UNIT:3
Application Layer Protocols: CoAP and MQTT

When considering constrained networks and/or a large-scale deployment of constrained nodes, verbose web based and data model protocols, may be too heavy for IoT applications. To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained nodes and networks .Two of the most popular protocols are CoAP and MQTT. Figure2.19 highlights their position in a common IoT protocol stack.

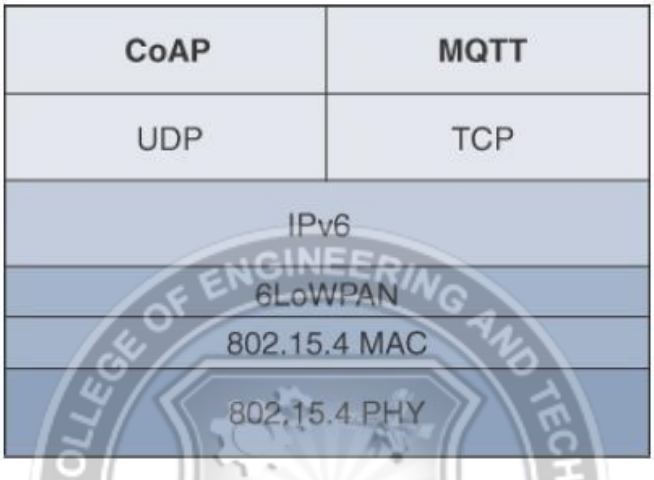

Figure2.19: Example of a High-Level IoTProtocol Stack for CoAPandMQTT

In Figure 2.19, CoAP and MQTT are naturally at the top of this sample IoT stack, based on anIEEE 802.15.4 mesh network. While there are a few exceptions, you will almost always findCoAP deployed over UDP and MQTT running over TCP. The following sections take a deeper lookat CoAPand MQTT.

CoAP

Constrained Application Protocol (CoAP) resulted from the IETF Constrained REST ful Environments (CoRE) working group's efforts to develop a generic framework for resource-oriented application star getting constrained nodes and networks. TheCoAP frame work defines simple and flexible ways to manipulate sensors and actuators for data or device management.

The CoAP messaging model is primarily designed to facilitate the exchange of messages overUDP between endpoints, including the secure transport protocol Datagram Transport LayerSecurity(DTLS).

From a formatting perspective, a CoAP message is composed of a short fixed-lengthHeader field (4 bytes), a variable-length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field.Figure2.20 details the CoAP message format, which deliver slow overhead while decreasing parsing complexity.

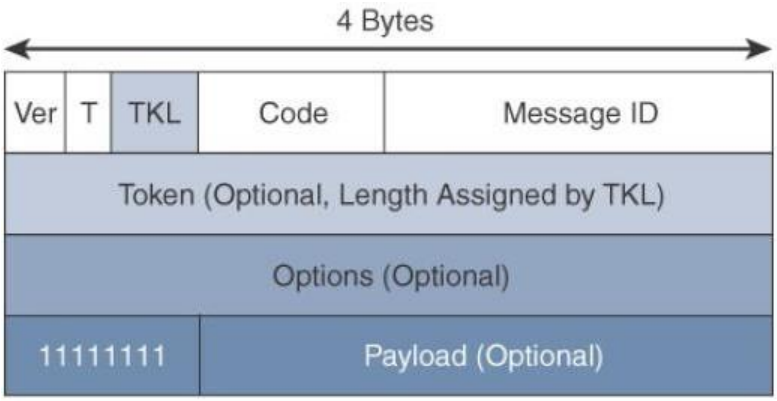

Figure2.20:CoAPMessageFormat

The CoAP message format is relatively simple and flexible. It allows CoAP to deliver lowoverhead, which is critical for constrained networks, while also being easy to parse and process for constrained devices.

Table2.4CoAPMessageFields

CoAP Message Field	Description
Ver (Version)	Identifies the CoAP version.
T (Type)	Defines one of the following four message types: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), or Reset (RST). CON and ACK are highlighted in more detail in Figure 6-9.
TKL (Token Length)	Specifies the size (0–8 Bytes) of the Token field.
Code	Indicates the request method for a request message and a response code for a response message. For example, in Figure 6-9, GET is the request method, and 2.05 is the response code. For a complete list of values for this field, refer to RFC 7252.
Message ID	Detects message duplication and used to match ACK and RST message types to Con and NON message types.
Token	With a length specified by TKL, correlates requests and responses.
Options	Specifies option number, length, and option value. Capabilities provided by the Options field include specifying the target resource of a request and proxy functions.
Payload	Carries the CoAP application data. This field is optional, but when it is present, a single byte of all 1s (0xFF) precedes the payload. The purpose of this byte is to delineate the end of the Options field and the beginning of Payload.

CoAP can run over IPv4 or IPv6. However, it is recommended that the message fit with in a single IP packet and UDP payload to avoid fragmentation. For IPv6, with the default MTU size being 1280 bytes and allowing for no fragmentation across nodes, the maximum CoAP message size could be upto1152bytes,including1024bytesforthepayload.

CoAP communications across an IoT infrastructure can take various paths. Connections can be between devices located on the same or different constrained networksor between devices and generic Internet or cloudservers, all operating overIP.Proxymechanisms are also defined, and RFC 7252 details a basic HTTP mapping for CoAP. As both HTTP and CoAP are IP-based protocols ,theproxy function can be located practically anywhere in the network, not necessarily at the border between constrained and non-constrained networks.

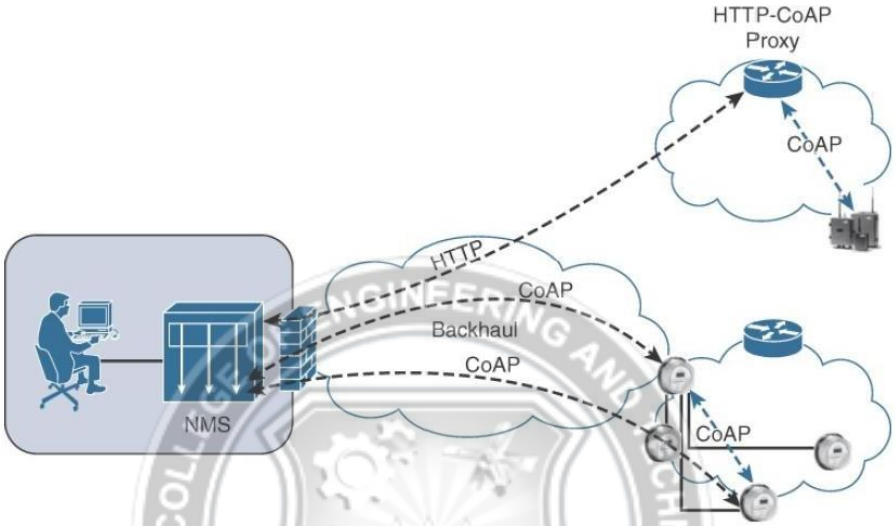

Figure2.21:CoAPCommunicationsinIoTInfrastructures

Just like HTTP, CoAP is based on the REST architecture, but with a “thing” acting asboth the client and the server. Through the exchange of asynchronous messages, a clientrequests an action via a method code on a server resource. A uniform resource identifier(URI) localized on the server identifies this resource. The server responds with a responsecode that may include a resource representation. The CoAP request/response semantics include the methods GET, POST,PUT, andDELETE.

Message QueuingTelemetryTransport(MQTT)

Attheendofthe1990s,engineers fro IBM andArcom(acquiredin2006byEurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries. Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards(OASIS).

The selection of a client/server and publish/subscribe framework based on the TCP/IP architecture, as showninFigure2.22

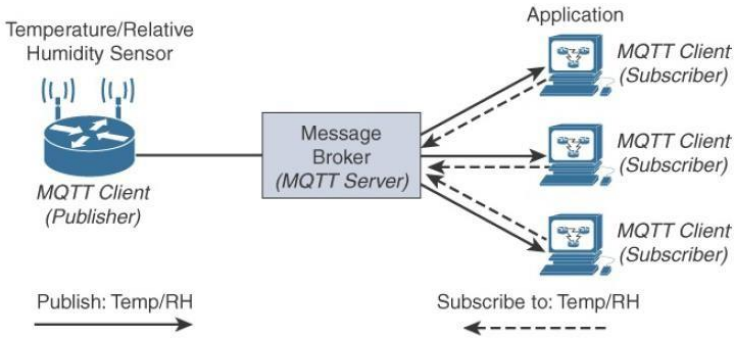

Figure2.22:MQTTPublish/SubscribeFramework

An MQTT client can act as a publisher to send data (or resource information) to anMQTT server acting as an MQTT message broker. In the example illustrated in Figure 2.22,the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensorthat publishes its Temp/RH data. The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers. Italso handles the subscription and unsubscription process and pushes the application data toMQTTclients acting as subscribers.

The application on the right side of Figure 2.22 is an MQTT client that is a subscriber to the Temp/RH data being generated by the publisher or sensor on the left. This model ,where subscribers express a desire to receive information from publishers, is well known. A great example is the collaboration and social networking application Twitter.

With MQTT, clients can subscribe to all data (using a wildcard character) or specificdata from the information tree of a publisher. In addition, the presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers. In fact, publishers and subscribers do not even know (or need to know) about each other. A benefit of having this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures. This also means that publishers and subscribers do not have to be online at the same time. MQTT control packets runovera TCP transport usingport1883.TCP ensures an ordered, lossless tream of bytes between the MQTT client and the MQTT server. Optionally, MQTT can be secured usingTLSonport8883,andWebSocket (definedinRFC6455)can also be used.

MQTT is a lightweight protocol because each control packet consists of a 2-byte fixedheader with optional variable header fields and optional payload. You should note that acontrol packet can contain a payload up to 256 MB. Figure 2.23 provides an overview of the MQTT messageformat.

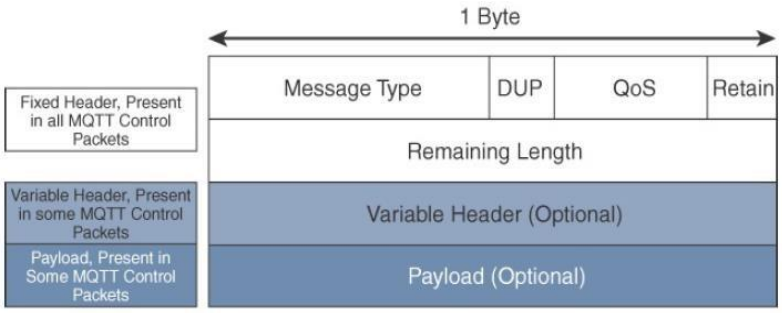

Figure2.23*MQTTMessageFormat*

Compared to the CoAP message format, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP. The first MQTT field in the header is Message Type, which identifies the kind of MQTT packet within a message. Fourteen different types of contro lpackets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. Note that values 0 and 15 are reserved. MQTT message types aresummarizedinTable2.5.

Table2.5MQTTMessageType

Message Type	Value	Flow	Description
CONNECT	1	Client to server	Request to connect
CONNACK	2	Server to client	Connect acknowledgement
PUBLISH	3	Client to server Server to client	Publish message
PUBACK	4	Client to server Server to client	Publish acknowledgement
PUBREC	5	Client to server Server to client	Publish received
PUBREL	6	Client to server Server to client	Publish release
PUBCOMP	7	Client to server Server to client	Publish complete
SUBSCRIBE	8	Client to server	Subscribe request
SUBACK	9	Server to client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to server	Unsubscribe request
UNSUBACK	11	Server to client	Unsubscribe acknowledgement
PINGREQ	12	Client to server	Ping request
PINGRESP	13	Server to client	Ping response
DISCONNECT	14	Client to server	Client disconnecting

The next field in the MQTT header is DUP (Duplication Flag). This flag, when set,allows the client to notate that the packet has been sent previously,but an acknowledgement was not received. The QoS header field allows for the selection of three different QoS levels. The next field is the Retain flag. Only found in a PUBLISH message, the Retain flag notifies the server to hold onto the message data. This allows new subscribers to instantly receive the last known value without having to wait for the next update from the publisher.

The last mandatory field in the MQTT message header is RemainingLength. This field specifies the number of bytes in theMQTTpacket following thisfield.

Subscription store sources generate SUBSCRIBE/SUBACK controlpackets, while unsubscription is performed through the exchange of UNSUBSCRIBE/UNSUBACK controlpackets.GracefulterminationofaconnectionisdonethroughaDISCONNECTcontrolpacket, which also offers the capability for a client to reconnect by re-sending its client ID to resume the operations.

Comparison of CoAP and MQTT

Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support