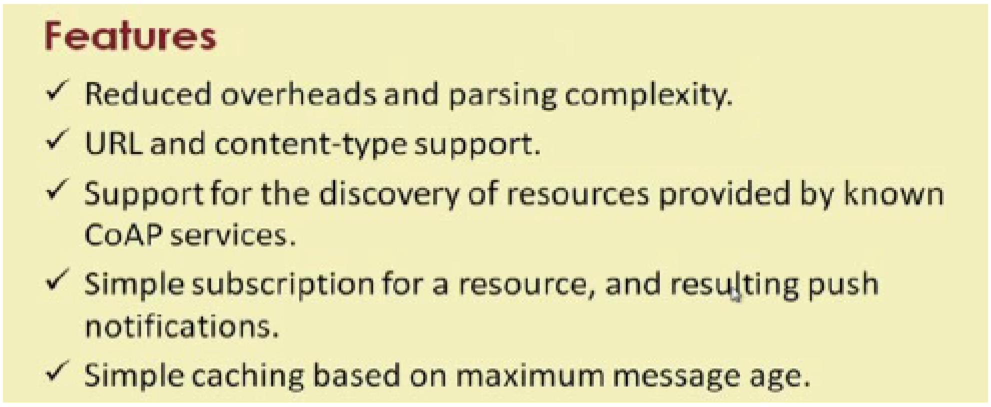




- ✓ Confirmable and non-confirmable modes represent the reliable and unreliable transmissions, respectively, while the other modes are used for request/response.
- ✓ Piggyback is used for client/server direct communication where the server sends its response directly after receiving the message, i.e., within the acknowledgment message.
- ✓ On the other hand, the separate mode is used when the server response comes in a message separate from the acknowledgment, and may take some time to be sent by the server.
- ✓ Similar to HTTP, CoAP utilizes GET, PUT, PUSH, DELETE messages requests to retrieve, create, update, and delete, respectively



CoAP Request-Response Model

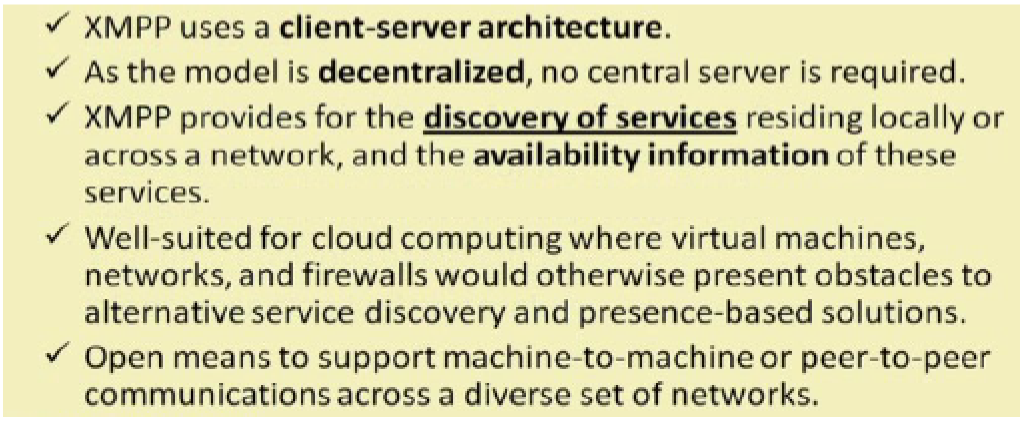


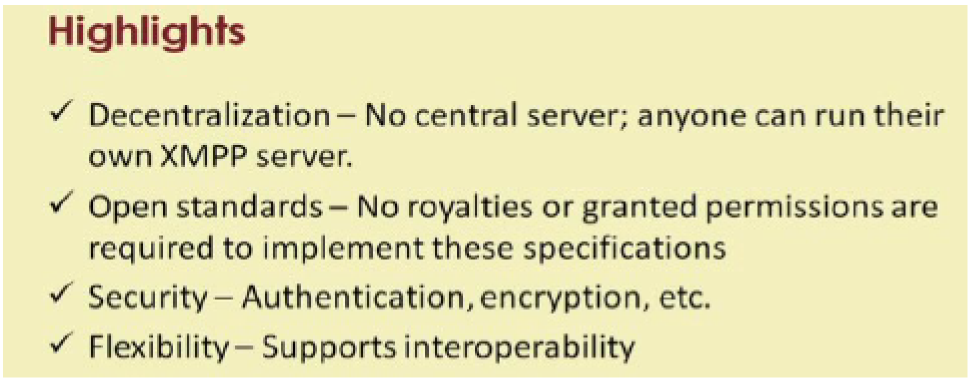
Features

- ✓ Reduced overheads and parsing complexity.
- ✓ URL and content-type support.
- ✓ Support for the discovery of resources provided by known CoAP services.
- ✓ Simple subscription for a resource, and resulting push notifications.
- ✓ Simple caching based on maximum message age.

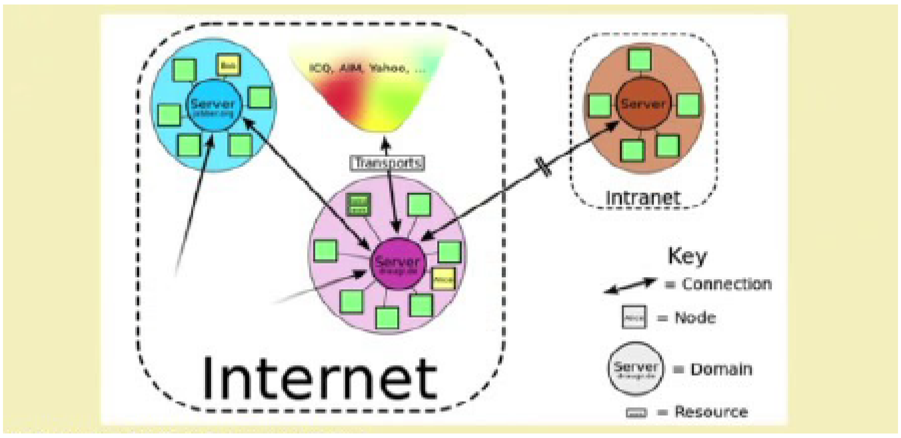
<u>XMPP :</u>

XMPP is **Extensible Messaging and Presence Protocol.** It is a message oriented middleware that is based on XML, whereas XML is particularly used for unstructured data. XMPP is useful for real time exchange of structured data and it is an open standard protocol. XMPP uses a client server architecture, it uses a decentralized model meaning that there is no server that is involved in the message transfer and it provides facilities for discovery of messages which are residing locally or globally across the network and the availability information of these services.

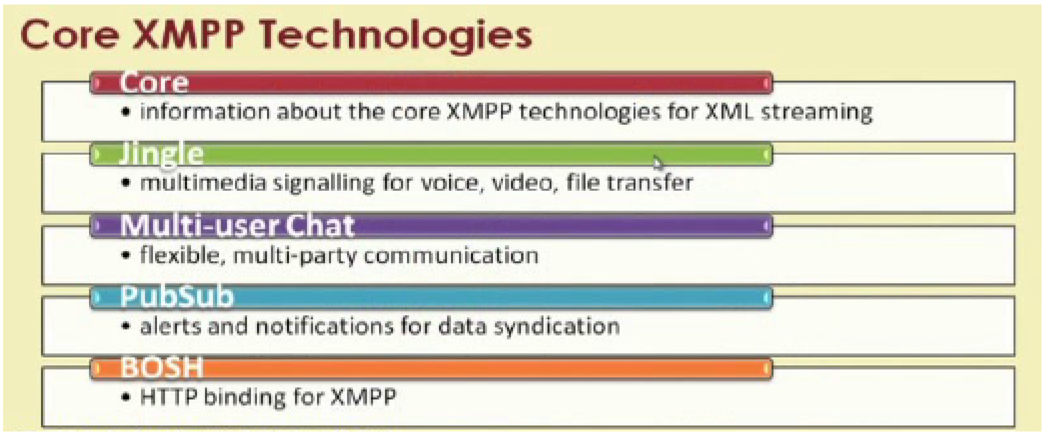

- ✓ XMPP uses a **client-server architecture**.
- ✓ As the model is **decentralized**, no central server is required.
- ✓ XMPP provides for the **discovery of services** residing locally or across a network, and the **availability information** of these services.
- ✓ Well-suited for cloud computing where virtual machines, networks, and firewalls would otherwise present obstacles to alternative service discovery and presence-based solutions.
- ✓ Open means to support machine-to-machine or peer-to-peer communications across a diverse set of networks.


Highlights

- ✓ Decentralization – No central server; anyone can run their own XMPP server.
- ✓ Open standards – No royalties or granted permissions are required to implement these specifications
- ✓ Security – Authentication, encryption, etc.
- ✓ Flexibility – Supports interoperability

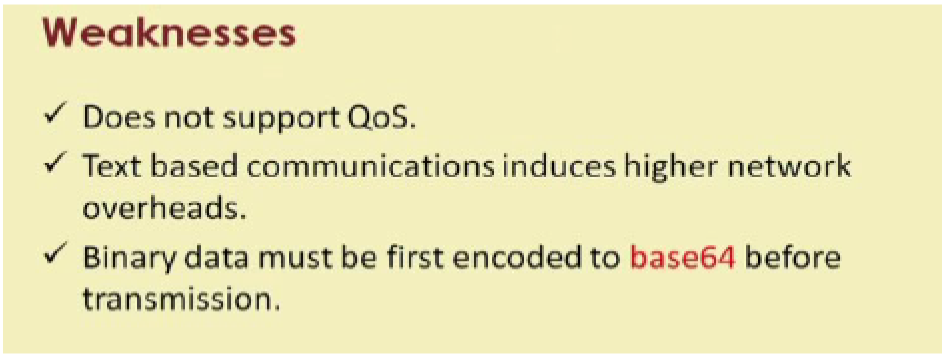


This particular figure what we see is with the help of XMPP, not only it is possible to communicate with other servers like in the case of the traditional internet, but also with other messaging platform such as ICQ, AIM, Yahoo and so on. Additionally it is also possible to communicate with other intranets.



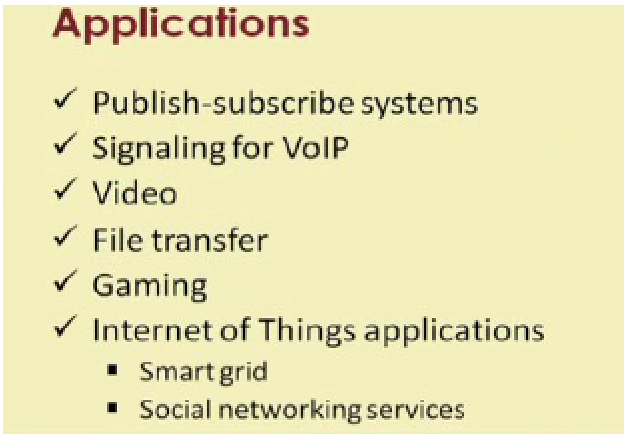
Core XMPP Technologies

- **Core**
  - information about the core XMPP technologies for XML streaming
- **Jingle**
  - multimedia signalling for voice, video, file transfer
- **Multi-user Chat**
  - flexible, multi-party communication
- **PubSub**
  - alerts and notifications for data syndication
- **BOSH**
  - HTTP binding for XMPP



Weaknesses

- ✓ Does not support QoS.
- ✓ Text based communications induces higher network overheads.
- ✓ Binary data must be first encoded to base64 before transmission.



Applications

- ✓ Publish-subscribe systems
- ✓ Signaling for VoIP
- ✓ Video
- ✓ File transfer
- ✓ Gaming
- ✓ Internet of Things applications
  - ▪ Smart grid
  - ▪ Social networking services