Internet protocol version 6 (IPv6)

The Internet Protocol Version 6 or IPv6, as it is commonly known, is a resultant of the developments on and beyond IPv4 due to fast depleting address ranges in IPv4. The IPv4 was not designed to handle the needs of the future Internet systems, making it cumbersome and wasteful to use for IoT-based applications. The needs of massive scalability and limited resources gave rise to IPv6, which was developed by the IETF (Internet Engineering Task Force); it is also termed as the Internet version 2 .

Similar to IPv4, IPv6 also works on the OSI layer 3 (network layer). However, in contrast to IPv4 (which is 32 bits long and offers around 4,294,967,296 addresses), IPv6 has a massive logical address range (which is 128 bits long). Additional features in IPv6 include auto-configuration features, end-to-end connectivity, inbuilt security measures (IPSec), provision for faster routing, support for mobility, and many others.

These features not only make IPv6 practical for use in IoT but also makes it attractive for a majority of the present-day and upcoming IoT-based deployments. Interestingly, as IPv6 was designed entirely from scratch, it is not backward compatible; it cannot be made to support IPv4 applications directly. Figure 8.2 shows the differences between IPv4 and IPv6 packet structures.

Some of the important features of IPv6 are as follows:

(i) Larger Addressing Range: IPv6 has roughly four times more addressable bits than IPv4. This magnanimous range of addresses can accommodate the address requirements for any number of connected or massively networked devices in the world.

(ii) Simplified Header Structure: Unlike IPv4, the IPv6 header format is quite simple. Although much bigger than the IPv4 header, the IPv6 header's increased size is mainly attributed to the increased number of bits needed for addressing purposes.

(iii) End-to-End Connectivity: Unlike IPv4, the IPv6 paradigm allows for globally unique addresses on a significantly massive scale. This scheme of addressing enables packets from a source node using IPv6 to directly reach the destination node without the need for network address translations en-route (as is the case with IPv4).

(iv) Auto-configuration: The configuration of addresses is automatically done in IPv6. It supports both stateless and stateful auto-configuration methods and can work even in the absence of DHCP (dynamic host configuration protocol) servers. This mechanism is not possible in IPv4 without DHCP servers.

(v) **Faster Packet Forwarding**: As IPv6 headers have all the seldom-used optional fields at the end of its packet, the routing decisions by a router are taken much faster, by checking only the first few fields of the header.

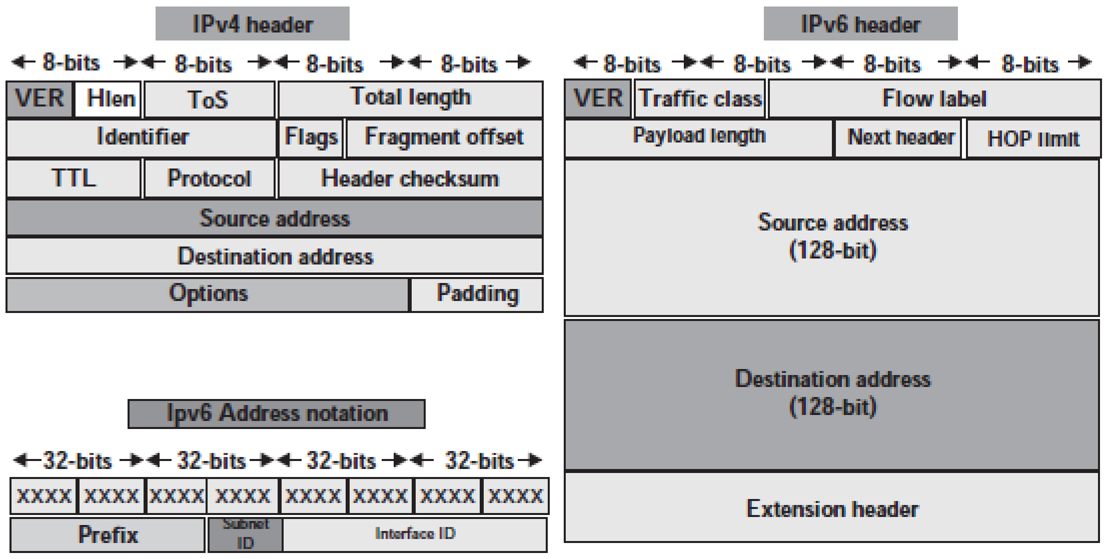
(vi) Inbuilt Security: IPv6 supports inbuilt security mechanisms (IPSec) that IPv4 does not directly support. IPv4 security measures were attained using separate mechanisms in conjunction with IPv4. The present-day version of IPv6 has security as an optional feature.

(vii) Anycast Support: Multiple networking interfaces are assigned the same IPv6 addresses globally; these addresses are known as anycast addresses. This mechanism enables routers to send packets to the nearest available destination during routing.

(viii) Mobility Support: IPv6 has one of the essential features that is crucial for IoT and the modern-day connected applications: mobility support. The mobility support of IPv6 allows for mobile nodes to retain their IP addresses and remain connected, even while changing geographic areas of operation.

(ix) Enhanced Priority Support: The priority support system in IPv6 is entirely simplified as compared to IPv4. The use of traffic classes and flow labels determine the most efficient routing paths of packets for the routers.

(x) Extensibility of Headers: The options part of an IPv6 header can be extended by adding more information to it; it is not limited in size. Some applications may require quite a large options field, which may be comparable to the size of the packet itself.



IPv6 Addressing

The IPv6 addressing scheme has a crucial component: the interface identifier (IID). IID is made up of the last 64 bits (out of the 128 bits) in the IPv6 address. IPv6 incorporates theMAC(media access control) address of the system for IID generation. As a device's MAC address is considered as its hardware footprint and is globally unique, the use of MAC makes IID unique too. The IID is auto-configured by a host using IEEE's extended unique identifier (EUI-64) format. notation. IPv6 supports three types of unicasting: Global unicast address (GUA), link local address (LL), and unique local address (ULA).

The GUA is synonymous with IPv4's static addresses (public IP). It is globally identifiable and uniquely addressable. The global routing prefix is designated by the first (most significant) 48 bits. The first three bits of this routing prefix is always set-to 001; these three bits are also the most significant bits of this prefix. In contrast, LLs are auto-configured IPv6 addresses, whose communication is limited to within a network segment only (under a gateway or a router). The first 16 bits of LL addresses are fixed and equals FE80 in hexadecimal. The subsequent 48 bits are set to 0. As these addresses are not routable, the LLs' scope is restricted to within the operational purview of a router or a gateway. Finally, ULAs are locally global and unique. They are meant for use within local networks only. Packets from ULAs are not routed to the Internet. The first half of an ULA is divided into four parts and the last half is considered as a whole. The four parts of the first part are the following: Prefix, local bit, global ID, and subnet ID, whereas the last half contains the IID. ULA's prefix is always assigned as FD in hexadecimal (1111 110 in binary). If the least significant bit in this prefix is assigned as 1, it signifies locally assigned addresses.

IPv6 Address Assignment

Any node in an IPv6 network is capable of auto-configuring its unique LL address. Upon assigning an IPv6 address to itself, the node becomes part of many multicast groups that are responsible for any communication within that segment of the network. The node then sends a neighbor solicitation message to all its IPv6 addresses. If no reply is received in response to the neighbor solicitation message, the node assumes that there is no duplicate address in that segment, and its address is locally unique. This mechanism is known as duplicate address detection (DAD) in IPv6. Post DAD, the node configures the IPv6 address to all its interfaces and then sends out neighbor advertisements informing its neighbors about the address assignment of its interfaces. This step completes the IPv6 address assignment of a node.

IPv6 Communication

An IPv6 configured node starts by sending a router solicitation message to its network segment; this message is essentially a multicast packet. It helps the node in determining the presence of routers in its network segment or path. Upon receiving the solicitation message, a router responds to the node by advertising its presence on that link. Once discovered, the router is then set as that node's default gateway. In case the selected gateway is made unavailable due to any reason, a new default gateway is selected using the previous steps.

If a router upon receiving a solicitation message determines that it may not be the best option for serving as the node's gateway, the router sends a redirect message to the node informing it about the availability of a better router (which can act as a gateway) within its next hop.

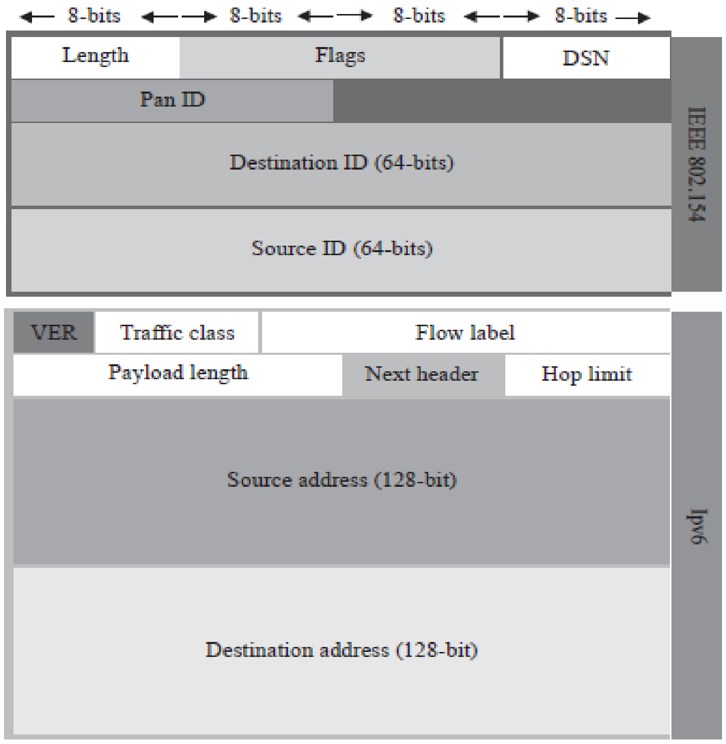
IPv6 Mobility

A mobile IPv6 node located within its home link uses its home address for routing all communication to it. However, when the mobile IPv6 node goes beyond its home link, it has to first connect to a foreign link for enabling communication. A new IPv6 address is acquired from the foreign link, which is also known as the mobile node's care-of-address (CoA). The mobile node now binds its CoA to its home agent (a router/gateway to which the node was registered in its home segment). This binding between the CoA and the home agent is done by establishing a tunnel between them. Whenever the node's home agent receives a correspondence message, it is forwarded to the mobile node's CoA over the established tunnel. Upon receiving the message from a correspondent node, the mobile node may choose not to reply through its
home agent; it can communicate directly to the correspondent node by setting its home address in the packet's source address field. This mechanism is known as route optimization.

6LoWPAN

6LoWPAN allows low power and constrained devices/nodes to connect to the Internet. 6LoWPAN stands for IPv6 over low power wireless personal area networks. As the name suggests, it enables IPv6 support for WPANs, which are limited concerning power, communication range, memory, and throughput . 6LoWPAN is designed to be operational and straightforward over low-cost systems, and extend IPv6 networking capabilities to IEEE 802.15.4-based networks. Popular uses of this protocol are associated with smart grids, M2M applications, and IoT. 6LoWPAN allows constrained IEEE 802.15.4 devices to accommodate 128-bit long IPv6 addresses. This is achieved through header compression, which allows the protocol to compress and retro-fit IPv6 packets to the IEEE 802.15.4 packet format.

6LoWPANnetworks can consist of both limited capability (concerning throughput, processing, memory, range) devices—called reduced function devices (RFD)—and devices with significantly better capabilities, called full function devices (FFD). The RFDs are so constrained that for accessing IP-based networks, they have to forward their data to FFDs in their personal area network (PAN). The FFDs yet again forward the forwarded data from the RFD to a 6LoWPAN gateway in a multi-hop manner. The gateway connects the packet to the IPv6 domain in the communication network. From here on, the packet is forwarded to the destination IP-enabled node/device usingregular IPv6-based networking.



6LoWPAN Stack The 6LoWPAN stack rests on top of the IEEE 802.15.4 PHY and MAC layers, which are generally associated with low rate wireless personal area networks (LR-WPAN). The choice of IEEE 802.15.4 for the base layer makes 6LoWPAN suitable for low power LR-WPANs. The network layer in 6LoWPAN enabled devices (layer 3) serves as an adaptation layer for extending IPv6 capabilities to IEEE 802.15.4 based devices. Figure 8.5 shows the 6LoWPAN packet structure.

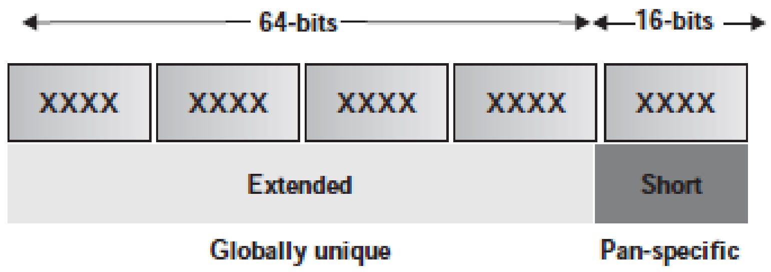
• **PHY and MAC layers**: The PHY layer consists of 27 wireless channels, each having their separate frequency band and varying data rates. The MAC layer defines the means and methods of accessing the defined channels and use them for communication. The LoWPANMAC layer is characterized by the following:

(i) Beaconing tasks for device identification. These tasks include both beacon generation and beacon synchronization.

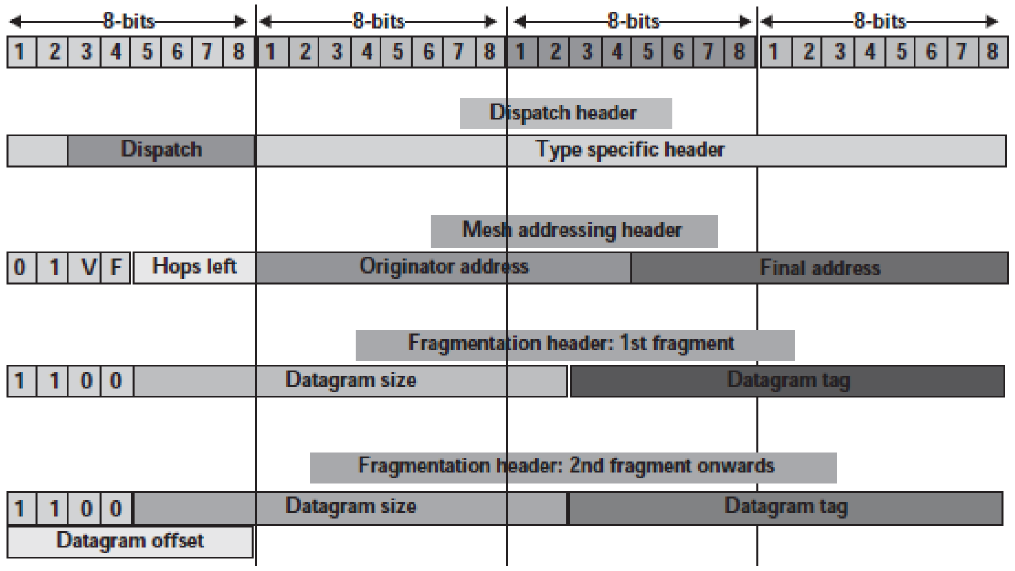
(ii) Channel access control is provided by CSMA/CA.

(iii) PAN membership control functions. Membership functions include association and dissociation tasks.

Adaptation layer: As mentioned previously, 6LoWPAN accommodates and retro-fits the IPv6 packet to the IEEE 802.15.4 packet format. The challenge presented to 6LoWPAN is evident from the fact that IPv6 requires a minimum of 1280 octets for transmission. In contrast, IEEE 802.15.4 can support a maximum of only 1016 octets (127 bytes): 25 octets for frame overheads and 102 octets for payload. Additional inclusion of options in the IEEE 802.15.4 frame, such as security in the headers, leaves only 81 octets for IPv6 packets to use, which is insufficient. Even out of these available 81 octets, the IPv6 header reserves 40 octets for itself, 8 octets for UDP (user datagram protocol), and 20 octets for TCP (transmission control protocol), which are added in the upper layers. This leaves only 13 octets available at the disposal of the upper layers and the data itself. The 6LoWPAN adaptation layer between the MAC and the network layers takes care of these issues through the use of header compression, packet forwarding, and packet fragmentation.



Address Format: The 6LoWPAN address format is made up of two parts: 1) the short (16-bit) address and 2) the extended (64-bit) address. The short address is PAN specific and is used for identifying devices within a PAN only, which makes its operational scope highly restricted and valid within a local network only. In contrast, the globally unique extended address is valid globally and can be used to identify devices, even outside the local network uniquely. Encapsulation Header Formats The encapsulation headers, as the name suggests, defines methods and means by which 6LoWPAN encapsulates the IPv6 payloads within IEEE 802.15.4 frames 6LoWPAN has three encapsulation header types associated with it: dispatch, mesh addressing, and fragmentation. This system is similar to the IPv6 extension headers. The headers are identified by a header type field placed in front of the headers. The dispatch header type is used to initiate communication between a node and a destination node. The mesh addressing header is used for multi-hop forwarding by providing support for layer two forwarding of messages. Finally, the fragmentation header is used to fit large payloads to the IEEE 802.15.4 frame size.

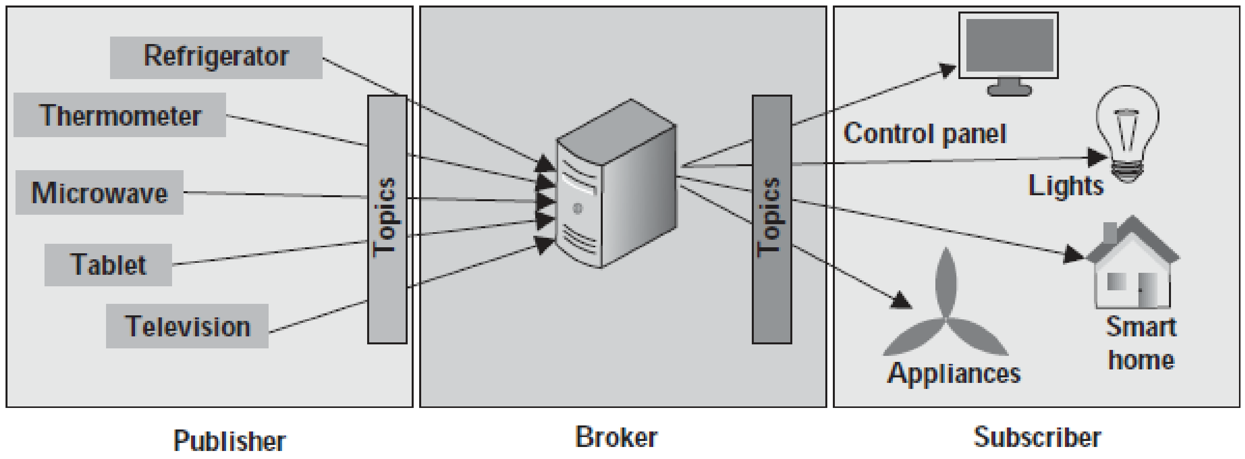


MQTT

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. Figure 8.15 shows the working of MQTT. The MQTT paradigm consists of numerous clients connecting to a server; this server is referred to as a broker. The clients can have the roles of information publishers (sending messages to the broker) or information subscribers (retrieving messages from the broker). This allows MQTT to be largely decoupled from the applications being used with MQTT.

Operational Principle

MQTT is built upon the principles of hierarchical topics and works on TCP for communication over the network. Brokers receive new messages in the form of topics from publishers. A publisher first sends a control message along with the data message. Once updated in the broker, the broker distributes this topic's content to all the subscribers of that topic for which the new message has arrived. This paradigm enables publishers and subscribers to be free from any considerations of the address and ports of multiple destinations/subscribers or network considerations of the subscribers, and vice versa. In the absence of any subscribers of a topic, a broker normally discards messages received for that topic unless specified by the publisher otherwise. This feature removes data redundancies and ensures that maximally updated information is provided to the subscribers. It also reduces the requirements of storage at the broker.

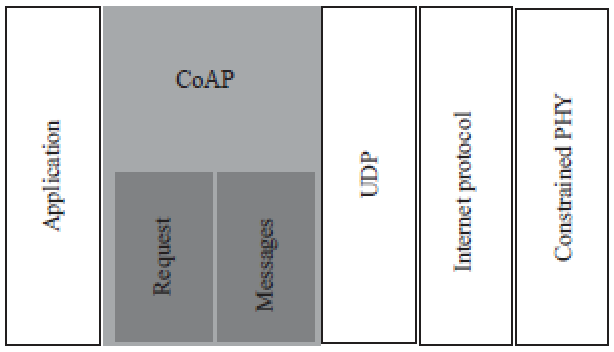


The publishers can set up default messages for subscribers in agreement with the broker if the publisher's connection is abruptly broken with the broker. This arrangement is referred to as the last will and testament feature of MQTT. Multiple brokers can communicate in order to connect to a subscriber's topic if it is not present directly with the subscriber's primary broker. 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.

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 [18]. 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. CoAP follows a request–response paradigm for communication over these lossy networks. Additional highlights of this protocol include support for service discovery, resource discovery, URIs (uniform resource identifier), Internet media handling support, easy HTTP integration, and multicasting support, that too while maintaining low overheads. Typically, CoAP implementations can act as both clients and servers (not simultaneously). A CoAP client's request signifies a request for action from an identified resource on a server, which is similar to HTTP. The response sent by the server in the form of a response code can contain resource representations as well. However, CoAP interchanges are asynchronous and datagram-oriented over UDP. Figure 8.17 shows the placement of CoAP in a protocol stack. Packet traffic collisions are handled by a logical message layer incorporating the exponential backoff mechanism for providing reliability. The reliability feature of CoAP is optional. The two seemingly distinct layers of messaging (which handle the UDP and asynchronous messaging) and request-response (which handles the connection establishment) are part of the CoAP header.



CoAP Features

The CoAP is characterized by the following main features:

(i) It has suitable web protocol for integrating IoT and M2M services in constrained environments with the Internet.

(ii) CoAP enables UDP binding and provides reliability concerning unicast as well as multicast requests.

(iii) Message exchanges between end points in the network or between nodes is asynchronous.

(iv) The limited packet header incurs significantly lower overheads. This also results in less complexity and processing requirements for parsing of packets.

(v) CoAP has provisions for URI and other content-type identifier support. CoAP additionally provides DTLS (datagram transport layer security) binding.

(vi) It has a straightforward proxy mechanism and caching capabilities, which is responsible for overcoming the effects of the lossy network without putting extra constraints on the low-power devices. The caching is based on the concept of the maximum age of packets.

(vii) The protocol provides a stateless mapping with HTTP. The server or receiving node does not retain information about the source of the message; rather, it is expected that the message packet carries that information with it. This enables CoAP's easy and uniform integration with HTTP.

CoAP Messaging

CoAP defines four messaging types: 1) Confirmable (CON), 2) non-confirmable (NON), 3) acknowledgment (ACK), and 4) reset. The method codes and the response codes are included in the messages being carried. These codes determine whether the message is a request message or a response message. Requests are typically carried in confirmable and non-confirmable message types. However, responses are carried in both of these message types as well as with the acknowledgment message. The transmission of responses with acknowledgment messages is known as piggybacking and is quite synonymous with CoAP.

Operational Principle
CoAP is built upon the exchange of messages between two or more UDP end points. Options and payload follow the compact 4-byte binary header in CoAP. This arrangement is typical of request and response messages of CoAP. A 2-byte message ID is used with each message to detect duplicates. Whenever a message is marked as a CON message, it signifies that the message is reliable. In the event of delivery failure of a CON message, subsequent retries are attempted with exponential back-off until the receiving end point receives an ACK with the same message ID . In case the recipient does not have the resources to process the CON message, a RESET message is sent to the originator of the CON message instead of an ACK message.

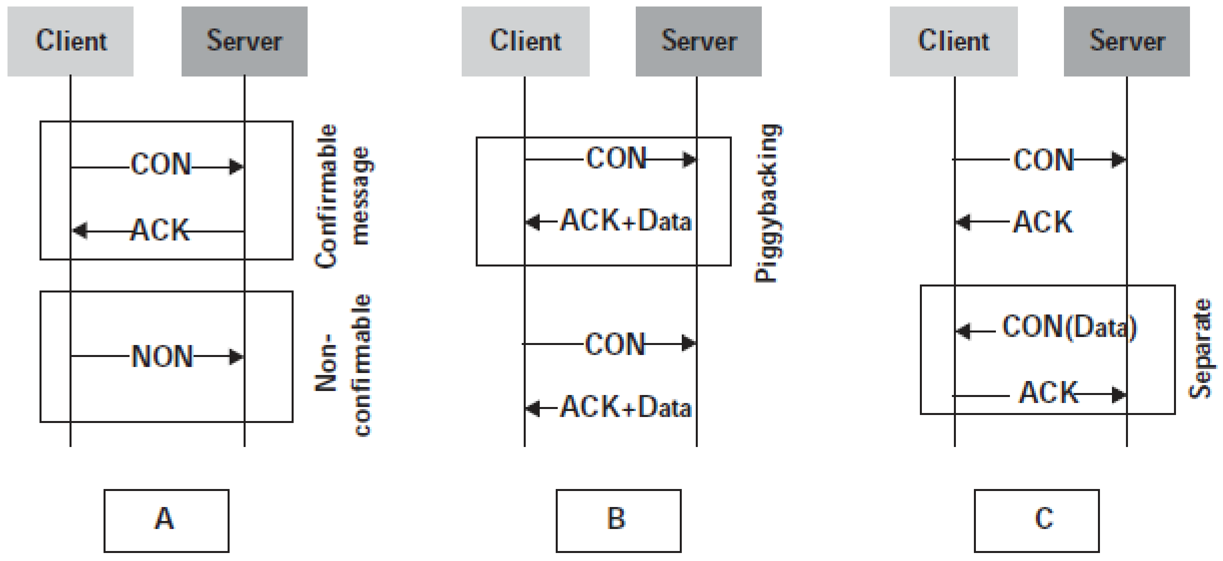


Figure 8.18 Various CoAP response–response models. (A): CON and NON messages, (B): Piggyback messages, and (C): Separate messages

Specific messages, which do not require reliable message transmission (such as rapid temporal readings of the environment from a sensor node), are sent as NON messages. NON messages do not receive an acknowledgment (Figure 8.18). However, the message ID associated with it prevents duplication. NON messages elucidate a IoT Communication Technologies 191 NON or CON response from a server, based on the settings and semantics of the application. If the receiver of the NON cannot process the message, a RESET message is sent to the originator of the NON message. If a server fails to respond immediately to a request received by it in a CON message, an empty ACK response is sent to the requester to stop request retransmissions. Whenever the response is ready, a new CON message is used to respond to the previous request by the client. Here, the client then has to respond to the server using an ACK message. This scheme is known as a separate response.

The multicast support of CoAP over UDP results in multicast CoAP requests. The request and response semantics of CoAP is carried in the form of method and response codes in the CoAP messages itself. The options field of CoAP carries information about the requests and responses such as URI and MIME (multipurpose Internet mail extensions). The concept of tokens is used to match requests with their corresponding responses. The need for a token mechanism arose due to the asynchronous nature of the CoAP messaging. Similar to HTTP, CoAP uses GET, PUT, POST, and DELETE methods.

AMQP

AMQP or the advanced message queuing protocol is an open standard middleware at the application layer developed for message-oriented operations. It tries to bring about the concept of interoperability between clients and the server by enabling cross-vendor implementations. Figure 8.19 shows the various components of AMQP and their relationships. An AMQP broker is tasked with maintaining message queues between various subscribers and publishers. The protocol is armed with features of message orientation, queuing, reliability, security, and routing. Both request–response and publish–subscribe methods are supported. AMQP is considered as a wire-level protocol. Here, the data format description is released on the network as a stream of bytes. This description allows AMQP to connect to anyone who can interpret and create messages in the same format. It also results in a level of interoperability where anyone with compliant or supporting means can make use of this protocol without any need for a specific programming language.

AMQP Features

AMQP is built for the underlying TCP and is designed to support a variety of messaging applications efficiently. It provides a wide variety of features such as flow-controlled communication, message-oriented communication, message delivery guarantees (at most once, at least once, and exactly once), authentication support, and an optional SSL or TLS based encryption support.

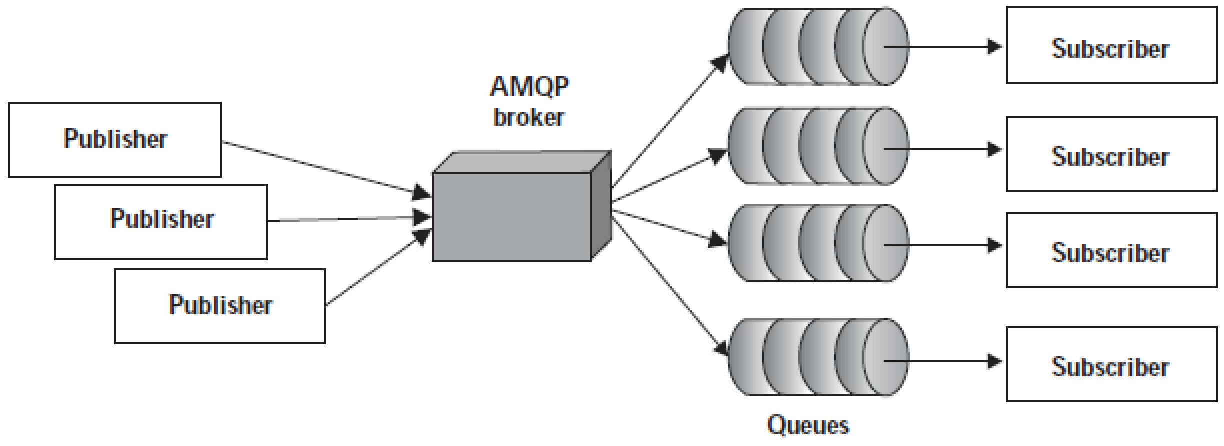


Figure 8.19 AMQP components and their relationships

The AMQP is specified across four layers: 1) type system, 2) process to process asynchronous and symmetric message transfer protocol, 3) extensible message format, and 4) set of extensible messaging capabilities. In continuation, the primary unit of data in AMQP is referred to as a frame. These frames are responsible for the initiation of connections, termination of connections, and control of messages between two peers using AMQP. There are nine frame types in AMQP:

I. Open responsible for opening the connection between peers.
II. Begin: responsible for setup and control of messaging sessions between peers.
III. Attach: responsible for link attachment.

IV. Transfer: responsible for message transfer over the link.
V. Flow: responsible for updating the flow control state.
VI. Disposition: responsible for updating of transfer state.
VII. Detach: responsible for detachment of link between two peers.
VIII. End: responsible for truncation of a session.
IX. Close: responsible for closing/ending a connection.

Operational Principle:

The workings of AMQP revolve around the link protocol. A new link is initiated between peers that need to exchange messages by sending an ATTACH frame. A DETACH frame terminates the link between peers. Once a link is established, unidirectional messages are sent using the TRANSFER frame. Flow control is maintained by using a credit-based flow-control scheme, which protects a process from being overloaded by voluminous messages. Every message transfer state has to be mutually settled by both the sender and the receiver of the message. This settlement scheme ensures reliability measures for messaging in AMQP. Any change in state and settlement of transfer is notified using a DISPOSITION frame. This allows for the implementation of various reliability guarantees. A session can accommodate multiple links in both directions. Unlike the link, a session is bidirectional and sequential. Upon initiation with a BEGIN frame, a session enables a conversation between peers. The session is terminated using an END frame. Multiple logically independent sessions can be multiplexed between peers over a connection. The OPEN frame initiates a connection and the connection is terminated by using a CLOSE frame.

SOAP

SOAP or simple object access protocol is used for exchanging structured information in web services by making use of XML information set formatting over the application layer protocol (HTTP, SMTP) based transmission and negotiation of messages, as shown in Figure 8.21 [21]. This allows SOAP to communicate with two or more systems with different operating systems using XML, making it language and platform independent. The use of SOAP facilitates the messaging layer of the web services protocol stack.

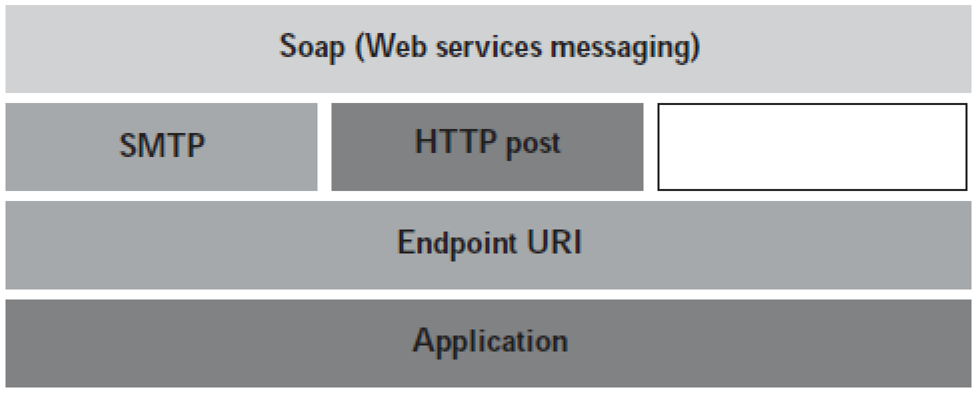


Figure 8.21 A representation of the position of the SOAP API in a stack

A SOAP application can send a request with the requisite search parameters to a server with web services enabled. The target server responds in a SOAP response format with the results of the search. The response from the server can be directly integrated with applications at the requester's end, as it is already in a structured and parsable format. Figure 8.22 illustrates the basic working of SOAP. SOAP is made up of three broad components: 1) Envelope (which defines the structure of the message and its processing instructions), 2) encoding rules (which handles various datatypes arising out of the numerous applications), and

3) convention (which is responsible for web procedure calls and their responses).
This messaging protocol extends the features of neutrality (can operate over any application layer protocol), independence (independent of programming models), and extensibility (features such as security and web service addressing can be extended) to its services. The use of SOAP with HTTP-based request–response
exchanges does not require the modification of the communication and processing infrastructures. It can easily pass through network/system firewalls and proxies (similar to tunneling), as illustrated in Figure 8.22

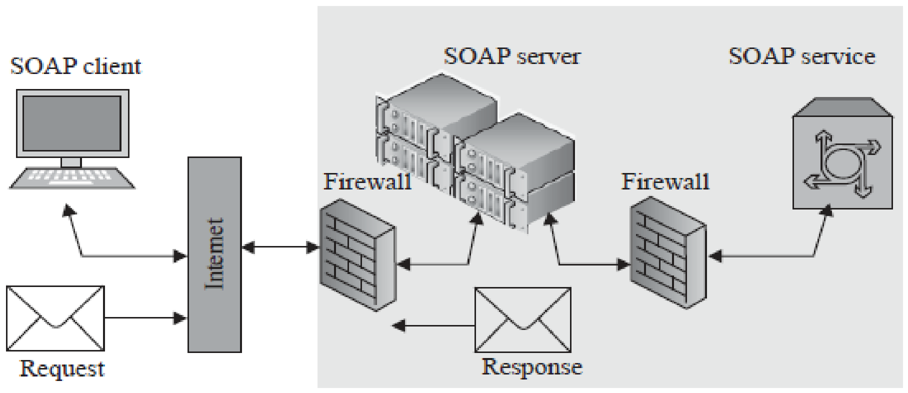


Figure 8.22 Working of SOAP

However, the use of XML affects the parsing speed and hence, the performance of this protocol. Additionally, the verbose nature of SOAP is not recommended for use everywhere. The specifications of the SOAP architecture are defined across several layers, such as message format layer, message exchange patterns (MEP) layer, transport protocol binding layer, message processing model layer, and protocol extensibility layer.

REST

Representational state transfer or REST encompasses a set of constraints for the creation of web services, mainly using a software architectural style. The web services adhering to REST styles are referred to as RESTful services; these services enable interoperability between various Internet-connected devices. RESTful systems are stateless: the web services on the server do not retain client states. The use of stateless protocols and standards makes RESTful systems quite fast, reliable, and scalable. The reuse of components can be easily managed without hindering the regular operations of the system as a whole. Requesting systems can manipulate textual web resource representations by making use of this stateless behavior of REST. RESTful web services, in response to requests made to a resource’s URI, mainly responds with either an HTML, XML, or JSON (JavaScript Object Notation) formatted payload. As RESTful services use HTTP for transfer over the network, the following four methods are commonly used: 1) GET (read-only access to a resource), 2) POST (for creating a new resource), 3) DELETE (used for removing a resource), and 4) PUT (used for updating an existing resource or creating a new one). Figure 8.23 represents the REST style and its components.
REST offers several advantages over regular web-based services. Enhanced network efficiency through the use of REST is ensured by an increase in the performance of interaction between components. Its use also enables a uniform and simple interface, easy live operational modification capabilities, reliability against
component and data failures, portability of components, robust scalability, and support for a large number of components. In REST, requests are used for identifying individual resources. As the

resources can be represented in a variety of formats such as HTML, XML, JSON, and others, RESTful services can identify the individual resources from their representations, which allows them to modify, update, or delete these resources.

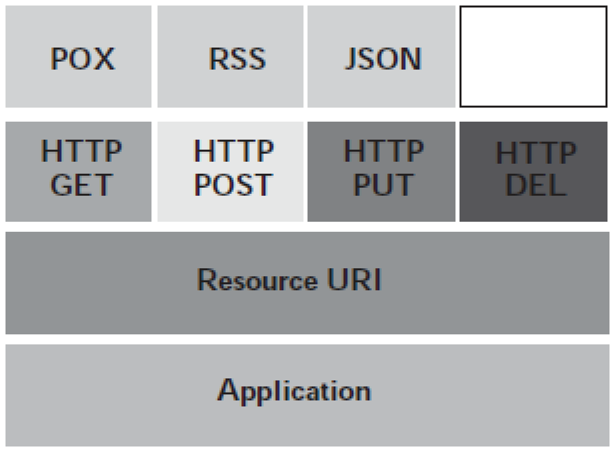


Figure 8.23 A representation of the REST style and its components

The REST messages contain sufficient information in them to direct a parser on how to interpret the messages. REST client's can dynamically discover all web resources and actions associated with an initial URI. This enhances the dynamicity of applications using REST by avoiding the need to hard-code all clients with the information of the proper structure or dynamics of the web application. RESTful systems are guided by six general constraints, which define and restrict the process of client–server interactions and requests–responses. These guidelines increase system performance, scalability, reliability, modifiability, portability, and visibility. All RESTful systems have to adhere to these six guidelines strictly:

(i) **Statelessness**: The statelessness of the client–server communication prevents the storage of any contextual information of the client on the server. Each client request has to be self-sufficient in informing its responders about its services and session state. This is done by including the possible links for new state transitions within the representation of each application state. Generally, upon detecting pending requests, the server infers that the client is in a state of transition.

(ii) **Uniform Interface**: Each part or component of a RESTful system must evolve independently as a result of the decoupling of architectures and its simplification.

(iii) **Cacheability**: The responses have to be implicitly, or in some cases, explicitly clear on whether they have to be cached or not. This helps the clients in retaining the most updated data in response to requests. Caching also reduces the number of client–server interactions, thereby improving the performance and scalability of the system as a whole.

(iv) **Client–server Architecture**: The user–interface interactions should be separate from data storage ones. This would result in enhanced portability of user interfaces across multiple platforms. The separation also allows for the independent evolution of components, which would result in scalability over the Internet across various organizational domains.

(v) **Layered System**: The client in RESTful services is oblivious to the nature of the server to which it is connected: an end point server or an intermediary server. The use of intermediaries also helps in improving the balancing of load and enhancing security measures and system scalability.

(vi) **Code on Demand**: This is an optional parameter. Here, the functionality of clients can be extended for a short period by the server. For example, the transfer of executable codes from compiled components.