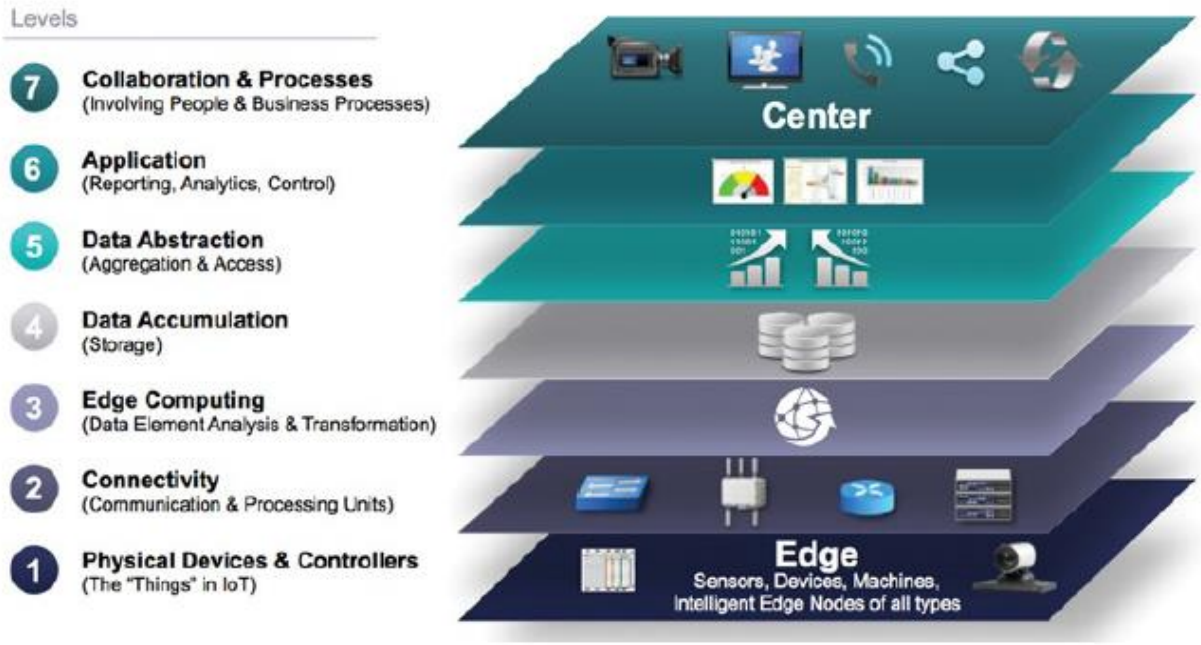
- **Encryption**
  If you are using wifi, use Wireless Protected Access 2 (WPA2) for wireless network encryption. You may also adopt a Private Pre-Shared Key (PPSK) approach. To ensure privacy and data integrity for communication between applications, be sure to adopt TLS or Datagram Transport-Layer Security (DTLS), which is based on TLS, but adapted for unreliable connections that run over UDP. TLS encrypts application data and ensures its integrity.
- **Port protection**
  Port protection ensures that only the ports required for communication with the gateway or upstream applications or services remain open to external connections. All other ports should be disabled or protected by firewalls. Device ports might be exposed when exploiting Universal Plug and Play (UPnP) vulnerabilities. Thus, UPnP should be disabled on the router.

The IoT World Forum (IoTWF) Standardized Architecture

In 2014 the IoTWF architectural committee (led by Cisco, IBM, Rockwell Automation, and others)published a seven-layer IoT architectural reference model. While various IoT reference models exist, theone put forth by the IoT World Forum offers a clean, simplified perspective on IoT and includes edgecomputing, data storage, and access. It provides a succinct way of visualizing IoT from a technicalperspective. Each of the seven layers is broken down into specific functions, and security encompassesthe entire model. Figure belowdetails the IoT Reference Model published by the IoTWF.



As shown in Figure 2-2, the IoT Reference Model defines a set of levels with control flowing from thecenter (this could be either a cloud service or a dedicated data center), to the edge,

which includessensors, devices, machines, and other types of intelligent end nodes. In general, data travels up the stack,originating from the edge, and goes northbound to the center. Using this reference model, we are able toachieve the following:

- Decompose the IoT problem into smaller parts
- Identify different technologies at each layer and how they relate to one another
- Define a system in which different parts can be provided by different vendors
- Have a process of defining interfaces that leads to interoperability
- Define a tiered security model that is enforced at the transition points between levels

The following sections look more closely at each of the seven layers of the IoT Reference Model.

Layer 1: Physical Devices and Controllers Layer

The first layer of the IoT Reference Model is the physical devices and controllers layer. This layer is hometo the "things" in the Internet of Things, including the various endpoint devices and sensors that send andreceive information. The size of these "things" can range from almost microscopic sensors to giantmachines in a factory. Their primary function is generating data and being capable of being queriedand/or controlled over a network.

Layer 2: Connectivity Layer

In the second layer of the IoT Reference Model, the focus is on connectivity. The most important functionof this IoT layer is the reliable and timely transmission of data. More specifically, this includestransmissions between Layer 1 devices and the network and between the network and informationprocessing that occurs at Layer 3 (the edge computing layer).As you may notice, the connectivity layer encompasses all networking elements of IoT and doesn't reallydistinguish between the last-mile network (the network between the sensor/endpoint and the IoT gateway,discussed later in this chapter), gateway, and backhaul networks. Functions of the connectivity layer aredetailed in Figure 2-3.

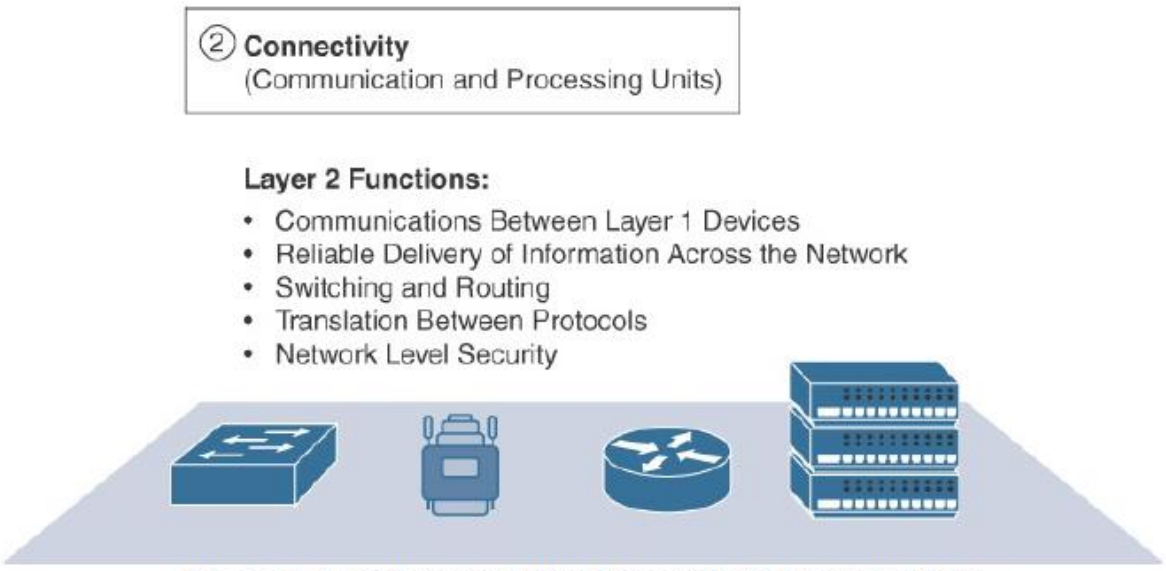


Figure 2-3 *IoT Reference Model Connectivity Layer Functions*

Layer 3: Edge Computing Layer

Edge computing is the role of Layer 3. Edge computing is often referred to as the "fog" layer and isdiscussed in the section "Fog Computing," later in this chapter. At this layer, the emphasis is on datareduction and converting network data flows into information that is ready for storage and processing byhigher layers. One of the basic principles of this reference model is that information processing is initiated

as early and as close to the edge of the network as possible. Figure 2-4 highlights the functions handledby Layer 3 of the IoT Reference Model.

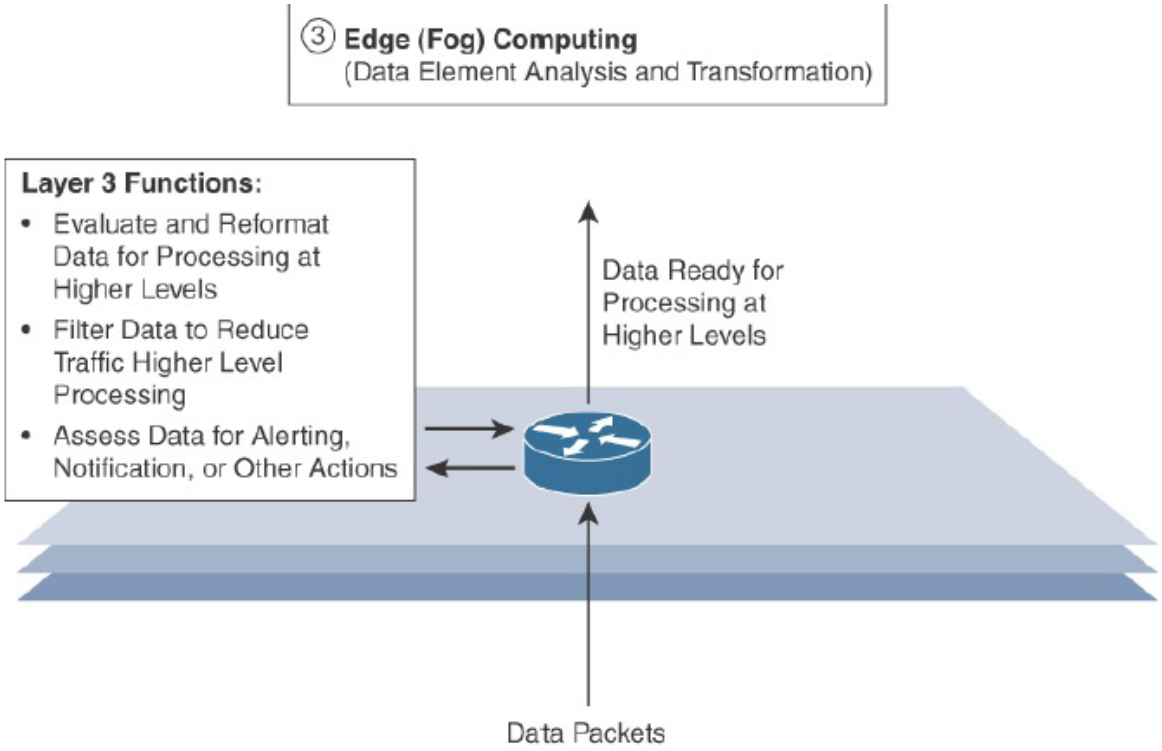


Figure 2-4 *IoT Reference Model Layer 3 Functions*

Another important function that occurs at Layer 3 is the evaluation of data to see if it can be filtered oraggregated before being sent to a higher layer. This also allows for data to be reformatted or decoded,making additional processing by other systems easier. Thus, a critical function is assessing the data to seeif predefined thresholds are crossed and any action or alerts need to be sent.

Upper Layers: Layers 4–7

The upper layers deal with handling and processing the IoT data generated by the bottom layer. For thesake of completeness, Layers 4–7 of the IoT Reference Model are summarized in Table 2-2.

IoT Reference Model Layer	Functions
Layer 4: Data accumulation layer	Captures data and stores it so it is usable by applications when necessary. Converts event-based data to query-based processing.
Layer 5: Data abstraction layer	Reconciles multiple data formats and ensures consistent semantics from various sources. Confirms that the data set is complete and consolidates data into one place or multiple data stores using virtualization.
Layer 6: Applications layer	Interprets data using software applications. Applications may monitor, control, and provide reports based on the analysis of the data.
Layer 7: Collaboration and processes layer	Consumes and shares the application information. Collaborating on and communicating IoT information often requires multiple steps, and it is what makes IoT useful. This layer can change business processes and delivers the benefits of IoT.

Table 2-2 *Summary of Layers 4–7 of the IoTWF Reference Model*

M2M Communication

Machine-to-machine communication, or M2M, is exactly as it sounds: two machines "communicating," or exchanging data, without human interfacing or interaction. This includes serial connection, powerline connection (PLC), or wireless communications in the industrial Internet of Things (IoT). Switching over to wireless has made M2M communication much easier and enabled more applications to be connected.

In general, when someone says M2M communication, they often are referring to cellular communication for embedded devices. Examples of M2M communication in this case would be vending machines sending out inventory information or ATM machines getting authorization to despense cash.

As businesses have realized the value of M2M, it has taken on a new name: the Internet of Things (IoT). IoT and M2M have similar promises: to fundamentally change the way the world operates. Just like IoT, M2M allows virtually any sensor to communicate, which opens up the possibility of systems monitoring themselves and automatically responding to changes in the environment, with a much reduced need for human involvement. M2M and IoT are almost synonymous—the exception is IoT (the newer term) typically refers to wireless communications, whereas M2M can refer to any two machines—wired or wireless—communicating with one another.

IEEE 802.15.4

IEEE 802.15.4 is a wireless access technology for low-cost and low-data-rate devices that are powered or run on batteries. In addition to being low cost and offering a reasonable battery life, this access technology enables easy installation using a compact protocol stack while remaining both simple and flexible. Several network communication stacks, including deterministic ones, and profiles leverage this technology to address a wide range of IoT use cases in both the consumer and business markets.

IEEE 802.15.4 is commonly found in the following types of deployments:

- Home and building automation
- Automotive networks
- Industrial wireless sensor networks
- Interactive toys and remote controls

Criticisms of IEEE 802.15.4 often focus on its MAC reliability, unbounded latency, and susceptibility to interference and multipath fading.

The negatives around reliability and latency often have to do with the Collision Sense Multiple Access/Collision Avoidance (CSMA/CA) algorithm. CSMA/CA is an access method in which a device "listens" to make sure no other devices are transmitting before starting its own transmission. If another device is transmitting, a wait time (which is usually random) occurs before "listening" occurs again. Interference and multipath fading occur with IEEE 802.15.4 because it lacks a frequency-hopping technique. Later variants of 802.15.4 from the IEEE start to address these issues.

Standardization and Alliances:

IEEE 802.15.4 or IEEE 802.15 Task Group 4 defines low-data-rate PHY and MAC layer specifications for wireless personal area networks (WPAN). This standard has evolved over the years and is a well-known solution for low-complexity wireless devices with low data rates that need many months or even years of battery.

Since 2003, the IEEE has published several iterations of the IEEE 802.15.4 specification. Newer releases
typically supersede older ones, integrate addendums, and add features or clarifications to previous versions.

The IEEE 802.15.4 PHY and MAC layers are the foundations for several networking protocol stacks. These protocol stacks make use of 802.15.4 at the physical and link layer levels, but the upper layers are different. These protocol stacks are promoted separately through various organizations and often commercialized. Some of the most well-known protocol stacks based on 802.15.4 are highlighted in Table 2-1.

S.No	Protocol	Description
1.	ZigBee	Promoted through the ZigBee alliance, ZigBee defines upper-layer components (network through application) as well as application profiles. Common profiles include building automation, home automation, and healthcare. ZigBee also defines device object functions such as device role, device discovery, network join and security
2.	6LoWPAN	6LoWPAN is an IPv6 adaptation layer defined by the IETF 6LoWPAN working group that describes how to transport IPv6 packets over IEEE 802.15.4 layers. RFCs document header compression and IPv6 enhancement to cope with the specific details of IEEE 802.15.4
3.	ZigBee IP	An evolution of the ZigBee protocol stack, ZigBee IP adopts the 6LoWPAN adaptation layer , IPv6 network layer, RPL routing protocol. In addition, it offers improvement in IP security.
4.	ISA100.11a	This is developed by the International Society of Automation (ISA) as "Wireless Systems for Industrial automation: Process Control and Related Applications".It is based on IEEE 802.15.4-2006. The network and transport layers are based on IETF 6LoWPAN, IPv6, and UDP standards
5.	Wireless HART	Wireless HART promoted by the HART Communication Foundation, is a protocol stack that offers a time-synchronized, self-organizing, and self healing mesh architecture, leveraging IEEE 802.15.4-2006 over the 2.4GHz frequency band.
6.	Thread	Constructed on top of IETF 6LoWPAN /IPv6, Thread is a protocol stack for a secure and reliable mesh network to connect and control products in the home.

ZigBee

ZigBee is one of the most well-known protocols listed in Table 4-2. In addition, ZigBee has continued to evolve over time as evidenced by the release of Zigbee IP and is representative of how IEEE 802.15.4 can be leveraged at the PHY and MAC layers, independent of the protocol layers above. The Zigbee Alliance is an industry group formed to certify interoperability between vendors and it is committed to driving and evolving ZigBee as an IoT solution for interconnecting smart objects. ZigBee solutions are aimed at smart objects and sensors that have low bandwidth and low power needs.

Furthermore, products that are ZigBee compliant and certified by the ZigBee Alliance should interoperate even though different vendors may manufacture them. The ZigBee specification has undergone several revisions.

The main areas where ZigBee is the most well-known include automation for commercial, retail, and home applications and smart energy. In the industrial and commercial automation space, ZigBee-based devices can handle various functions, from measuring temperature and humidity to tracking assets. For home automation, ZigBee can control lighting, thermostats, and security functions. ZigBee Smart Energy brings together a variety of interoperable products, such as smart meters, that can monitor and control the use and delivery of utilities, such as electricity and water. These ZigBee products are controlled by

- **Serial Communications:** Many legacy devices in certain industries, such as manufacturing and utilities, communicate through serial lines. Data is transferred using either proprietary or standards based protocols, such as DNP3, Modbus, or IEC 60870-5-101. In the past, communicating this serial data over any sort of distance could be handled by an analog modem connection. However, as service provider support for analog line services has declined, the solution for communicating with these legacy devices has been to use local connections. To make this work, you connect the serial port of the legacy device to a nearby serial port on a piece of communications equipment, typically a router. This local router then forwards the serial traffic over IP to the central server for processing. Encapsulation of serial protocols over IP leverages mechanisms such as raw socket TCP or UDP. While raw socket sessions can run over both IPv4 and IPv6, current implementations are mostly available for IPv4 only.
- **IPv6 Adaptation Layer:** IPv6-only adaptation layers for some physical and data link layers for recently standardized IoT protocols support only IPv6. While the most common physical and data link layers (Ethernet, Wi-Fi, and so on) stipulate adaptation layers for both versions, newer technologies, such as IEEE 802.15.4 (Wireless Personal Area Network), IEEE 1901.2, and ITU G.9903 (Narrowband Power Line Communications) only have an IPv6 adaptation layer specified. This means that any device implementing a technology that requires an IPv6 adaptation layer must communicate over an IPv6-only sub network. This is reinforced by the IETF routing protocol for LLNs, RPL, which is IPv6 only.

6LoWPAN

While the Internet Protocol is key for a successful Internet of Things, constrained nodes and constrained networks mandate optimization at various layers and on multiple protocols of the IP architecture. Some optimizations are already available from the market or under development by the IETF. Figure 2.12 highlights the TCP/IP layers where optimization is applied.

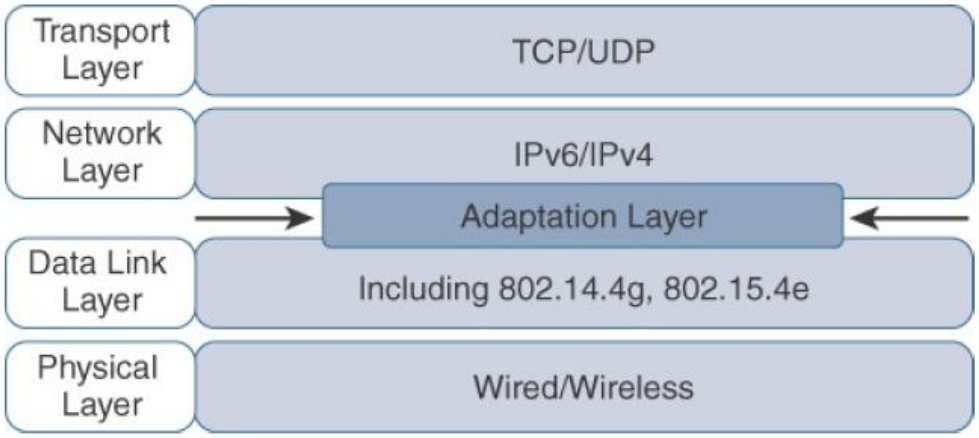


Figure 2.12: Optimizing IP for IoT Using an Adaptation Layer

In the IP architecture, the transport of IP packets over any given Layer 1 (PHY) and Layer 2 (MAC) protocol must be defined and documented. The model for packaging IP into lower-layer protocols is often referred to as an *adaptation layer*.

Unless the technology is proprietary, IP adaptation layers are typically defined by an IETF working group and released as a Request for Comments (RFC). An RFC is a publication from the IETF that officially documents Internet standards, specifications, protocols, procedures, and events. For example, RFC 864 describes how an IPv4 packet gets encapsulated over an Ethernet frame, and RFC 2464 describes how the same function is performed for an IPv6 packet.

IoT-related protocols follow a similar process. The main difference is that an adaptation layer designed for IoT may include some optimizations to deal with constrained nodes and networks. The main examples of adaptation layers optimized for constrained nodes or "things" are the ones under the 6LoWPAN working group and its successor, the 6Lo working group.

The initial focus of the 6LoWPAN working group was to optimize the transmission of IPv6 packets over constrained networks such as IEEE 802.15.4. Figure 2.13 shows an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well-known IP protocol stack for reference.

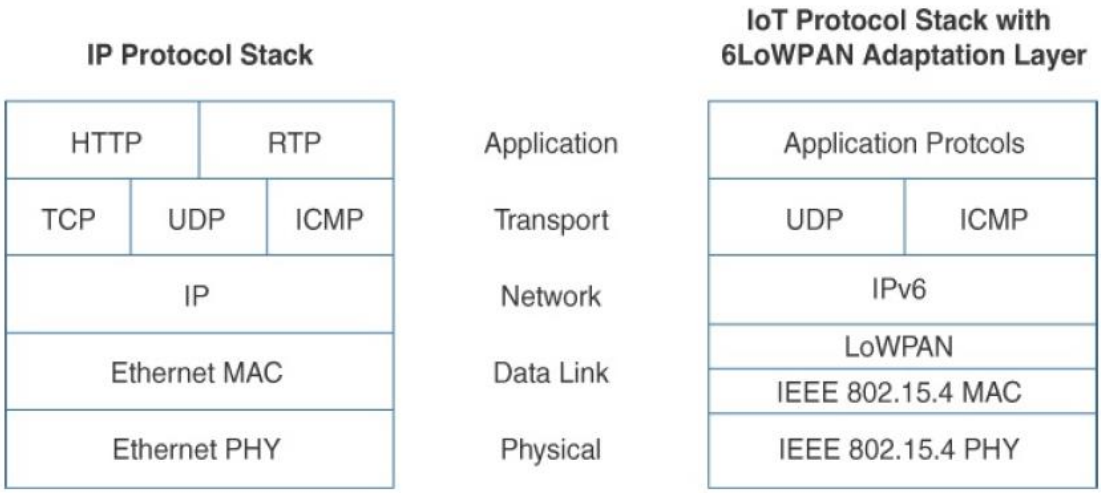


Figure 2.13: Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

The 6LoWPAN working group published several RFCs, but RFC 4994 is foundational because it defines frame headers for the capabilities of header compression, fragmentation, and mesh addressing. These headers can be stacked in the adaptation layer to keep these concepts separate while enforcing a structured method for expressing each capability. Depending on the implementation, all, none, or any combination of these capabilities and their corresponding headers can be enabled. Figure 2.14 shows some examples of typical 6LoWPAN header stacks.

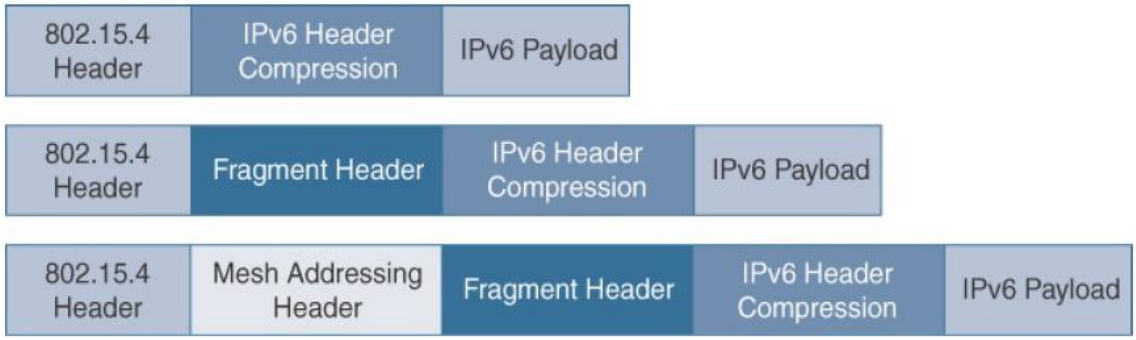


Figure 2.14 6LoWPAN Header Stack

Header Compression

IPv6 header compression for 6LoWPAN was defined initially in RFC 4944 and subsequently updated by RFC 6282. This capability shrinks the size of IPv6's 40-byte headers and User Datagram Protocol's (UDP's) 8-byte headers down as low as 6 bytes combined in some cases. Note that header compression for 6LoWPAN is only defined for an IPv6 header and not IPv4.

The 6LoWPAN protocol does not support IPv4, and, in fact, there is no standardized IPv4 adaptation layer for IEEE 802.15.4. 6LoWPAN header compression is stateless, and conceptually it is not too complicated. However, a number of factors affect the amount of compression, such as implementation of RFC 4944 versus RFC 6922, whether UDP is included, and various IPv6 addressing scenarios.

At a high level, 6LoWPAN works by taking advantage of shared information known by all nodes from their participation in the local network. In addition, it omits some standard header fields by assuming commonly used values. Figure 2.15 highlights an example that shows the amount of reduction that is possible with 6LoWPAN header compression.

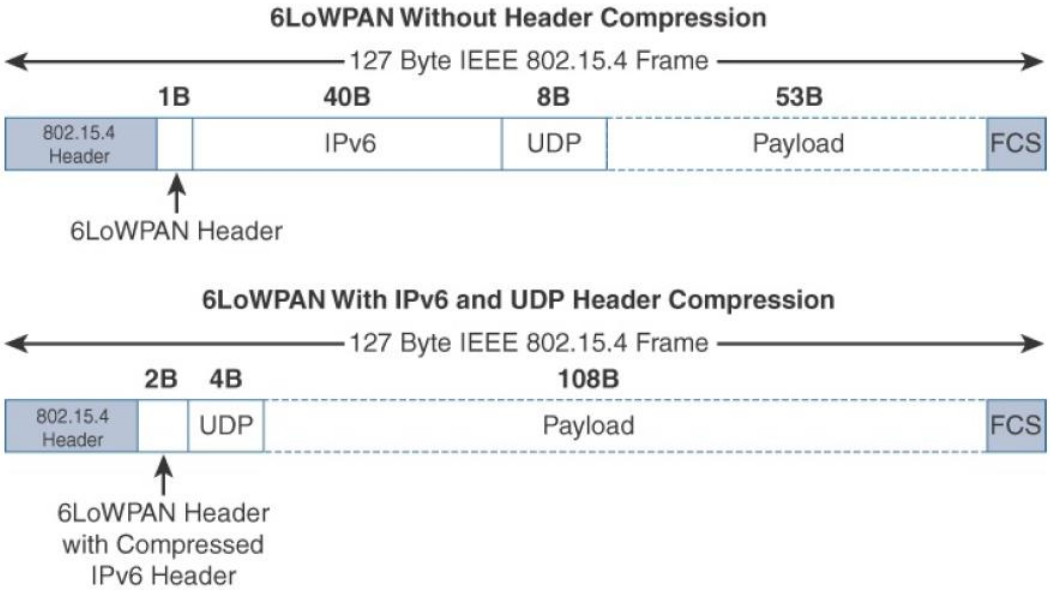


Figure 2.15 6LoWPAN Header Compression

At the top of Figure 2.15, you see a 6LoWPAN frame without any header compression enabled: The full 40- byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only a single byte in this case. Notice that uncompressed IPv6 and UDP headers leave only 53 bytes of data payload out of the 127- byte maximum frame size in the case of IEEE 802.15.4.

The bottom half of Figure 2.15 shows a frame where header compression has been enabled for a best-case scenario. The 6LoWPAN header increases to 2 bytes to accommodate the compressed IPv6 header, and UDP has been reduced in half, to 4 bytes from 8. Most importantly, the header compression has allowed the payload to more than double, from 53 bytes to 108 bytes, which is obviously much more efficient. Note that the 2-byte header compression applies to intra-cell communications, while communications external to the cell may require some field of the header to not be compressed.

Fragmentation

The maximum transmission unit (MTU) for an IPv6 network must be at least 1280 bytes. The term *MTU* defines the size of the largest protocol data unit that can be passed. For IEEE 802.15.4, 127 bytes is the MTU. This is a problem because IPv6, with a much larger MTU, is carried inside the 802.15.4 frame with a much smaller one. To remedy this situation, large IPv6 packets must be fragmented across multiple 802.15.4 frames at Layer 2.

The fragment header utilized by 6LoWPAN is composed of three primary fields: Datagram Size, Datagram Tag, and Datagram Offset. The 1-byte Datagram Size field specifies the total size of the unfragmented payload. Datagram Tag identifies the set of fragments for a payload. Finally, the Datagram Offset field delineates how far into a payload a particular fragment occurs. Figure 2.16 provides an overview of a 6LoWPAN fragmentation header.

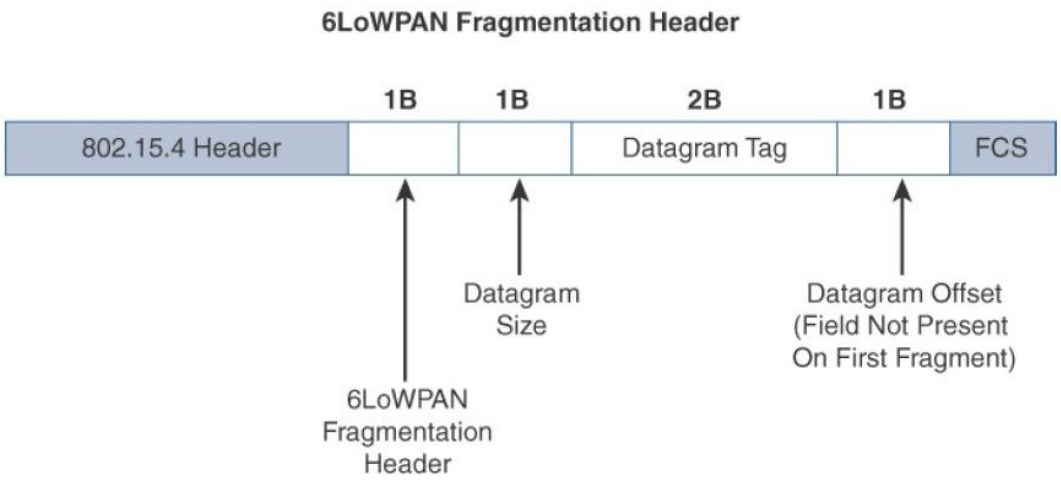


Figure 2.16 6LoWPAN Fragmentation Header

In Figure 2.16, the 6LoWPAN fragmentation header field itself uses a unique bit value to identify that the subsequent fields behind it are fragment fields as opposed to another capability, such as header compression. Also, in the first fragment, the Datagram Offset field is not present because it would simply be set to 0. This results in the first fragmentation header for an IPv6 payload being only 4 bytes long. The remainder of the fragments have a 5-byte header field so that the appropriate offset can be specified.

Mesh Addressing

The purpose of the 6LoWPAN mesh addressing function is to forward packets over multiple hops. Three fields are defined for this header: Hop Limit, Source Address, and Destination Address. Analogous to the IPv6 hop limit field, the hop limit for mesh addressing also provides an upper limit on how many times the frame can be forwarded. Each hop decrements this value by 1 as it is forwarded. Once the value hits 0, it is dropped and no longer forwarded.

The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 addresses indicating the endpoints of an IP hop. Figure 2.17 details the 6LoWPAN mesh addressing header fields.

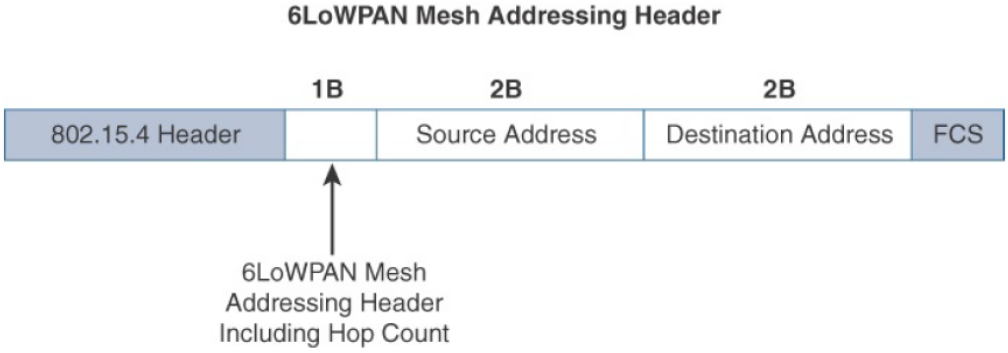


Figure 2.17: 6LoWPAN Mesh Addressing Header

Note that the mesh addressing header is used in a single IP subnet and is a Layer 2 type of routing known as mesh-under. RFC 4944 only provisions the function in this case as the definition of Layer 2 mesh routing specifications was outside the scope of the 6LoWPAN working group, and the IETF doesn't define "Layer 2 routing." An implementation performing Layer 3 IP routing does not need to implement a mesh addressing header unless required by a given technology profile.

Application Transport methods

SCADA

In the world of networking technologies and protocols, IoT is relatively new. Combined with the fact that IP is the de facto standard for computer networking in general, older protocols that connected sensors and actuators have evolved and adapted themselves to utilize IP.

A prime example of this evolution is supervisory control and data acquisition (SCADA). Designed decades ago, SCADA is an automation control system that was initially implemented without IP over serial links, before being adapted to Ethernet and IPv4.

A Little Background on SCADA

For many years, vertical industries have developed communication protocols that fit their specific requirements. Many of them were defined and implemented when the most common networking technologies were serial link-based, such as RS-232 and RS-485. This led to SCADA networking protocols, which were well structured, compared to the other protocols, running directly over serial physical and data link layers.

At a high level, SCADA systems collect sensor data and telemetry from remote devices, while also providing the ability to control them. Used in today's networks, SCADA systems allow global, real-time, data-driven decisions to be made about how to improve business processes.

SCADA networks can be found across various industries, but you find SCADA mainly concentrated in the utilities and manufacturing/industrial verticals. Within these specific industries, SCADA commonly uses certain protocols for communications between devices and applications. For example, Modbus and its variants are industrial protocols used to

2.18. Connection management links the DNP3 layers with the IP layers in addition to the configuration parameters and methods necessary for implementing the network connection. The IP layers appear transparent to the DNP3 layers as each piece of the protocol stack in one station logically communicates with the respective part in the other. This means that the DNP3 endpoints or devices are not aware of the underlying IP transport that is occurring.

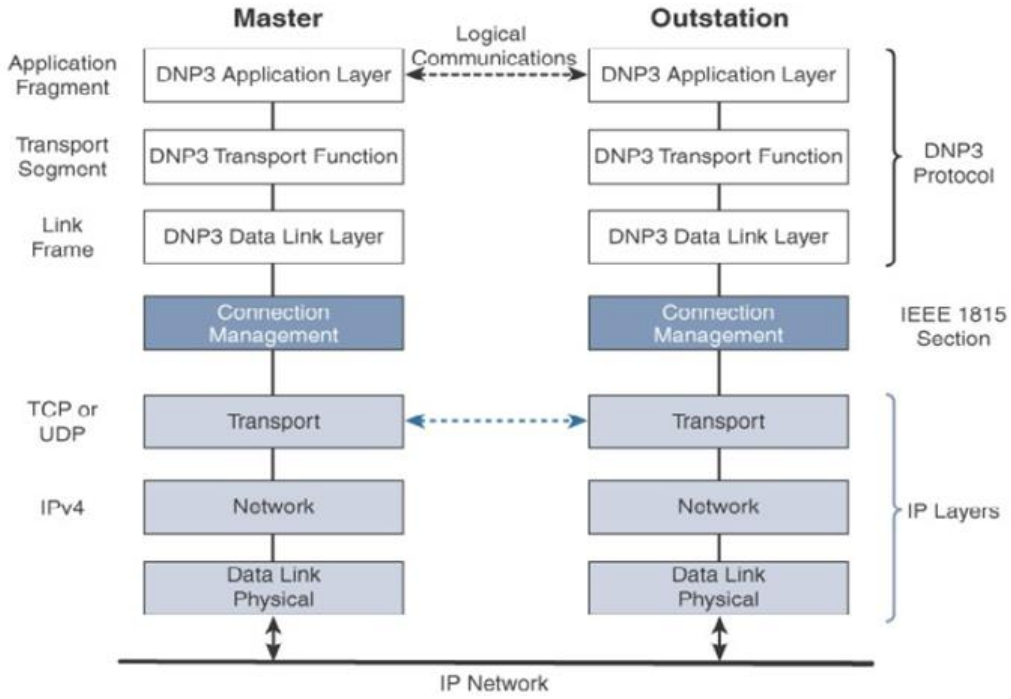


Figure 2.18: Protocol Stack for Transporting Serial DNP3 SCADA over IP

In Figure 2.18, the master side initiates connections by performing a TCP active open. The outstation listens for a connection request by performing a TCP passive open. *Dual endpoint* is defined as a process that can both listen for connection requests and perform an active open on the channel if required. Master stations may parse multiple DNP3 data link layer frames from a single UDP datagram, while DNP3 data link layer frames cannot span multiple UDP data grams. Single or multiple connections to the master may get established while a TCP keepalive timer monitors the status of the connection. Keepalive messages are implemented as DNP3 data link layer status requests. If a response is not received to a keepalive message, the connection is deemed broken, and the appropriate action is taken.

IoT Application Layer Protocols

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. Figure 2.19 highlights their position in a common IoT protocol stack.

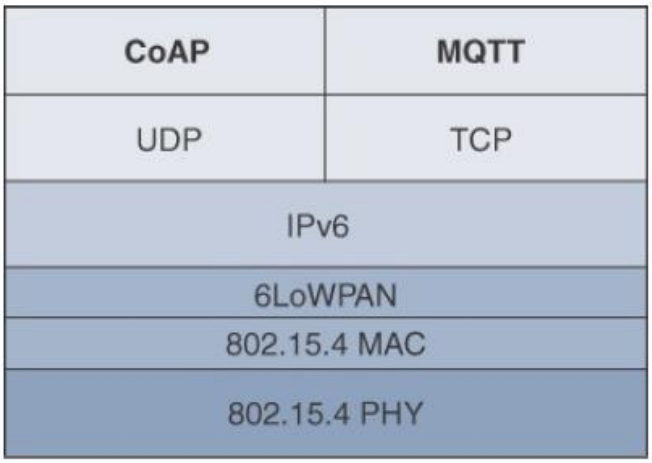


Figure 2.19: Example of a High-Level IoT Protocol Stack for CoAP and MQTT

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

CoAP

Constrained Application Protocol (CoAP) resulted from the IETF Constrained RESTful Environments (CoRE) working group's efforts to develop a generic framework for resource-oriented applications targeting constrained nodes and networks. The CoAP framework 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 over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS).

From a formatting perspective, a CoAP message is composed of a short fixed-length Header field (4 bytes), a variable-length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field. Figure 2.20 details the CoAP message format, which delivers low overhead while decreasing parsing complexity.

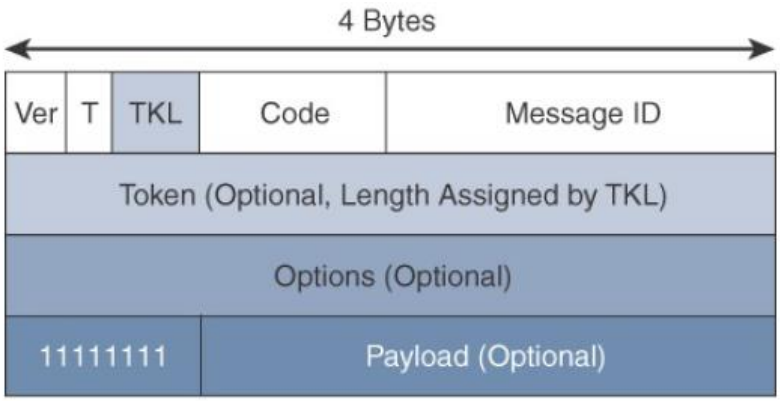


Figure 2.20: CoAP Message Format

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

Table 2.4 CoAP Message Fields

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 within 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 up to 1152 bytes, including 1024 bytes for the payload. In the

case of IPv4, as IP fragmentation may exist across the network, implementations should limit themselves to more conservative values and set the IPv4 Don't Fragment (DF) bit.

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

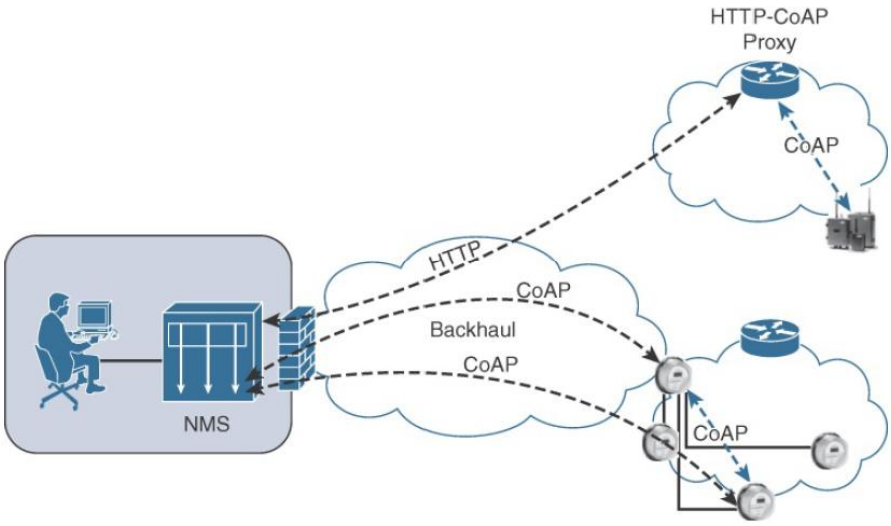


Figure 2.21: CoAP Communications in IoT Infrastructures

Just like HTTP, CoAP is based on the REST architecture, but with a "thing" acting as both the client and the server. Through the exchange of asynchronous messages, a client requests 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 response code that may include a resource representation. The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

Message Queuing Telemetry Transport (MQTT)

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) 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 shown in Figure 2.22

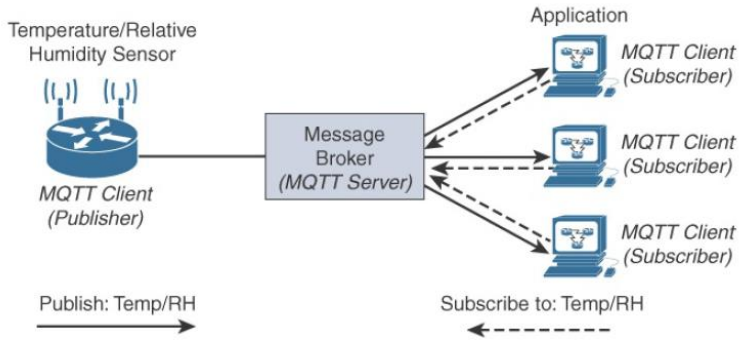


Figure 2.22: MQTT Publish/Subscribe Framework

An MQTT client can act as a publisher to send data (or resource information) to an MQTT 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) sensor that 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. It also handles the subscription and unsubscription process and pushes the application data to MQTT clients 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 specific data 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 run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server. Optionally, MQTT can be secured using TLS on port 8883, and WebSocket (defined in RFC 6455) can also be used.

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

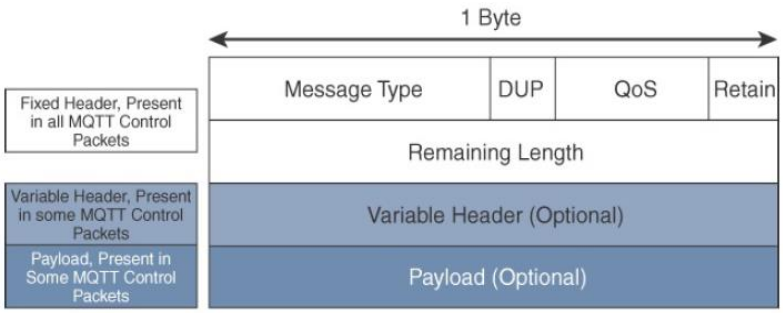


Figure 2.23 *MQTT Message Format*

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 control packets 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 are summarized in Table 2.5.

Table 2.5 MQTT Message Type

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 Remaining Length. This field specifies the number of bytes in the MQTT packet following this field.

MQTT sessions between each client and server consist of four phases: session establishment, authentication, data exchange, and session termination. Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties. When the server is delivering an application message to more than one client, each client is treated independently.

Subscriptions to resources generate SUBSCRIBE/SUBACK control packets, while unsubscription is performed through the exchange of UNSUBSCRIBE/UNSUBACK control packets. Graceful termination of a connection is done through a DISCONNECT control packet, which also offers the capability for a client to reconnect by re-sending its client ID to resume the operations.

A message broker uses a topic string or topic name to filter messages for its subscribers. When subscribing to a resource, the subscriber indicates the one or more topic levels that are used to structure the topic name. The forward slash (/) in an MQTT topic name is used to separate each level within the topic tree and provide a hierarchical structure to the topic names.

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