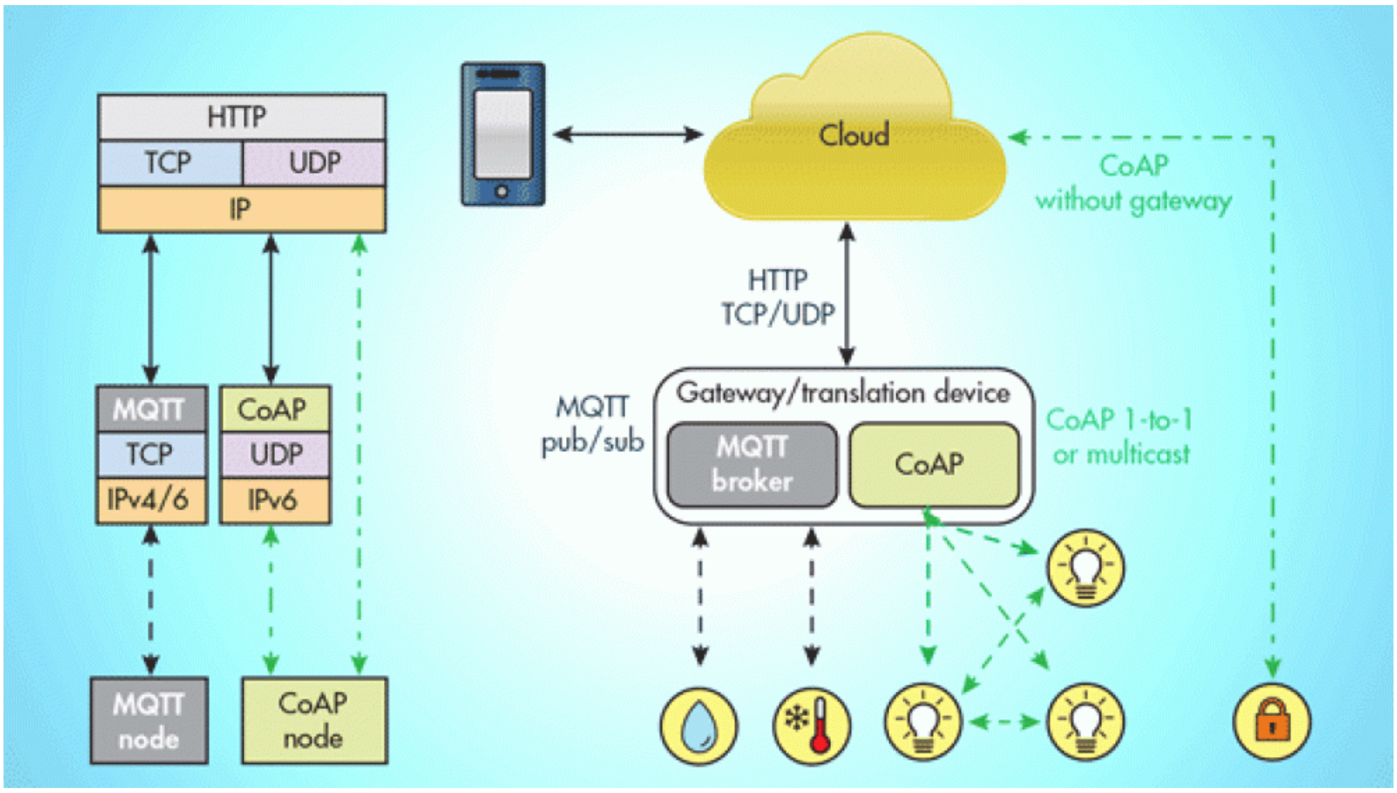
ElectronicDesign

ELECTRONIC DESIGN



TECHNOLOGIES > IOT

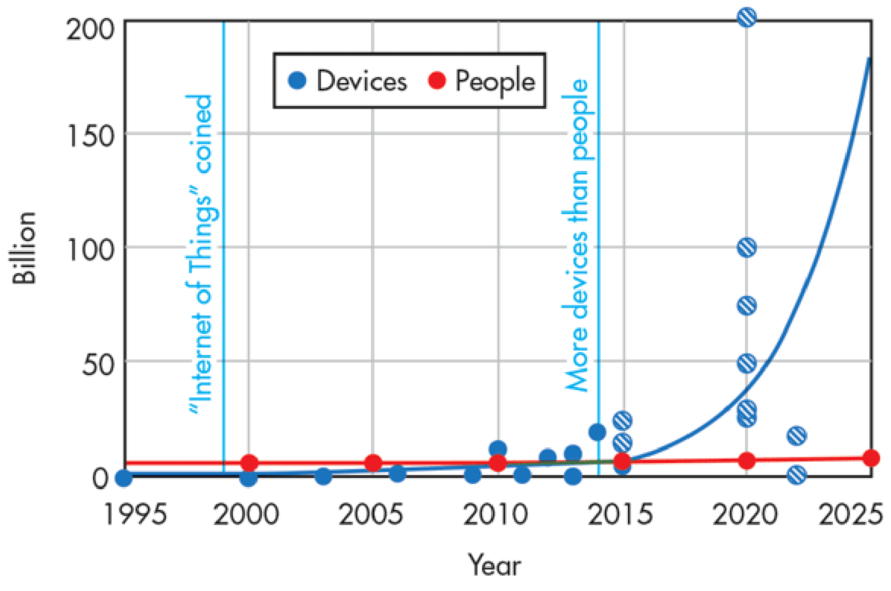
MQTT and CoAP: Underlying Protocols for the IoT

With HTTP's protocol overhead, how will billions of low-power, low-cost IoT devices communicate on the Internet?

James Stansberry | Oct 07, 2015

Download this article in .PDF format

A fascinating **article** from Philip N. Howard at George Washington University asserts that the number of connected devices surpassed the number of people on the planet in 2014 *(Fig. 1)*. The author estimates that by 2020, the Internet of Things (IoT) will approach 50 billion connected devices. In other words, while humans will continue to connect their devices to the Web in greater numbers, a bigger explosion will come from "things" connecting to the Web that weren't connected before, or didn't exist, or now use their connection as a core feature.



1. According to a recent study by Philip N. Howard, there are now more connected devices than people on the planet.

The question is, how will these billions of connected things communicate between end nodes, the cloud, and service providers?

This article dives into that subject as it relates to a particular class of connected devices—very low cost, battery-powered, and operable for at least seven years without any manual intervention (i.e., to replace batteries).

In particular, we will examine two emerging messaging protocols to address the needs of these "lightweight" IoT nodes. The first protocol, Message Queuing Telemetry Transport (MQTT), is old by today's standards, since it dates back to 1999. And the second, Constrained Application Protocol (CoAP), is relatively new and gaining traction.

IoT Communication Protocol Requirements

One definition of the IoT states that it's the connection of devices to the Internet that were not previously connected. For example, a factory owner may connect digital lights. A triathlete may connect a battery-powered heart-rate monitor. A home- or building-automation provider may connect battery-powered wireless sensor nodes.

The important point here is that in all of these use cases, the "thing" must communicate through the Internet to be considered an "IoT" node.

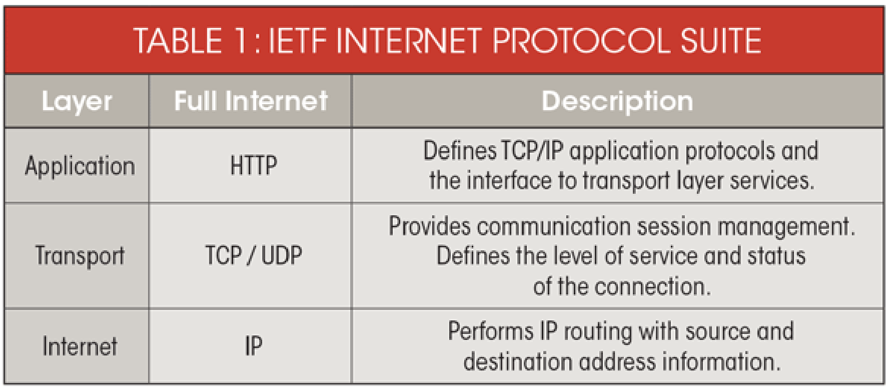


TABLE 1: IETF INTERNET PROTOCOL SUITE

Layer	Full Internet	Description
Application	HTTP	Defines TCP/IP application protocols and the interface to transport layer services.
Transport	TCP / UDP	Provides communication session management. Defines the level of service and status of the connection.
Internet	IP	Performs IP routing with source and destination address information.

The Internet Protocol Suite encompasses application, transport, and Internet layers.

Since it must use the Internet, the connected device must also adhere to the Internet Engineering Task Force's (IETF) Internet Protocol Suite *(Table 1)*. However, the Internet has historically connected resource-rich devices with lots of power, memory, and connection options. As such, its protocols have been considered too heavy to apply wholesale to applications in the emerging IoT.

Other aspects of the IoT also drive modifications to the IETF's work. In particular, networks of IoT end nodes will be lossy, and the devices attached to them will be very low power, saddled with constrained resources and expected to live for years.

The requirements for both the network and its end devices might look like Table 2. This new model needs new, lighter-weight protocols that don't require large amount of resources. MQTT and CoAP address these needs through small message sizes, message management, and lightweight message overhead. Table 2 summarizes the key requirements.

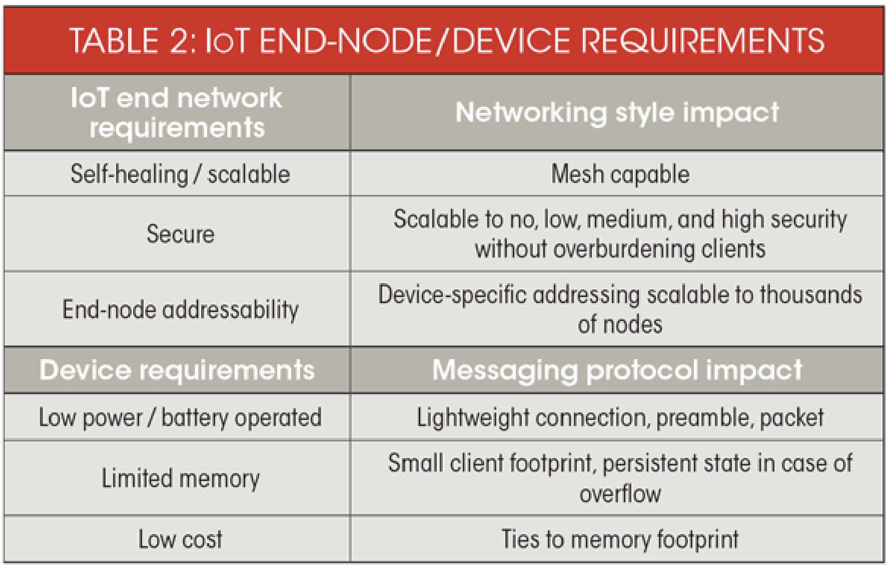


TABLE 2: IoT END-NODE/DEVICE REQUIREMENTS

IoT end network requirements	Networking style impact
Self-healing / scalable	Mesh capable
Secure	Scalable to no, low, medium, and high security without overburdening clients
End-node addressability	Device-specific addressing scalable to thousands of nodes
Device requirements	**Messaging protocol impact**
Low power / battery operated	Lightweight connection, preamble, packet
Limited memory	Small client footprint, persistent state in case of overflow
Low cost	Ties to memory footprint

Low-cost, power-constrained devices and associated networks have multiple requirements.

MQTT and CoAP support communication from Internet-based resource-rich devices to IoT-based resource-constrained devices. Both CoAP and MQTT implement a lightweight application layer, leaving much of the error correction to message retries, simple reliability strategies, or reliance on more resource-rich devices for post-processing of raw end-node data *(Fig. 2)*.

MQTT Overview

IBM invented MQTT for satellite communications with oil-field equipment . MQTT had reliability and low power at its core, and thus made good sense to be applied to IoT networks.

The MQTT standard has since been adopted by the OASIS open-standards society and released as version 3.1.1 . It's also supported within the Eclipse community, as well as by many commercial companies offering open-source stacks and consulting.

MQTT uses a "publish/subscribe" model and requires a central MQTT broker to manage and route messages among an MQTT network's nodes. Eclipse describes MQTT as "a many-to-many communication protocol for passing messages between multiple clients through a central broker." It uses TCP for its transport layer, which is characterized as "reliable, ordered and error-checked."

MQTT Strengths

Publish/subscribe model

MQTT's "pub/sub" model scales well and can be power efficient. Brokers and nodes publish information and others subscribe according to the message content, type, or subject. (These are MQTT standard terms.) Generally, the broker subscribes to all messages and then manages information flow to its nodes. The publish/subscribe model offers several specific benefits noted below.

Space decoupling

While the node and the broker need to have each other's IP address, nodes can publish information and subscribe to other nodes' published information without any knowledge of each other, since everything goes through the central broker. This reduces overhead that can accompany TCP sessions and ports, and allows the end nodes to operate independently of one another.

Time decoupling

A node can publish its information regardless of other nodes' states. Other nodes can then receive the published information from the broker when they are active. This allows nodes to remain in sleepy states even when other nodes are publishing messages directly relevant to them.

Synchronization decoupling

A node that's in the midst of one operation is not interrupted to receive a published message to which it's subscribed. The message is queued by the broker until the receiving node is finished with its existing operation. This saves operating current and reduces repeated operations by avoiding interruptions of ongoing operations or sleepy states.

Security

MQTT uses unencrypted TCP and is not "out-of-the-box" secure. However, because it uses TCP, it can—and should—use TLS/SSL Internet security. TLS is a very secure method for encrypting traffic, but is also resource-intensive for lightweight clients due to its required handshake and increased packet overhead. For networks where energy is a very high priority and security much less so, encrypting just the packet payload may suffice.

MQTT Quality of Service (QoS) levels

The term "QoS" means other things outside of MQTT. In MQTT, "QoS" levels 0, 1, and 2 describe increasing levels of guaranteed message delivery.

• MQTT QoS Level 0 (at most once): Commonly known as "Fire and forget," this is a single transmit burst with no guarantee of message arrival. It might be used for highly repetitive message types or non-mission critical messages.

• MQTT QoS Level 1 (at least once): This level attempts to guarantee a message is received at least once by the intended recipient. Once a published messaged is received and understood by the intended recipient, it acknowledges the message with an acknowledgement message (PUBACK) addressed to the publishing node. Until the PUBACK is received by the publisher, it stores the message and retransmits it periodically. This type of message may be useful for a non-critical node shutdown.

• MQTT QoS Level 2 (exactly once): This level attempts to guarantee the message is received and decoded by the intended recipient. It's the most secure and reliable MQTT level of QoS. The publisher sends a message announcing it has a QoS level 2 message. Its intended recipient gathers the announcement, decodes it, and indicates that it's ready to receive the message. The publisher relays its message. Once the recipient understands the message, it completes the transaction with an acknowledgement. This type of message may be useful for turning on or off lights or alarms in a home.

Last will and testament

MQTT provides a "last will and testament (LWT)" message that can be stored in the MQTT broker in case a node unexpectedly disconnects from the network. This LWT retains the node's state and purpose, including the types of commands it published and its subscriptions. If the node disappears, the broker notifies all subscribers of the node's LWT. And if the node returns, the broker notifies it of its prior state. This feature nicely accommodates lossy networks and scalability.

Flexible topic subscriptions

An MQTT node may subscribe to all messages within a given functionality. For example a "kitchen oven node" may subscribe to all messages for

```
kitchen/oven/+
```

with the "+" as a wildcard. This allows a minimal amount of code (i.e., memory and cost). Another example is if a node in the kitchen is interested in all temperature information regardless of the end node's functionality. In this case,

```
kitchen/+/temp
```

will collect any message in the kitchen from any node reporting “temp.” There are other equally useful MQTT wildcards for reducing code footprint and, therefore, memory size and cost.

Issues with MQTT

Central broker

The use of a central broker can be a drawback for distributed IoT systems. For example, a system may start small with a remote control and window shade, thus requiring no central broker. Then as the system grows, for example adding security sensors, light bulbs, or other window shades, the network naturally grows and expands, and may have need of a central broker. However, none of the individual nodes wants to take on the cost and responsibility as it requires resources, software, and complexity that are not core to the end-node function.

In systems that already have a central broker, it can become a single point of failure for the complete network. For example, if the broker is a powered node without a battery backup, then battery-powered nodes may continue operating during an electrical outage while the broker is offline, thus rendering the network inoperable.

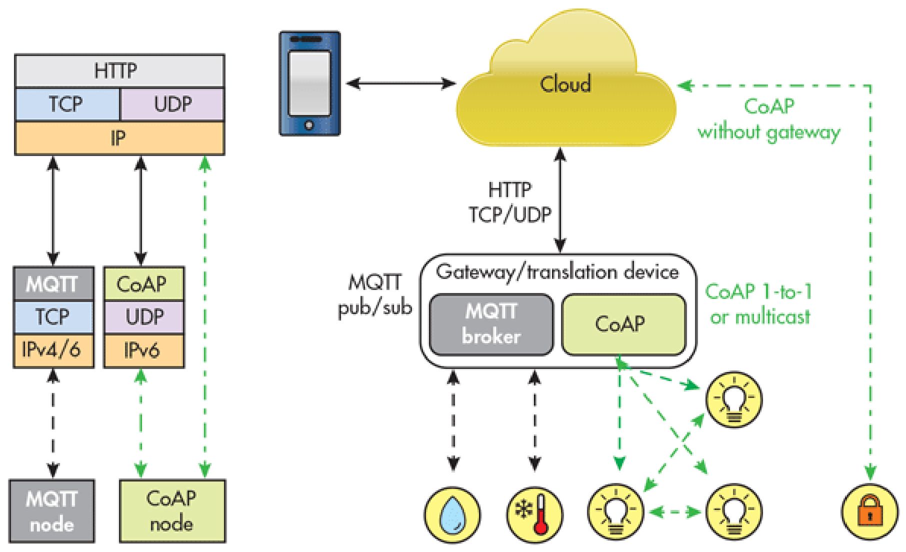
TCP

TCP was originally designed for devices with more memory and processing resources than may be available in a lightweight, IoT-style network. For example, the TCP protocol requires that connections be established in a multi-step handshake process before any messages are exchanged. This drives up wake-up and communication times, and reduces battery life over the long run.

Also, in TCP, it’s ideal for two communicating nodes to hold their TCP sockets open for each other continuously with a persistent session, which again may be difficult with energy- and resource-constrained devices.

Wake-up time

Again, using TCP without session persistence can require incremental transmit time for connection establishment. For nodes with periodic, repetitive traffic, this can lead to lower operating life.



2. MQTT and CoAP support communication to the cloud and smartphones.

CoAP Overview

With the growing importance of the IoT, the IETF took on lightweight messaging and defined the CoAP . As defined by the IETF, CoAP is for “use with constrained nodes and constrained (e.g., low-power, lossy) networks.” The Eclipse community also supports CoAP as an open standard, and like MQTT, CoAP is commercially supported and growing rapidly among IoT providers.

CoAP is a client/server protocol and provides a one-to-one “request/report” interaction model with accommodations for multi-cast, although multi-cast is still in the early stages of IETF standardization. Unlike MQTT, which has been adapted to IoT needs from a decades-old protocol, the IETF specified CoAP from the outset to support IoT with lightweight messaging for constrained devices operating in a constrained environment. CoAP is designed to interoperate with HTTP and the RESTful Web through simple proxies, making it natively compatible to the Internet.

Strengths of CoAP

Native UDP

CoAP runs over UDP , which is inherently and intentionally less reliable than TCP, depending on repetitive messaging for reliability instead of consistent connections. For example, a temperature sensor may send an update every few seconds, even though nothing has changed from one transmission to the next. If a receiving node misses one update, the next will arrive in a few seconds and will likely be not much different than the first.

UDP’s connectionless datagrams also enable faster wake-up and transmit cycles as well as smaller packets with less overhead. This allows devices to remain in a sleepy state for longer periods of time, conserving battery power.

Multi-cast support

A CoAP network is inherently one-to-one; however, it supports one-to-many or many-to-many multi-cast requirements. This is inherent in CoAP because it's built on top of IPv6, which enables multicast addressing for devices in addition to their normal IPv6 addresses. Note that multicast message delivery to sleeping devices is unreliable or can impact the device's battery life if it must wake regularly to receive these messages.

Security

CoAP uses DTLS on top of its UDP transport protocol. Like TCP, UDP is unencrypted, but can be—and should be—augmented with DTLS.

Resource/service discovery

CoAP uses URI to provide a standard presentation and interaction expectations for network nodes. This allows a degree of autonomy in the message packets, since the target node's capabilities are partly understood by its URI details. In other words, a battery-powered sensor node may have one type of URI while a line-powered flow-control actuator may have another. Nodes communicating to the battery-powered sensor node might be programmed to expect longer response times, more repetitive information, and limited message types. Nodes communicating to the line-powered flow-control actuator might be programmed to expect rich, detailed messages at a very rapid pace.

Asynchronous communication

Within the CoAP protocol, most messages are sent and received using the request/report model; however, other modes of operation allow nodes to be somewhat decoupled. For example, CoAP has a simplified "observe" mechanism similar to MQTT's pub/sub that enables nodes to observe others without actively engaging them.

As an example of the "observe" mode, node 1 can observe node 2 for specific transmission types. Then, any time node 2 publishes a relevant message, node 1 receives it when it awakens and queries another node. It's important to note that one of the network nodes must hold messages for observers. This is similar to MQTT's broker model, except that CoAP has no broker requirement, and therefore no expectation of being able to hold or queue messages for observers.

Draft additions to the standard may provide a similar CoAP function to MQTT's pub/sub model over the short-to-medium term. The leading candidate today is a draft proposal from Michael Koster , allowing CoAP networks to implement a pub/sub model like MQTT's mentioned above.

Issues with CoAP

Standard maturity

MQTT is currently a more mature and stable standard than CoAP. For many IoT developers, it's easier to get an MQTT network up and running very quickly than a similar network using CoAP. That said, CoAP has tremendous market momentum and is rapidly evolving to provide a standardized foundation, with important add-ons now in the ratification pipeline.

It’s likely that CoAP will reach a level of stability and maturity similar to MQTT in the very near term. But the standard is evolving for now, which may present some interoperability challenges.

Message reliability (QoS level)

CoAP’s “reliability” is MQTT’s QoS. It provides a very simple method of providing a “confirmable” message and a “non-confirmable” message. The confirmable message is acknowledged with an acknowledgement message (ACK) from the intended recipient. This confirms the message is received, but stops short of confirming that its contents were decoded correctly or at all. A non-confirmable message is “fire and forget.”

Summary

MQTT and CoAP are rapidly emerging as leading lightweight messaging protocols for the booming IoT market. Each protocol offers unique benefits, and each poses challenges and tradeoffs. Both protocols are being implemented for mesh-networking applications, in which lightweight end nodes are a necessary aspect of almost every network, and for gateway bridging logic to allow inter-standard communication.

References:

MQTT

Specification: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Excellent source for MQTT information: http://www.hivemq.com/mqtt-essentials-wrap-up/

CoAP

Specification: https://tools.ietf.org/html/rfc7252

Excellent source for CoAP information: http://coap.technology/

MQTT-SN

Specification: http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn

General coverage of IoT messaging protocols

Excellent white paper on using MQTT, CoAP, and other messaging protocols:
http://www.prismtech.com/sites/default/files/documents/MessagingComparsionNov2013USROW_vfinal.pdf

Source URL: http://www.electronicdesign.com/iot/mqtt-and-coap-underlying-protocols-iot